Universidad de Oviedo

ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN

MÁSTER UNIVERSITARIO EN INGENIERÍA INFORMÁTICA

ÁREA DE CIENCIA DE LA COMPUTACIÓN E INTELIGENCIA ARTIFICIAL

ANEXO I: CÓDIGO

Aprendiendo perfiles de usuario: distinción entre intereses consolidados e intereses recientes

Pérez Núñez, Pablo

TUTOR: D. Jorge Díez Peláez

FECHA: 11 de julio de 2018

Índice

1. Introducción

A continuación se incluirán cada uno de los ficheros de código realizados para llevar a cabo la experimentación. Para mayor comodidad, se han subido todos estos ficheros a un repositorio público:

`https://github.com/uo232368/TFM`.

Destacar la ausencia de los datos dentro del repositorio debido al gran tamaño de los mismos.

2. Clase LastFM

Esta clase contiene procedimientos para llevar a cabo la limpieza y gestión de datos además de otros métodos genéricos utilizados a lo largo de la investigación.

```
# -*- coding: utf-8 -*-
import pandas as pd
import numpy as np
import scipy.sparse as sp


import sys, time, os
import sqlite3 as lite
import fileinput, csv
from ggplot import *
import collections
from difflib import SequenceMatcher
import pickle
from datetime import datetime, timedelta
import re

import tensorflow as tf

class bcolors:
    HEADER = '\033[95m'
    OKBLUE = '\033[94m'
    OKGREEN = '\033[92m'
    WARNING = '\033[93m'
    FAIL = '\033[91m'
    ENDC = '\033[0m'
    BOLD = '\033[1m'
    UNDERLINE = '\033[4m'

class LastFM:

    DATA_PATH = "/Users/pablo/Desktop/W2VEC/Datasets/lastfm"
    DATABASE_NAME = "LastFM.sqlite"

    ##########################
    # Metodos
    ##########################

    def __init__(self):
        self.DATA_PATH = "datos/"

        #reload(sys)
        #sys.setdefaultencoding('utf8')

        pd.set_option('display.max_rows', 10)
        pd.set_option('display.max_columns', 500)
        pd.set_option('display.width', 1000)

    def doQuery(self,SQL):

        con = lite.connect(self.DATA_PATH+"/"+self.DATABASE_NAME)
        con.text_factory = lambda x: str(x, 'utf-8', 'ignore')
        dft = pd.read_sql_query(SQL, con)
```

```
53          return dft
54
55      def createDict(self, pickleName="datos/DUP_ID_DICT", similarity=0.95):
56
57          '''
58          Este metodo genera un diccionario donde, para cada nombre de cancion diferente, se le
                asigna un ID.
59          Si existen repetidos , les asigna a todos el mismo id, de forma que (a,b,b',b',b',c)
                tendrian ids (0,1,1,1,1,2)
60
61          Funcionamiento basico.·
62            Para cada artista, se obtiene su lista de canciones·
63            Se compara cada cancion con el resto y, si existen similares se les asigna a todas
                  (las similares y la que se compara) el mismo id.
64
65          Posee alguna optimizacion para reducir el orden de complejidad a logaritmico.
66
67          Los resultados se almacenan en dos pickles:·
68               DUP_ID_DICT => Diccionario con clave = Nombre de cancion y valor= ID·
69               DUP_ID_DICT_INV => Diccionario anterior inevertido (clave=valor,valor=clave)
70
71          Genera tambien los pickles:·
72               ALL_SONGS => Todas las canciones (incluidas repeticiones)·
73               ALL_SONGS_UNIQUE => Las canciones sin repeticion·
74               ALL_SONGS_ID_COUNT => Todas las canciones con los id generados y el numero de
                     veces que aparece cada cancion
75
76
77          :param: Grado de similitud deseado.
78          :return: Diccionario con, nombre de cancion => Id asignado. Ademas se guarda en un
                pickle
79          '''
80
81          ########################
82
83          def similar(a, b):
84              # Metodo para ver si 2 strings son parecidos. Retorna porcentaje en tanto por uno
85              return SequenceMatcher(None, str(a), str(b)).ratio()
86
87          def checkDuplicates(CURR_ID, SONG_LIST, ARTIST):
88              # A partir de la lista de canciones, se encarga de detectar duplicados e
                    incremetar el id actual
89              # el cual retorna para continuar en la funcion principal.
90
91              AVOID_LIST = []
92              RES = {}
93
94              TEMP_LIST = SONG_LIST
95              i = 0
96
97              regex = r"(\d)"
98
99              for s in SONG_LIST:
100                 TEMP_LIST = TEMP_LIST[1:]
101
102                 SIMILARS = []
103
```

```
104             # Si ya se visito en una iteracion anterior nada
105             if s in AVOID_LIST:
106                 continue
107 
108             # Buscar items similares del actual
109             for s2 in TEMP_LIST:
110                 SML = similar(s, s2)
111 
112                 # Si ya se visito en una iteracion anterior nada
113                 if s2 in AVOID_LIST:
114                     continue
115 
116                 # Si son muy similares
117                 if (SML > similarity):
118 
119                     m1 = re.finditer(regex, s)
120                     m2 = re.finditer(regex, s2)
121                     m1_items = []
122                     m2_items = []
123                     arequal = True
124 
125                     for matchNum, match in enumerate(m1):
126                         m1_items.append(match)
127                     for matchNum, match in enumerate(m2):
128                         m2_items.append(match)
129 
130                     if (len(m1_items) == len(m2_items)):
131                         for i in range(len(m2_items)):
132                             arequal = m1_items[i].group() == m2_items[i].group()
133 
134                             if not arequal: break
135                     else:
136                         arequal = False
137 
138                     # Si son iguales
139                     if (arequal):
140                         SIMILARS.append(s2)
141                         AVOID_LIST.append(s2)
142 
143             i += 1
144 
145             # Si el item posee repetidos, asignar el mismo id a todos
146             if (len(SIMILARS) > 0):
147                 SIMILARS.append(s)
148                 for i in range(len(SIMILARS)):
149                     RES.update({str(ARTIST) + " - " + str(SIMILARS[i]): CURR_ID});
150                 CURR_ID += 1;
151 
152             else:
153                 SIMILARS.append(s)
154                 RES.update({str(ARTIST) + " - " + str(s): CURR_ID})
155                 CURR_ID += 1;
156 
157         if (len(RES) != len(SONG_LIST)):
158             print("Algo va mal...")
159 
160         return RES, CURR_ID
```

```
161
162         def getID(s, DICT):
163             id = DICT.get(s.longname, None)
164             if (id is None):
165                 return (-1)
166             else:
167                 return (id)
168
169         ##########################
170
171         # Si no existe el diccionario, generar
172         if os.path.isfile(pickleName + ".pkl"):
173             print("Ya existe un diccionario previo...")
174             return()
175
176         if not os.path.isfile("datos/ALL_SONGS.pkl") and not
                os.path.isfile("datos/ALL_SONGS_UNIQUE.pkl"):
177             print("No existen los pickles:\n\t · ALL_SONGS.pkl\n\t · ALL_SONGS_UNIQUE.pkl")
178
179             ALL_SONGS = self.doQuery("select userid,timestamp,artname,traname from
                    user_track")
180             ALL_SONGS.artname = ALL_SONGS.artname.astype(str)
181             ALL_SONGS.traname = ALL_SONGS.traname.astype(str)
182             ALL_SONGS["longname"] = ALL_SONGS.artname + " - " + ALL_SONGS.traname
183             ALL_SONGS.to_pickle("datos/ALL_SONGS.pkl");
184
185             ALL_SONGS_UNIQUE = ALL_SONGS.drop_duplicates("longname")
186             ALL_SONGS_UNIQUE = ALL_SONGS_UNIQUE.sort_values("longname")
187             ALL_SONGS_UNIQUE.to_pickle("datos/ALL_SONGS_UNIQUE.pkl");
188
189         else:
190             ALL_SONGS_UNIQUE = pd.read_pickle("datos/ALL_SONGS_UNIQUE.pkl")
191             ALL_SONGS = pd.read_pickle("datos/ALL_SONGS.pkl")
192
193         ##########################
194
195         CURR_ID = 0
196
197         RES = {}
198
199         ARTISTS = ALL_SONGS_UNIQUE.groupby("artname")
200
201         for i, g in ARTISTS:
202             ARTIST = i
203             # Si hay mas de una cancion para el artista, hay que ver si se repiten.
204             if (len(g) > 1):
205                 # Comparar los nombres de las canciones buscando repetidos
206                 PARTIAL_RES, CURR_ID = checkDuplicates(CURR_ID, g.traname.values, ARTIST)
207                 RES.update(PARTIAL_RES)
208
209             # Si solo hay una cancion para el artista, se le asigna el numero.
210             else:
211                 RES.update({str(g.longname.values[0]): CURR_ID})
212                 CURR_ID += 1
213             print(len(RES))
214
215         #---------------------------
```

```
216         # Crear diccionario con "Para cada string, el id al que se refiere" y el inverso
217         #-----------------------------
218 
219         self.save_pickle(RES, pickleName)
220 
221         # Crear el diccionario inverso
222         '''ESTE DICCIONARIO TENDRA MENOS ITEMS DADO QUE SE ELIMINAN LAS CLAVES REPETIDAS'''
223 
224         INV_DICT = dict(zip(RES.values(), RES.keys()))
225         self.save_pickle(INV_DICT, pickleName + "_INV")
226 
227         #-----------------------------
228         # Anadir al dataframe el ID y el numero de reproducciones
229         #-----------------------------
230         if not os.path.isfile("datos/ALL_SONGS_ID_COUNT.pkl"):
231 
232             # Anadir a cada cancion su id
233             ID_COL = ALL_SONGS.apply(lambda row: getID(row, RES), axis=1)
234             ALL_SONGS_ID = pd.concat([ALL_SONGS, ID_COL], axis=1)
235             ALL_SONGS_ID.columns = ["user", "timestamp",'artname', 'traname', 'longname',
                    'ID']
236 
237             ID_COUNT = ALL_SONGS_ID.groupby(['ID']).size().reset_index(name='counts')
238             RES = pd.merge(ALL_SONGS_ID, ID_COUNT, how='left', on=['ID'])
239             RES.timestamp = pd.to_datetime(RES.timestamp, format="%Y-%m-%dT%H:%M:%SZ")
240             RES.to_pickle("datos/ALL_SONGS_ID_COUNT.pkl");
241         else:
242             print("Fichero ya existente...")
243 
244         #-----------------------------
245 
246     def generaFicherosUsuario(self, filename):
247         '''
248         A partir de los datos (YA FILTRADOS) , genera para cada usuario un fichero con las
                canciones escuchadas,
249         en orden y representadas por su id
250         :param: Diccionario con "Titulo"=>ID
251         '''
252 
253         DATA = pd.read_pickle(filename)
254 
255         # Obtener los id de usuarios
256         USER_LIST = DATA.groupby("user")
257 
258         #Para cada usuario
259         for u,r in USER_LIST:
260 
261             #Imprimir ID
262             self.printB(u)
263 
264             # Obtener todas las canciones del usuario
265             USER_SONG_LIST = r
266 
267             # Ordenar por fecha
268             USER_SONG_LIST.sort_values("timestamp", ascending=True, inplace=True)
269             USER_SONG_LIST = USER_SONG_LIST.reset_index()
270 
```

```
271             # Lista con los id
272             ID_LIST = USER_SONG_LIST.ID.values
273 
274             # Crear fichero con canciones para cada uno de los usuarios (Separado por ;)
275             file = open("datos/usuarios/" + str(u) + ".txt", "w")
276             file.write(";".join(str(x) for x in ID_LIST))
277             file.close()
278 
279     def
        getCleanData(self,songMin=10,userMin=100,years=[2005,2006,2007,2008,2009],saveAs="MAIN_DATA"):
280 
281         if os.path.isfile("datos/ALL_SONGS_ID_COUNT.pkl"):
282             ALL_SONGS_ID_COUNT = pd.read_pickle("datos/ALL_SONGS_ID_COUNT.pkl")
283         else:
284             print("No existen el pickle:\n\t · ALL_SONGS_ID_COUNT.pkl [generado en
                createDict()]")
285             return ()
286 
287         INIT_LEN = len(ALL_SONGS_ID_COUNT)
288 
289         ALL_SONGS_ID_COUNT["year"] = ALL_SONGS_ID_COUNT.timestamp.dt.year.astype(int)
290 
291         print("—"*70)
292         #----------------------------
293         print(" · Eliminar datos que NO sean de los anos " +(',').join(str(x) for x in
            years))+".")
294         ALL_SONGS_ID_COUNT = ALL_SONGS_ID_COUNT.loc[ALL_SONGS_ID_COUNT.year.isin(years)]
295 
296         #----------------------------
297         print(" · Eliminar canciones con menos de " + str(songMin) +" rerpoducciones.")
298 
299         #print(len(ALL_SONGS_ID_COUNT.loc[ALL_SONGS_ID_COUNT.counts < songMin]))
300         ALL_SONGS_ID_COUNT = ALL_SONGS_ID_COUNT.loc[ALL_SONGS_ID_COUNT.counts >=songMin]
301 
302         #----------------------------
303         print(" · Eliminar usuarios con menos de " + str(userMin) +" rerpoducciones.")
304 
305         USERS = ALL_SONGS_ID_COUNT.groupby(["user"])
306         BAD_USERS=[]
307 
308         for u,d in USERS:
309             if len(d)<userMin: BAD_USERS.append(u)
310 
311 
312         ALL_SONGS_ID_COUNT = ALL_SONGS_ID_COUNT.loc[~ALL_SONGS_ID_COUNT.user.isin(BAD_USERS)]
313 
314 
315         #----------------------------
316         #----------------------------
317 
318         '''
319         Es necesario anadir unos nuevos ID para facilitar el OneHot y eliminar los no
            utilizados
320         '''
321 
322         #Se crea un diccionario con los IDs actuales (con saltos) y el correspondiente ID
            nuevo
```

```
323         key = ALL_SONGS_ID_COUNT.ID.unique().tolist()
324         value = range(0, len(key))
325         dict = dict(zip(key, value))
326 
327         #Eliminar vieja columna y anadir la nueva
328         ID_NEW = ALL_SONGS_ID_COUNT.ID.apply(lambda row: dict.get(row))
329         ALL_SONGS_ID_COUNT = ALL_SONGS_ID_COUNT.drop(['ID'], axis=1)
330         ALL_SONGS_ID_COUNT = pd.concat([ALL_SONGS_ID_COUNT, ID_NEW], axis=1)
331 
332         #----------------------------
333 
334 
335         print("-"*70)
336 
337         print("Reproducciones iniciales: "+str(INIT_LEN))
338         print("Reproducciones eliminadas: "+str(INIT_LEN-len(ALL_SONGS_ID_COUNT)))
339         print("Usuarios eliminados: "+str(len(BAD_USERS))+"\n\t" + ','.join(BAD_USERS))
340 
341                 ALL_SONGS_ID_COUNT.to_pickle(saveAs+".pkl");
342 
343 
344 
345 
346         return ALL_SONGS_ID_COUNT
347 
348     ##########################
349     # Metodos W2V
350     ##########################
351 
352     def getSongDataArrays(self, train=75):
353         '''
354         Retorna un array de arrays [[cancion1,cancion2... ],[cancion1, cancion2...]...]
355         :param: Array de arrays
356         '''
357 
358         filename = "datos/CLEAN_DATA_[10,100].pkl"
359 
360         DATA = pd.read_pickle(filename)
361 
362         # Obtener los id de usuarios
363         USER_LIST = DATA.groupby("user")
364 
365         RES_TRAIN=[]
366         RES_REC=[]
367 
368         RES_USERS=[]
369 
370         #Para cada usuario
371         for u,r in USER_LIST:
372 
373             # Obtener todas las canciones del usuario
374             USER_SONG_LIST = r
375 
376             # Ordenar por fecha
377             USER_SONG_LIST.sort_values("timestamp", ascending=True, inplace=True)
378             USER_SONG_LIST = USER_SONG_LIST.reset_index()
379 
```

```
380             # Lista con los id
381             ID_LIST = USER_SONG_LIST.ID.values.tolist()
382             # Lista con las fechas
383 
384             # Separar para embeddigns y para recomendacion
385             TRAIN_INDX = int(len(ID_LIST)*(train/100))
386 
387             TRAIN_LIST = ID_LIST[:TRAIN_INDX]
388             REC_LIST = ID_LIST[TRAIN_INDX:]
389 
390             #Pasar a string las listas
391             TRAIN_LIST = list(map(str, TRAIN_LIST))
392             REC_LIST = list(map(str, REC_LIST))
393 
394             RES_TRAIN.extend(TRAIN_LIST)
395             RES_REC.extend(REC_LIST)
396 
397             RES_USERS.append(u)
398 
399         self.save_pickle(RES_TRAIN,"datos/Extend/ARRAY_ARRAYS_TRAIN")
400         self.save_pickle(RES_REC,"datos/Extend/ARRAY_ARRAYS_REC")
401 
402         return(RES_TRAIN,RES_REC,RES_USERS)
403 
404     def getUserDataArrays(self, train=75 , days=30, minSongs=5):
405         '''
406         Genera los datos para el metodo D2V.
407 
408         :param train: Porcentaje de datos para TRAIN
409         :param days: Numero de dias para el perfil reciente
410         :param minSongs: Numero minimo de canciones que ha de tener un usuario en su perfil
                reciente para no ser eliminado
411         :return: Retorna datos de TRAIN COMPLETOS y RECIENTES y datos de DEV y TEST
412         '''
413 
414         filename = "datos/CLEAN_DATA_[10,100].pkl"
415 
416         DATA = pd.read_pickle(filename)
417 
418         # Obtener los id de usuarios
419         USER_LIST = DATA.groupby("user")
420 
421         RES_TRAIN_PRESENT=[]
422         RES_TRAIN_COMPLETE=[]
423         RES_REC=[]
424 
425         RES_USERS=[]
426 
427         #Para cada usuario
428         for u,r in USER_LIST:
429 
430             # Obtener todas las canciones del usuario
431             USER_SONG_LIST = r
432 
433             # Ordenar por fecha
434             USER_SONG_LIST.sort_values("timestamp", ascending=True, inplace=True)
435             USER_SONG_LIST = USER_SONG_LIST.reset_index()
```

```
436
437             # Lista con los id y las fechas
438             ID_LIST = USER_SONG_LIST.ID.values.tolist()
439             DATES_LIST = USER_SONG_LIST.timestamp.values.tolist()
440             DATES_LIST = np.array(list(map(lambda x: datetime.utcfromtimestamp(x /
                    1000000000), DATES_LIST)))
441
442             # Separar TRAIN DEV y TEST
443             TRAIN_INDX = int(len(ID_LIST)*(train/100))
444
445             TRAIN_LIST = ID_LIST[:TRAIN_INDX]
446             REC_LIST = ID_LIST[TRAIN_INDX:]
447             DATES_TRAIN = DATES_LIST[:TRAIN_INDX]
448
449             # Obtener X dias antes
450             PRESENT_START = DATES_TRAIN[-1] - timedelta(days=days)
451             PAST_LENGTH = sum((DATES_TRAIN >= PRESENT_START) == False)
452             PRESENT_ITEMS = TRAIN_LIST[PAST_LENGTH:]
453
454             #Si el usuario no posee suficientes canciones en el perfil presente, no se anade
455             if(len(PRESENT_ITEMS)<=minSongs):
456                 print("Skiping "+str(u))
457                 continue
458
459             #Pasar a string las listas
460             PRESENT_LIST = list(map(str,PRESENT_ITEMS))
461             TRAIN_LIST = list(map(str, TRAIN_LIST))
462             REC_LIST = list(map(str, REC_LIST))
463
464             RES_TRAIN_COMPLETE.append(TRAIN_LIST)
465             RES_TRAIN_PRESENT.append(PRESENT_LIST)
466             RES_REC.append(REC_LIST)
467
468             RES_USERS.append(u)
469
470         print(len(RES_TRAIN_COMPLETE),len(RES_USERS))
471
472         self.save_pickle(RES_TRAIN_COMPLETE,"datos/Append/ARRAY_ARRAYS_TRAIN_COMPLETE")
473         self.save_pickle(RES_TRAIN_PRESENT,"datos/Append/ARRAY_ARRAYS_TRAIN_PRESENT")
474         self.save_pickle(RES_REC,"datos/Append/ARRAY_ARRAYS_REC")
475
476
477         return(RES_TRAIN_COMPLETE,RES_TRAIN_PRESENT,RES_REC,RES_USERS)
478
479     def getSkipGramData(self, window=2):
480
481         filename = "datos/ARRAY_ARRAYS_EXTEND"
482
483         if os.path.isfile(filename+".pkl"):
484             DATA = self.load_pickle(filename)
485             DATA = self.save_pickle(filename)
486
487         else:
488             print("No existe el pickle, creando...")
489             DATA = self.getDataArrays();
490
491         DATA = list(map(int, DATA))
```

```
492         LENGTH = max(DATA)+1
493 
494         RES_LEN = (len(DATA)-(window*2))*(window*2)
495 
496         batch_inputs = np.ndarray(shape=(RES_LEN), dtype=np.int32)
497         batch_context = np.ndarray(shape=(RES_LEN, 1), dtype=np.int32)
498 
499         j=0
500 
501         for i in range(window,len(DATA)-window):
502             for w in range(1,window+1):
503 
504                 batch_inputs[j] = DATA[i]
505                 batch_context[j,0] = DATA[i-w]
506                 j+=1
507 
508                 batch_inputs[j] = DATA[i]
509                 batch_context[j, 0] = DATA[i+w]
510                 j+=1
511 
512         return(batch_inputs,batch_context,LENGTH)
513 
514     ##########################
515     # Estadisticas e info acerca de los datos
516     ##########################
517 
518     def getPlaysByUser(self, times = list(range(10,110,10)) ):
519 
520         if os.path.isfile("datos/ALL_SONGS_ID_COUNT.pkl"):
521             ALL_SONGS_ID_COUNT = pd.read_pickle("datos/ALL_SONGS_ID_COUNT.pkl")
522 
523         else:
524             print("No existen el pickle:\n\t · ALL_SONGS_ID_COUNT.pkl [generado en
                    createDict()]")
525             return()
526 
527         #Obtener los id de usuarios
528         USER_LIST = ALL_SONGS_ID_COUNT.groupby("user")
529 
530 
531         print('\t'.join(str(x) for x in ["user",1]+times))
532 
533         for u, s in USER_LIST:
534             COUNTS = [len(s)]
535             for i in times: COUNTS.append(len(s.loc[s.counts>=i]))
536 
537             COUNTS = [u]+COUNTS
538             print('\t'.join(str(x) for x in COUNTS))
539 
540     def getPlaysByYear(self):
541 
542         if os.path.isfile("datos/ALL_SONGS_ID_COUNT.pkl"):
543             ALL_SONGS_ID_COUNT = pd.read_pickle("datos/ALL_SONGS_ID_COUNT.pkl")
544 
545         else:
546             print("No existen el pickle:\n\t · ALL_SONGS_ID_COUNT.pkl [generado en
                    createDict()]")
```

```
547             return ()
548 
549 
550         MIN_YEAR = min(ALL_SONGS_ID_COUNT.year)
551         MAX_YEAR = max(ALL_SONGS_ID_COUNT.year)
552         YEARS = (list(range(MIN_YEAR, MAX_YEAR + 1)))
553 
554         USER_LIST = ALL_SONGS_ID_COUNT.groupby("user")
555 
556         print('\t'.join(str(x) for x in ["user"] + YEARS))
557 
558         for u, r in USER_LIST:
559             COUNTS = []
560             for y in YEARS:
561                 COUNTS.append(len(r.loc[(r.year == y)]))
562 
563             COUNTS = [u] + COUNTS
564             print('\t'.join(str(x) for x in COUNTS))
565 
566     def getUserDays(self):
567 
568         if os.path.isfile("datos/ALL_SONGS_ID_COUNT.pkl"):
569             ALL_SONGS_ID_COUNT = pd.read_pickle("datos/ALL_SONGS_ID_COUNT.pkl")
570 
571         else:
572             print("No existen el pickle:\n\t · ALL_SONGS_ID_COUNT.pkl [generado en
                createDict()]")
573             return ()
574 
575         ALL_SONGS_ID_COUNT["date"] = ALL_SONGS_ID_COUNT.timestamp.dt.date
576 
577         USER_LIST = ALL_SONGS_ID_COUNT.groupby("user")
578         print("user\tdays")
579         for u, r in USER_LIST:
580             days = r.groupby("date")
581             print(str(u) + "\t" + str(len(days)))
582 
583     ##########################
584     # Metodos auxiliares
585     ##########################
586 
587     def printW(self,x):
588         print(bcolors.WARNING + str(x) + bcolors.ENDC)
589 
590     def printB(self,x):
591         print(bcolors.BOLD + str(x) + bcolors.ENDC)
592 
593     def printG(self,x):
594         print(bcolors.OKGREEN + str(x) + bcolors.ENDC)
595 
596     def save_pickle(self,obj, name):
597         with open(name + '.pkl', 'wb') as f:
598             pickle.dump(obj, f, 2)
599 
600     def load_pickle(self,name):
601         with open(name + '.pkl', 'rb') as f:
602             return pickle.load(f)
```

3. Métodos

En los siguientes ficheros se crea, entrena y evalúan los métodos de aprendizaje de embeddings empleados durante la experimentación (W2V, D2V-DM y D2V-DBOW).

3.1. Word2Vec

```
# —*— coding: utf—8 —*—
import os.path
import tensorflow as tf
import numpy as np
import math
import time
import pickle

#########################
# eMtodos
#########################

def getSkipGramData( window=2):

    def load_pickle(name):
        with open(name + '.pkl', 'rb') as f:
            return pickle.load(f)

    filename = "datos/Extend/ARRAY_ARRAYS_TRAIN"

    if os.path.isfile(filename + ".pkl"):
        DATA = load_pickle(filename)
    else:
        print("No existe el pickle...")
        exit()

    DATA = list(map(int, DATA))
    LENGTH = max(DATA) + 1

    RES_LEN = (len(DATA) — (window * 2)) * (window * 2)

    batch_inputs = np.ndarray(shape=(RES_LEN), dtype=np.int32)
    batch_context = np.ndarray(shape=(RES_LEN, 1), dtype=np.int32)

    j = 0

    for i in range(window, len(DATA) — window):
        for w in range(1, window + 1):
            batch_inputs[j] = DATA[i]
            batch_context[j, 0] = DATA[i — w]
            j += 1

            batch_inputs[j] = DATA[i]
            batch_context[j, 0] = DATA[i + w]
            j += 1

    return (batch_inputs, batch_context, LENGTH)

########################
```

```
50 # Llamadas
51 ########################
52
53 batch_size = 512
54 embedding_size = 64  # ñTamao del embedding (capa oculta)
55 window = 2 # ñTamao de la ventana del skipgram
56 num_sampled = 64 # Ejemplos negativos (Para el NSE)
57 seed = 100
58 learning_rate = 0.1
59 epochs = 1000
60
61 # Carpeta donde se almacena el modelo de la red
62 #model_path = 'models/w2v'
63 model_path = 'models/w2v/TITANXP'
64 model_name = 'model'
65 complete_path = model_path+"/"+model_name
66 emb_path = 'embeddings/w2v'
67
68 # Obtener todos lo datos
69 print("Obteniendo datos...")
70 ALL_X, ALL_Y, vocabulary_size = getSkipGramData(window=window)
71 print("Datos obtenidos!")
72
73 #Mezclar datos
74 TOTAL_LENGTH = len(ALL_X)
75 indx = np.arange(TOTAL_LENGTH)
76 np.random.seed(seed)
77 np.random.shuffle(indx)
78
79 ALL_X=ALL_X[indx]
80 ALL_Y=ALL_Y[indx]
81
82 #Separar datos en TRAIN, DEV y TEST
83 TRAIN_LENGTH = int(TOTAL_LENGTH*0.98)
84 DEV_LENGTH= int(TOTAL_LENGTH*0.01)
85 TEST_LENGTH = int(TOTAL_LENGTH*0.01)
86 #Train
87 X= ALL_X[:TRAIN_LENGTH]
88 Y= ALL_Y[:TRAIN_LENGTH]
89 #Dev
90 DEV_X  = ALL_X[TRAIN_LENGTH:TRAIN_LENGTH+DEV_LENGTH]
91 DEV_Y  = ALL_Y[TRAIN_LENGTH:TRAIN_LENGTH+DEV_LENGTH]
92 #Test
93 TEST_X = ALL_X[TRAIN_LENGTH+DEV_LENGTH:]
94 TEST_Y = ALL_Y[TRAIN_LENGTH+DEV_LENGTH:]
95
96 #øCreacin del grafo de TF.
97 graph = tf.Graph()
98
99 with graph.as_default():
100
101     #¿Nmero global de iteraciones
102     global_step = tf.Variable(0, dtype=tf.int32, trainable=False, name='global_step')
103
104     #Datos de entrada
105     #Array del ñtamao del batch con las X
```

```
106     train_inputs = tf.placeholder(tf.int32, shape=[None], name="train_inputs")
107     #Array del ntamao del batch con las Y asociadas
108     train_labels = tf.placeholder(tf.int32, shape=[None, 1], name="train_labels")
109
110     #Conexin primera y segunda capa
        ---------------------------------------------------------------------------------------
111     # Matriz de pesos entre la capa de entrada y la de embeddings
112     embeddings = tf.Variable(tf.random_uniform([vocabulary_size, embedding_size], -1.0, 1.0),
            name='embeddings')
113
114     #embeddings = tf.get_variable("embeddings", shape=[vocabulary_size,
            embedding_size],initializer=tf.contrib.layers.xavier_initializer(seed=seed))
115
116     # Como la relacion entre ambas es lineal, no hay bias ni funcin de activacin
117     # Por tanto se puede obtener el embedding directamente de la matriz anterior ( Embedding
            de la palabra 10 == fila 10 de la matriz de pesos)
118     embed = tf.nn.embedding_lookup(embeddings, train_inputs, name="embed")
119
120     #Conexin segunda y ltima capa
        ---------------------------------------------------------------------------------------
121     #Pesos y biases
122     nce_weights = tf.Variable(tf.truncated_normal([vocabulary_size,
            embedding_size],stddev=1.0 / math.sqrt(embedding_size)), name="weights")
123     nce_biases = tf.Variable(tf.zeros([vocabulary_size]), name="biases")
124
125
126     #Clculo de LOSS y optimizador
        ---------------------------------------------------------------------------------------
127     nce_loss = tf.reduce_mean(
128         tf.nn.nce_loss(weights=nce_weights,
129                        biases=nce_biases,
130                        labels=train_labels,
131                        inputs=embed,
132                        num_sampled=num_sampled,
133                        num_classes=vocabulary_size),name="nce_loss")
134
135     #optimizer =
            tf.train.AdamOptimizer(learning_rate).minimize(nce_loss,global_step=global_step)
136     optimizer =
            tf.train.GradientDescentOptimizer(learning_rate).minimize(nce_loss,global_step=global_step)
137
138     init = tf.global_variables_initializer()
139
140     #Crear objeto encargado de almacenar la red
141     saver = tf.train.Saver(max_to_keep=1)
142
143 gpu_options = tf.GPUOptions(per_process_gpu_memory_fraction=0.333)
144
145 with tf.Session(graph=graph) as session:
146
147     # We must initialize all variables before we use them.
148     init.run()
149     print('Initialized')
150
151     #Obtener el checkpoint
152     ckpt = tf.train.get_checkpoint_state(model_path)
153
```

```
154     #Si existe el checkpoint restaurar
155     if ckpt and ckpt.model_checkpoint_path:
156         saver.restore(session, ckpt.model_checkpoint_path)
157         print('Modelo cargado...')
158     else:
159         print('No existe modelo...')
160 
161     ########################
162     #Obtener la matriz de embeddings
163     print("Obteniendo embeddings...")
164 
165     E = session.run(embeddings)
166     E.dump(emb_path+"/EMB_MATRIX")
167 
168     print("Embedings obtenidos!")
169     ########################
170 
171     exit()
172 
173     #Obtener el ▯nmero de iteraciones necesario y los datos restantes
174     ITRS= int(len(X)/batch_size)
175     RES_ITMS = len(X) - (ITRS*batch_size)
176 
177     #Si no es ▯divisin exacta, se ▯aade una ▯iteracin
178     if(RES_ITMS>0):ITRS+=1
179 
180     print("Iteraciones necesarias:"+str(ITRS))
181 
182     for e in range(epochs):
183         print("Epoch "+str(e))
184 
185         tss = time.time()
186 
187         for step in range(ITRS):
188 
189             #Si se ▯▯aadi una ▯iteracin, introducir los datos restantes
190             if(step==ITRS-1 and RES_ITMS>0):
191                 batch_inputs = X[(ITRS-1)*batch_size:]
192                 batch_context = Y[(ITRS-1)*batch_size:]
193             #Si no, es un batch normal
194             else:
195                 st = step*batch_size
196                 nd = (step+1)*batch_size
197 
198                 batch_inputs = X[st:nd]
199                 batch_context = Y[st:nd]
200 
201             feed_dict = {train_inputs: batch_inputs, train_labels: batch_context}
202 
203             # We perform one update step by evaluating the optimizer op (including it
204             # in the list of returned values for session.run()
205             _, loss_val, gs = session.run([optimizer, nce_loss,global_step],
                    feed_dict=feed_dict)
206 
207             if (gs % 100 == 0 and gs>0):
208 
209                 # The average loss is an estimate of the loss over the last 100 batches.
```

```
210             print('\tLast batch loss at global_step ', gs, ': ', loss_val)
211
212             # Now, save the graph
213             #saver.save(session, complete_path, global_step=global_step)
214
215         print(time.time()-tss)
216         saver.save(session, complete_path, global_step=global_step)
217
218         # Probar con DEV
219         feed_dict = {train_inputs: DEV_X, train_labels: DEV_Y}
220         loss_val = session.run([nce_loss], feed_dict=feed_dict)
221         print('DEV batch loss at global_step ' + str(gs) + ': ' + str(loss_val))
222         with open('train_results.txt', 'a') as the_file: the_file.write('DEV\t' + str(gs) +
                '\t' + str(loss_val) + '\t' + str(time.time() - tss)+'\n')
223
224         # Probar con TEST
225         feed_dict = {train_inputs: TEST_X, train_labels: TEST_Y}
226         loss_val = session.run([nce_loss], feed_dict=feed_dict)
227         print('TEST batch loss at global_step ' + str(gs) + ': ' + str(loss_val))
228         with open('train_results.txt', 'a') as the_file: the_file.write('TEST\t' + str(gs) +
                '\t' + str(loss_val) + '\t' + str(time.time() - tss)+'\n')
229
230
231         #-------------------------
```

3.2. Doc2Vec-DM

```
# -*- coding: utf-8 -*-

import os.path
import tensorflow as tf
from tensorflow.python.client import device_lib
import numpy as np
import math
import time
import pickle

#########################
# Mtodos
#########################

def getData(window=2,seed = 100,shuffle=True, file=None):

    #Cargar datos del tipo [[canciones user1],[canciones user2],...]
    if(file==None):DATA = np.load("datos/Append/ARRAY_ARRAYS_TRAIN.pkl")
    else:DATA = np.load(file)
    #print("ELIMINAR FILTRO---->");DATA = DATA[:5]

    #Obtener el nmero total de usuarios
    NUM_USERS = len(DATA)

    #Matriz resultado
    RES_COLS = window + 1 + 1 # USER ID, [C1,C2], C3
    RES = np.empty([0, RES_COLS], dtype=np.int32)

    #Para cada usuario, crear ejemplos
    for u in range(NUM_USERS):
        tss = time.time()

        USR_SONGS = DATA[u]
        USR_SONGS_SIZE = len(USR_SONGS)

        if(USR_SONGS_SIZE<window+1):print(str(u)+": "+str(USR_SONGS_SIZE));continue

        USR_RES_ROWS = USR_SONGS_SIZE - window

        #Crear matriz de retorno
        USR_RES = np.empty([USR_RES_ROWS, RES_COLS], dtype=np.int32)

        #Cambiar id de usuario
        USR_RES[:,0] = u

        #Obtener los datos (uno ms para la clase (cancin siguiente))
        for i in range(window+1):
            TO = USR_RES_ROWS+i
            USR_RES[:,1+i]= USR_SONGS[i:TO]

        #Finalmente aadir estos datos al conjunto de todos los usuarios
        RES = np.concatenate([RES,USR_RES])

    #Mezclar
    if(shuffle):
```

```
56          np.random.seed(seed)
57          np.random.shuffle(RES)
58
59      #Dividir en TRAIN, DEV y TEST
60      #98/1/1 TODO: VALE ESTAS ???????????
61
62      TOTAL_LENGTH = len(RES)
63      DEV_LENGTH = int(TOTAL_LENGTH * 0.01)
64      TEST_LENGTH = int(TOTAL_LENGTH * 0.01)
65      TRAIN_LENGTH = TOTAL_LENGTH - DEV_LENGTH - TEST_LENGTH
66
67      # Train
68      TRAIN = RES[:TRAIN_LENGTH,:]
69      X = TRAIN[:, :window + 1]
70      Y = np.matrix(TRAIN[:, -1]).T  # Hay que pasarla a matriz y transponer
71      # Dev
72      DEV = RES[TRAIN_LENGTH:TRAIN_LENGTH+DEV_LENGTH,:]
73      DEV_X = DEV[:, :window + 1]
74      DEV_Y = np.matrix(DEV[:, -1]).T  # Hay que pasarla a matriz y transponer
75      # Test
76      TEST = RES[TRAIN_LENGTH+DEV_LENGTH:,:]
77      TEST_X = TEST[:, :window + 1]
78      TEST_Y = np.matrix(TEST[:, -1]).T  # Hay que pasarla a matriz y transponer
79
80      return(X,Y,DEV_X,DEV_Y,TEST_X,TEST_Y, NUM_USERS)
81
82  ##########################
83  # Llamadas
84  ##########################
85
86  #Cargar matriz de pesos W2V
87  W2V_emb_path = 'embeddings/W2V/EMB_MATRIX'
88  W2V_EMB = np.load(W2V_emb_path)
89
90  batch_size = 1024
91  vocabulary_size = W2V_EMB.shape[0]
92  embedding_size_w = W2V_EMB.shape[1]  # nTamao del embedding (capa oculta)
93  embedding_size_d = embedding_size_w
94  window=2
95  seed = 100
96  num_sampled = 64 # Ejemplos negativos (Para el NSE)
97  learning_rate = 0.1
98  epochs = 1500
99
100 # Obtener todos lo datos
101 print("Obteniendo datos...")
102
103 model_name = 'complete'
104 model_path = 'models/d2v_dm/complete'
105 emb_path = 'embeddings/d2v_dm/complete'
106 X,Y,DEV_X,DEV_Y,TEST_X,TEST_Y, user_size = getData(window=window,seed=seed,
        file="datos/Append/ARRAY_ARRAYS_TRAIN_COMPLETE.pkl")
107
108 #model_name = 'present'
109 #model_path = 'models/d2v_dm/present'
110 #emb_path = 'embeddings/d2v_dm/present'
111 #X,Y,DEV_X,DEV_Y,TEST_X,TEST_Y, user_size = getData(window=window,seed=seed,
```

```
    file="datos/Append/ARRAY_ARRAYS_TRAIN_PRESENT.pkl")

print("Datos obtenidos!")

# Carpeta donde se almacena el modelo de la red
complete_path = model_path+"/"+model_name

#Creacin del grafo de TF.
graph = tf.Graph()

with graph.as_default():

    #Nmero global de iteraciones
    global_step = tf.Variable(0, dtype=tf.int32, trainable=False, name='global_step')

    #Datos de entrada
    #Array del tamao del batch con las X
    train_dataset = tf.placeholder(tf.int32, shape=[None, window + 1], name="train_inputs")

    #Array del tamao del batch con las Y asociadas
    train_labels = tf.placeholder(tf.int32, shape=[None, 1], name="train_labels")

    #Conexin primera y segunda capa
    # Matriz W de embeddigs de las canciones obtenida en el W2V previo
    word_embeddings = tf.Variable(W2V_EMB,trainable=False, name="word_embeddings") #
        FUNCIONA???
        https://stackoverflow.com/questions/37326002/is-it-possible-to-make-a-trainable-variable-not-tr

    # Matriz D de embeddigs para los usuarios (lo que se aprende ahora)
    doc_embeddings = tf.Variable(tf.random_uniform([user_size, embedding_size_d], -1.0, 1.0),
        name="doc_embeddings")

    #Conexin segunda y ltima capa

    combined_embed_vector_length = embedding_size_w + embedding_size_d
    # softmax weights, W and D vectors should be concatenated before applying softmax
    weights = tf.Variable(tf.truncated_normal([vocabulary_size,
        combined_embed_vector_length], stddev=1.0 / math.sqrt(combined_embed_vector_length)),
        name="weights")
    # softmax biases
    biases = tf.Variable(tf.zeros([vocabulary_size]), name="biases")

    #Clculo de LOSS y optimizador

    embed = []
    #Obtener el embedding del documento (Columna 0 del batch)
    embed_d = tf.nn.embedding_lookup(doc_embeddings, train_dataset[:, 0])
    embed.append(embed_d)

    #Obtener la SUMA de los embeddings de las canciones (Columnas restantes del batch)
    #El primero se hace fuera. (No se puede crear un vector de 0 inicial e ir aadiendo dado
        que no se conoce el tamao del batch)
    embed_w = tf.nn.embedding_lookup(word_embeddings, train_dataset[:, 1])
```

```
158
159     for j in range(2,window+1):
160         embed_w += tf.nn.embedding_lookup(word_embeddings, train_dataset[:, j])
161     embed.append(embed_w)
162
163     #Se concatena el embedding del usuario con la SUMA de los de las canciones (No se suma el
            embedding del doc) (hasta ahora era un array de 2 elementos)
164     embed = tf.concat(embed, 1)
165
166
167     nce_loss = tf.nn.nce_loss(weights=weights,
168                               biases=biases,
169                               labels=train_labels,
170                               inputs=embed,
171                               num_sampled=num_sampled,
172                               num_classes=vocabulary_size,
173                               name="nce_loss")
174
175     nce_loss = tf.reduce_mean(nce_loss)
176
177     optimizer =
            tf.train.GradientDescentOptimizer(learning_rate).minimize(nce_loss,global_step=global_step)
178
179     init = tf.global_variables_initializer()
180
181     #Crear objeto encargado de almacenar la red
182     saver = tf.train.Saver(max_to_keep=1)
183
184 gpu_options = tf.GPUOptions(per_process_gpu_memory_fraction=0.333)
185
186 with tf.Session(graph=graph,config=tf.ConfigProto(gpu_options=gpu_options)) as session:
187
188     # We must initialize all variables before we use them.
189     init.run()
190     print('Initialized')
191
192     #Obtener el checkpoint
193     ckpt = tf.train.get_checkpoint_state(model_path)
194
195     #Si existe el checkpoint restaurar
196     if ckpt and ckpt.model_checkpoint_path:
197         saver.restore(session, ckpt.model_checkpoint_path)
198         print('Modelo cargado...')
199     else:
200         print('No existe modelo...')
201
202
203     ##########################
204     #Obtener la matriz de embeddings
205     print("Obteniendo embeddings...")
206
207     E = session.run(doc_embeddings)
208     E.dump(emb_path+"/EMB_MATRIX")
209
210     print("Embedings obtenidos!")
211     ##########################
212
```

```
    exit()



    ITRS= int(len(X)/batch_size)
    RES_ITMS = len(X) - (ITRS*batch_size)

    #Si no es divisin exacta, se aade una iteracin
    if(RES_ITMS>0):ITRS+=1

    print("Iteraciones necesarias:"+str(ITRS))

    for e in range(epochs):
        print("Epoch "+str(e))

        tss = time.time()

        for step in range(ITRS):

            #Si se aadi una iteracin, introducir los datos restantes
            if(step==ITRS-1 and RES_ITMS>0):
                batch_inputs = X[(ITRS-1)*batch_size:]
                batch_context = Y[(ITRS-1)*batch_size:]
            #Si no, es un batch normal
            else:
                st = step*batch_size
                nd = (step+1)*batch_size

                batch_inputs = X[st:nd]
                batch_context = Y[st:nd]

            feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
            _, loss_val, gs = session.run([optimizer, nce_loss,global_step],
                feed_dict=feed_dict)


            if (gs % 1000 == 0 and gs>0):

                # The average loss is an estimate of the loss over the last 100 batches.
                print('\tLast batch loss at global-step ', gs, ': ', loss_val)

                # Now, save the graph
                #saver.save(session, complete_path, global_step=global_step)

        print(time.time()-tss)
        saver.save(session, complete_path, global_step=global_step)

        # Probar con DEV
        feed_dict = {train_dataset: DEV_X, train_labels: DEV_Y}
        loss_val = session.run([nce_loss], feed_dict=feed_dict)
        print('DEV batch loss at global-step ' + str(gs) + ': ' + str(loss_val))
        with open('train_results_D2V_DM_'+model_name+'.txt', 'a') as the_file:
            the_file.write('DEV\t' + str(gs) + '\t' + str(loss_val) + '\t' + str(time.time() -
            tss)+'\n')

        # Probar con TEST
        feed_dict = {train_dataset: TEST_X, train_labels: TEST_Y}
```

```
267         loss_val = session.run([nce_loss], feed_dict=feed_dict)
268         print('TEST batch loss at global-step ' + str(gs) + ': ' + str(loss_val))
269         with open('train_results_D2V_DM_'+model_name+'.txt', 'a') as the_file:
                the_file.write('TEST\t' + str(gs) + '\t' + str(loss_val) + '\t' + str(time.time()
                - tss)+'\n')
270
271
272         #-------------------------
```

3.3. Doc2Vec-DBOW

```
# -*- coding: utf-8 -*-

import os.path
import tensorflow as tf
from tensorflow.python.client import device_lib
import numpy as np
import math
import time
import pickle

#########################
# Mtodos
#########################

def getData(seed = 100,shuffle=True,file=None):

    #Cargar datos del tipo [[canciones user1],[canciones user2],...]
    if(file==None):DATA = np.load("datos/Append/ARRAY_ARRAYS_TRAIN.pkl")
    else:DATA = np.load(file)

    #print("ELIMINAR FILTRO---->");DATA = DATA[:5]

    #Obtener el nmero total de usuarios
    NUM_USERS = len(DATA)

    #Matriz resultado
    RES_COLS = 1 + 1 # USER ID => C1
    RES = np.empty([0, RES_COLS], dtype=np.int32)

    #Para cada usuario, crear ejemplos
    for u in range(NUM_USERS):

        USR_SONGS = DATA[u]
        USR_RES_ROWS = len(USR_SONGS)

        #Crear una matriz "vaca de (On de canciones) filas y 2 columnas (USER -> CANCION)
        USR_RES = np.empty([USR_RES_ROWS, RES_COLS], dtype=np.int32)

        #Rellenar
        USR_RES[:,0] = u
        USR_RES[:,1] = USR_SONGS

        #Finalmente aadir estos datos al conjunto de todos los usuarios
        RES = np.concatenate([RES,USR_RES])

    #Mezclar
    if(shuffle):
        np.random.seed(seed)
        np.random.shuffle(RES)

    #Dividir en TRAIN, DEV y TEST
    #98/1/1 TODO: VALE TAS ???????????

    TOTAL_LENGTH = len(RES)
    DEV_LENGTH = int(TOTAL_LENGTH * 0.01)
```

```
    TEST_LENGTH = int(TOTAL_LENGTH * 0.01)
    TRAIN_LENGTH = TOTAL_LENGTH - DEV_LENGTH - TEST_LENGTH

    # Train
    TRAIN = RES[:TRAIN_LENGTH,:]
    X = TRAIN[:, 0]
    Y = np.matrix(TRAIN[:, 1]).T  # Hay que pasarla a matriz y transponer
    # Dev
    DEV = RES[TRAIN_LENGTH:TRAIN_LENGTH+DEV_LENGTH,:]
    DEV_X = DEV[:, 0]
    DEV_Y = np.matrix(DEV[:, 1]).T  # Hay que pasarla a matriz y transponer
    # Test
    TEST = RES[TRAIN_LENGTH+DEV_LENGTH:,:]
    TEST_X = TEST[:, 0]
    TEST_Y = np.matrix(TEST[:,1]).T  # Hay que pasarla a matriz y transponer

    return(X,Y,DEV_X,DEV_Y,TEST_X,TEST_Y, NUM_USERS)

#########################
# Llamadas
#########################

#Cargar matriz de pesos W2V (NO SE NECESITA, SIMPLEMENTE ES PARA CONOCER EL NTAMAO DEL
    VOCABULARIO)
W2V_emb_path = 'embeddings/W2V/EMB_MATRIX'
W2V_EMB = np.load(W2V_emb_path)

batch_size = 1024
vocabulary_size = W2V_EMB.shape[0]
embedding_size_d = 64
seed = 100
num_sampled = 64 # Ejemplos negativos (Para el NSE)
learning_rate = 0.1
epochs = 1500


# Obtener todos lo datos
print("Obteniendo datos...")

#model_name = 'complete'
#model_path = 'models/d2v_dbow/complete'
#emb_path = 'embeddings/d2v_dbow/complete'
#X,Y,DEV_X,DEV_Y,TEST_X,TEST_Y, user_size = getData(seed=seed,
    file="datos/Append/ARRAY_ARRAYS_TRAIN_COMPLETE.pkl")

model_name = 'present'
model_path = 'models/d2v_dbow/present'
emb_path = 'embeddings/d2v_dbow/present'
X,Y,DEV_X,DEV_Y,TEST_X,TEST_Y, user_size = getData(seed=seed,
    file="datos/Append/ARRAY_ARRAYS_TRAIN_PRESENT.pkl")

print(user_size)

print("Datos obtenidos!")

# Carpeta donde se almacena el modelo de la red
complete_path = model_path+"/"+model_name
```

```
110
111 #@Creacin del grafo de TF.
112 graph = tf.Graph()
113
114 with graph.as_default():
115
116     #uNmero global de iteraciones
117     global_step = tf.Variable(0, dtype=tf.int32, trainable=False, name='global_step')
118
119     #Datos de entrada
        ------------------------------------------------------------------------------------------
120     #Array del ntamao del batch con las X
121     train_dataset = tf.placeholder(tf.int32, shape=[None], name="train_inputs")
122
123     #Array del ntamao del batch con las Y asociadas
124     train_labels = tf.placeholder(tf.int32, shape=[None, 1], name="train_labels")
125
126     #@Conexin primera y segunda capa
        ------------------------------------------------------------------------------------------
127
128     # Matriz D de embeddigs para los usuarios (lo que se aprende ahora)
129     doc_embeddings = tf.Variable(tf.random_uniform([user_size, embedding_size_d], -1.0, 1.0),
            name="doc_embeddings")
130
131     embed_d = tf.nn.embedding_lookup(doc_embeddings, train_dataset, name="embed")
132
133
134     #@Conexin segunda y ultima capa
        ------------------------------------------------------------------------------------------
135
136     weights = tf.Variable(tf.truncated_normal([vocabulary_size,embedding_size_d], stddev=1.0
            / math.sqrt(embedding_size_d)), name="weights")
137     # softmax biases
138     biases = tf.Variable(tf.zeros([vocabulary_size]), name="biases")
139
140     #aClculo de LOSS y optimizador-
        ------------------------------------------------------------------------------------------
141
142     nce_loss = tf.nn.nce_loss(weights=weights,
143                               biases=biases,
144                               labels=train_labels,
145                               inputs=embed_d,
146                               num_sampled=num_sampled,
147                               num_classes=vocabulary_size,
148                               name="nce_loss")
149
150     nce_loss = tf.reduce_mean(nce_loss)
151
152     optimizer =
            tf.train.GradientDescentOptimizer(learning_rate).minimize(nce_loss,global_step=global_step)
153
154     init = tf.global_variables_initializer()
155
156     #Crear objeto encargado de almacenar la red
157     saver = tf.train.Saver(max_to_keep=1)
158
159 gpu_options = tf.GPUOptions(per_process_gpu_memory_fraction=0.333)
```

```
160
161 with tf.Session(graph=graph,config=tf.ConfigProto(gpu_options=gpu_options)) as session:
162
163     # We must initialize all variables before we use them.
164     init.run()
165     print('Initialized')
166
167     #Obtener el checkpoint
168     ckpt = tf.train.get_checkpoint_state(model_path)
169
170     #Si existe el checkpoint restaurar
171     if ckpt and ckpt.model_checkpoint_path:
172         saver.restore(session, ckpt.model_checkpoint_path)
173         print('Modelo cargado...')
174     else:
175         print('No existe modelo...')
176
177
178     ##########################
179     #Obtener la matriz de embeddings
180     print("Obteniendo embeddings...")
181
182     E = session.run(doc_embeddings)
183     print(E.shape)
184     E.dump(emb_path+"/EMB_MATRIX")
185
186     print("Embedings obtenidos!")
187     ##########################
188
189     exit()
190
191
192     ITRS= int(len(X)/batch_size)
193     RES_ITMS = len(X) - (ITRS*batch_size)
194
195     #Si no es divisin exacta, se naade una iteracin
196     if(RES_ITMS>0):ITRS+=1
197
198     print("Iteraciones necesarias:"+str(ITRS))
199
200     for e in range(epochs):
201         print("Epoch "+str(e))
202
203         tss = time.time()
204
205         for step in range(ITRS):
206
207             #Si se naaadi una iteracin, introducir los datos restantes
208             if(step==ITRS-1 and RES_ITMS>0):
209                 batch_inputs = X[(ITRS-1)*batch_size:]
210                 batch_context = Y[(ITRS-1)*batch_size:]
211             #Si no, es un batch normal
212             else:
213                 st = step*batch_size
214                 nd = (step+1)*batch_size
215
216                 batch_inputs = X[st:nd]
```

```
            batch_context = Y[st:nd]

        feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
        _, loss_val, gs = session.run([optimizer, nce_loss,global_step],
            feed_dict=feed_dict)


        if (gs % 100 == 0 and gs>0):

            # The average loss is an estimate of the loss over the last 100 batches.
            print('\tLast batch loss at global-step ', gs, ': ', loss_val)

            # Now, save the graph
            #saver.save(session, complete_path, global_step=global_step)

    saver.save(session, complete_path, global_step=global_step)


    # Probar con DEV
    feed_dict = {train_dataset: DEV_X, train_labels: DEV_Y}
    loss_val = session.run([nce_loss], feed_dict=feed_dict)
    print('DEV batch loss at global-step ' + str(gs) + ': ' + str(loss_val))
    with open('train_results_D2V_DBOW_'+model_name+'.txt', 'a') as the_file:
        the_file.write('DEV\t' + str(gs) + '\t' + str(loss_val) + '\t' + str(time.time() -
        tss)+'\n')

    # Probar con TEST
    feed_dict = {train_dataset: TEST_X, train_labels: TEST_Y}
    loss_val = session.run([nce_loss], feed_dict=feed_dict)
    print('TEST batch loss at global-step ' + str(gs) + ': ' + str(loss_val))
    with open('train_results_D2V_DBOW_'+model_name+'.txt', 'a') as the_file:
        the_file.write('TEST\t' + str(gs) + '\t' + str(loss_val) + '\t' + str(time.time()
        - tss)+'\n')


    #-----------------------------
```

4. Tests

En esta sección final, se incluirán todos los ficheros referentes a los test llevados a cabo durante las tareas de recomendación.

4.1. TEST 01

```
# -*- coding: utf-8 -*-

import os.path
import os
import tensorflow as tf
import numpy as np
import random
import time
import math
import pandas as pd
import pickle
from scipy.stats import linregress
from collections import Counter

##########################################################################################################################
# eMtodos
##########################################################################################################################

def getData(REAL_SONG_IDS=[],data
    ="datos/Append/ARRAY_ARRAYS_REC.pkl",pastData="datos/Append/ARRAY_ARRAYS_TRAIN_COMPLETE.pkl",shuffl
    seed=100, train=.7):

    DATA = np.load(data)
    PAST_DATA = np.load(pastData)
    REAL_SET = set(REAL_SONG_IDS)

    random.seed(seed)

    #DATA = DATA[:20];print("ELIMINAR FILTRO")

    #User, cancion, +1 o 0
    TRAIN = np.empty([0, 3], dtype=np.int32)
    DEV = np.empty([0, 3], dtype=np.int32)
    TEST = np.empty([0, 3], dtype=np.int32)

    for u in range(len(DATA)):

        # Eliminar canciones escuchadas que no tienen embedding
        USER_PLAYS = DATA[u]
        USER_PLAYS = pd.DataFrame(list(map(int, USER_PLAYS)))
        USER_PLAYS = USER_PLAYS.loc[USER_PLAYS[0].isin(REAL_SONG_IDS)][0].values

        #Separar en train dev test
        TRAIN_ITEMS = int(len(USER_PLAYS)*train)
        DEV_ITEMS = int((len(USER_PLAYS)-TRAIN_ITEMS)/2)

        TRAIN_DATA = USER_PLAYS[:TRAIN_ITEMS]
        DEV_DATA = USER_PLAYS[TRAIN_ITEMS:TRAIN_ITEMS+DEV_ITEMS]
        TEST_DATA = USER_PLAYS[TRAIN_ITEMS+DEV_ITEMS:]
```

```
48
49          # Obtener IDS de canciones que no han sido escuchadas en TRAIN_COMPLETE y en TRAIN De
                ahora
50          USED_IDS = set(list(map(int, PAST_DATA[u])))
51          USED_IDS.update(TRAIN_DATA)
52          TRAIN_NOT_USED = list(REAL_SET.difference(USED_IDS))
53
54          # Obtener IDS de canciones que no han sido escuchadas en TRAIN_COMPLETE , en TRAIN y
                DEV
55          USED_IDS.update(DEV_DATA)
56          DEV_NOT_USED = list(REAL_SET.difference(USED_IDS))
57
58          # Obtener IDS de canciones que no han sido escuchadas en TRAIN_COMPLETE , en TRAIN y
                DEV y TEST
59          USED_IDS.update(TEST_DATA)
60          TEST_NOT_USED = list(REAL_SET.difference(USED_IDS))
61
62          # Crear casos de train
63          TRAIN_RES = np.empty([len(TRAIN_DATA) * 2, 3], dtype=np.int32)
64          TRAIN_RES[:, 0] = u
65          TRAIN_RES[:len(TRAIN_DATA), 1] = TRAIN_DATA
66          TRAIN_RES[:len(TRAIN_DATA), 2] = 1
67
68          TRAIN_RES[len(TRAIN_DATA):, 1] = random.sample(TRAIN_NOT_USED, len(TRAIN_DATA))
69          TRAIN_RES[len(TRAIN_DATA):, 2] = 0
70
71          TRAIN = np.concatenate([TRAIN, TRAIN_RES])
72
73          # Crear casos de dev
74          DEV_RES = np.empty([len(DEV_DATA) * 2, 3], dtype=np.int32)
75          DEV_RES[:, 0] = u
76          DEV_RES[:len(DEV_DATA), 1] = DEV_DATA
77          DEV_RES[:len(DEV_DATA), 2] = 1
78
79          DEV_RES[len(DEV_DATA):, 1] = random.sample(DEV_NOT_USED, len(DEV_DATA))
80          DEV_RES[len(DEV_DATA):, 2] = 0
81
82          DEV = np.concatenate([DEV, DEV_RES])
83
84          # Crear casos de test
85          TEST_RES = np.empty([len(TEST_DATA) * 2, 3], dtype=np.int32)
86          TEST_RES[:, 0] = u
87          TEST_RES[:len(TEST_DATA), 1] = TEST_DATA
88          TEST_RES[:len(TEST_DATA), 2] = 1
89
90          TEST_RES[len(TEST_DATA):, 1] = random.sample(TEST_NOT_USED, len(TEST_DATA))
91          TEST_RES[len(TEST_DATA):, 2] = 0
92
93          TEST = np.concatenate([TEST, TEST_RES])
94
95      #Mezclar
96      if(shuffle):
97          np.random.seed(seed)
98          np.random.shuffle(TRAIN)
99          np.random.shuffle(DEV)
100         np.random.shuffle(TEST)
101
```

```
102
103     return
            TRAIN[:,:2],np.matrix(TRAIN[:,-1]).T,DEV[:,:2],np.matrix(DEV[:,-1]).T,TEST[:,:2],np.matrix(TEST
104
105 def getBatchData(X, Y, batch_size, step, ITRS, RES_ITMS):
106
107     # Si se ha aadi una iteracin, introducir los datos restantes
108     if (step == ITRS - 1 and RES_ITMS > 0):
109         batch_inputs = X[(ITRS - 1) * batch_size:]
110         batch_context = Y[(ITRS - 1) * batch_size:]
111     # Si no, es un batch normal
112     else:
113         st = step * batch_size
114         nd = (step + 1) * batch_size
115
116         batch_inputs = X[st:nd]
117         batch_context = Y[st:nd]
118     return(batch_inputs,batch_context)
119
120 def trainNet(X,Y, DX,DY,TX,TY,config, user_size,train=True, save=True):
121
122     def getSlope(data):
123         r = linregress(range(len(data)), data)
124         return r.slope
125
126     batch_size = config['batch_size']
127     learning_rate = config['learning_rate']
128     epochs = config['epochs']
129
130     user_size = user_size
131     song_size = W2V.shape[0]
132
133     user_emb_size = 128
134     song_emb_size = 64
135     hidden_size = 32
136
137     num_sampled = 64  # Ejemplos negativos (Para el NSE)
138
139     model_name = 'model'
140     model_path = 'models/' + TEST_NAME + '/model'
141     complete_path = model_path + "/" + model_name
142
143     # Se almacena la loss de dev de las 5 ltimas epochs
144     slope_size = 100
145     train_hist=[]
146     dev_hist=[]
147
148     # Creacin del grafo de TF.
149     graph = tf.Graph()
150
151     with graph.as_default():
152
153         if(config["seed"]!=1):
154             tf.set_random_seed(config["seed"])
155
156         # Nmero global de iteraciones
157         global_step = tf.Variable(0, dtype=tf.int32, trainable=False, name='global_step')
```

```
158
159         # Datos de entrada
            ----------------------------------------------------------------------------------------
160         # Array del tamao del batch con las X
161         train_dataset = tf.placeholder(tf.int32, shape=[None, 2], name="train_inputs")
162
163         # Array del tamao del batch con las Y asociadas
164         train_labels = tf.placeholder(tf.float32, shape=[None, 1], name="train_labels")
165
166         # Embeddings
            ----------------------------------------------------------------------------------------
167
168         # Matriz W de embeddigs de las canciones obtenida en el W2V previo
169
170         O1 = tf.Variable(tf.truncated_normal([user_size, user_emb_size], mean=0.0, stddev=1.0
                / math.sqrt(user_emb_size)), name="O1")
171         BO1 = tf.Variable(tf.zeros([user_emb_size]), name="BO1")
172
173         W2V_EMB = tf.Variable(W2V, trainable=False, name="word_embeddings")  # FUNCIONA???
                https://stackoverflow.com/questions/37326002/is-it-possible-to-make-a-trainable-variable-no
174
175         T1 = tf.Variable(tf.truncated_normal([user_emb_size, hidden_size], mean=0.0,
                stddev=1.0 / math.sqrt(hidden_size)), name="T1")
176         B1 = tf.Variable(tf.zeros([hidden_size]), name="B1")
177
178         T2 = tf.Variable(tf.truncated_normal([song_emb_size, hidden_size], mean=0.0,
                stddev=1.0 / math.sqrt(hidden_size)), name="T2")
179         B2 = tf.Variable(tf.zeros([hidden_size]), name="B2")
180
181         # Operaciones
            ----------------------------------------------------------------------------------------
182
183         # Embedding de documento
184         ed = tf.nn.embedding_lookup(O1, train_dataset[:, 0])
185         ed = ed + BO1
186
187         # Embedding de cancion
188         ec = tf.nn.embedding_lookup(W2V_EMB, train_dataset[:, 1])
189
190         # Transformar a 32 documento
191         hd = tf.matmul(ed,T1)+B1
192
193         # Transformar a 32 cancion
194         hw = tf.matmul(ec,T2)+B2
195
196         # Obtener el producto escalar
197         dot_prod = tf.reduce_sum(tf.multiply(hd, hw), 1, keep_dims=True)
198
199         # aClculo de LOSS y optimizador-
            ----------------------------------------------------------------------------------------
200
201         # Obtener la loss
202         softplus_batch = tf.nn.softplus((1 - 2 * train_labels) * dot_prod)
203         loss_softplus = tf.reduce_mean(softplus_batch)
204
205         # Minimizar la loss
```

```
206         train_step =
                tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(loss=loss_softplus,global_step
207
208         # Obtener la probabilidad
209         prob = tf.sigmoid(dot_prod)
210
211         # Inicializar variables
212         init = tf.global_variables_initializer()
213
214         # Crear objeto encargado de almacenar la red
215         saver = tf.train.Saver(max_to_keep=1)
216
217     config = tf.ConfigProto()
218     config.gpu_options.allow_growth = True
219     with tf.Session(graph=graph, config=config) as session:
220
221         # We must initialize all variables before we use them.
222         init.run()
223
224         if(save):
225             # Obtener el checkpoint
226             ckpt = tf.train.get_checkpoint_state(model_path)
227
228             # Si existe el checkpoint restaurar
229             if ckpt and ckpt.model_checkpoint_path:
230                 saver.restore(session, ckpt.model_checkpoint_path)
231
232         #Si se desea entrenar
233         if(train):
234             #-----------------------------------------------------------------------------------------
235             # TRAIN
236             #-----------------------------------------------------------------------------------------
237
238             ITRS= int(len(X)/batch_size)
239             RES_ITMS = len(X) - (ITRS*batch_size)
240             if(RES_ITMS>0):ITRS+=1 #Si no es divisin exacta, se aade una iteracin
241
242             for e in range(epochs):
243                 TRAIN_LOSS = 0.0
244
245                 print("Epoch "+str(e))
246
247                 for step in range(ITRS):
248
249                     batch_inputs, batch_context =
                            getBatchData(X,Y,batch_size,step,ITRS,RES_ITMS)
250
251                     feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
252
253                     _, loss_val,softplus, gs = session.run([train_step,
                            loss_softplus,softplus_batch,global_step], feed_dict=feed_dict)
254
255                     TRAIN_LOSS += np.sum(softplus[:, 0])
256
257                 if (save): saver.save(session, complete_path, global_step=global_step)
258
259                 #-----------------------------------------------------------------------------------
```

```
260             # DEV
261             #-----------------------------------------------------------------------------
262
263             if(len(DX)>0):
264                 ITRS_TEST = int(len(DX) / batch_size)
265                 RES_ITMS_TEST = len(DX) - (ITRS_TEST * batch_size)
266                 TEST_LOSS = 0.0
267                 if (RES_ITMS_TEST > 0): ITRS_TEST += 1 # Si no es divisin exacta, se
                        naade una iteracin
268
269                 for step in range(ITRS_TEST):
270
271                     batch_inputs, batch_context =
                            getBatchData(DX,DY,batch_size,step,ITRS_TEST,RES_ITMS_TEST)
272
273                     feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
274                     loss_val, softplus = session.run([loss_softplus,softplus_batch],
                            feed_dict=feed_dict)
275
276                     TEST_LOSS+=np.sum(softplus[:,0])
277
278                 dev_hist.append(TEST_LOSS/(len(DX)*1.0))
279                 train_hist.append(TRAIN_LOSS/(len(X)*1.0))
280
281             if (len(dev_hist) >= slope_size):
282                 # Calcular la pendiente
283                 print(getSlope(dev_hist[-slope_size:]))
284                 print(dev_hist[-slope_size:])
285
286                 if (getSlope(dev_hist[-slope_size:]) > -1e-5 or e==epochs-1):
287
288                     def printArray(data,data2):
289                         for i in range(len(data)):
290                             a=data[i]
291                             b=data2[i]
292                             print(str(a).replace(".",",")+"\t"+str(b).replace(".",","))
293
294                     print("-" * 50)
295                     print(e)
296                     printArray(dev_hist,train_hist)
297                     return
298
299         else:
300             #-----------------------------------------------------------------------------
301             # TEST FINAL
302             #-----------------------------------------------------------------------------
303
304             TRUE_POSITIVE = 0
305             FALSE_NEGATIVE = 0
306             FALSE_POSITIVE = 0
307             TRUE_NEGATIVE = 0
308
309             ITRS_TEST = int(len(TX) / batch_size)
310             RES_ITMS_TEST = len(TX) - (ITRS_TEST * batch_size)
311             TEST_LOSS = 0.0
```

```
312             # Si no es divisin exacta, se aade una iteracin
313             if (RES_ITMS_TEST > 0): ITRS_TEST += 1
314 
315             for step in range(ITRS_TEST):
316                 # Si se ha aadi una iteracin, introducir los datos restantes
317                 if (step == ITRS_TEST - 1 and RES_ITMS_TEST > 0):
318                     batch_inputs = TX[(ITRS_TEST - 1) * batch_size:]
319                     batch_context = TY[(ITRS_TEST - 1) * batch_size:]
320                 # Si no, es un batch normal
321                 else:
322                     st = step * batch_size
323                     nd = (step + 1) * batch_size
324 
325                     batch_inputs = TX[st:nd]
326                     batch_context = TY[st:nd]
327 
328                 feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
329                 loss_val,loss_batch,  p = session.run([loss_softplus,softplus_batch, prob],
                        feed_dict=feed_dict)
330 
331                 TEST_LOSS += np.sum(loss_batch)
332 
333                 # aClculo de TP/TN/FN/FP para cada batch
334 
335                 for i in range(len(p)):
336                     real = batch_context[i, 0]
337                     pred = p[i, 0]
338 
339                     if (real == 1) and (pred > 0.5): TRUE_POSITIVE += 1
340                     if (real == 1) and (pred <= 0.5): FALSE_NEGATIVE += 1
341                     if (real == 0) and (pred > 0.5): FALSE_POSITIVE += 1
342                     if (real == 0) and (pred <= 0.5): TRUE_NEGATIVE += 1
343 
344             PRECISION = TRUE_POSITIVE / (TRUE_POSITIVE + FALSE_POSITIVE * 1.0)
345             RECALL = TRUE_POSITIVE / (TRUE_POSITIVE + FALSE_NEGATIVE * 1.0)
346 
347             F1 = 2.0 * ((PRECISION * RECALL) / (PRECISION + RECALL))
348 
349             print("-" * 50)
350             print("LOSS:\t" + str(TEST_LOSS / (len(TX) * 1.0)))
351             print("PRECISION:\t" + str(PRECISION))
352             print("RECALL:\t" + str(RECALL))
353             print("F1:\t" + str(F1))
354             print("-" * 50)
355             print("")
356         #-------------------------------------------------------------------------------------------------------------
357 
358 ##########################################################################################################################
359 # Llamadas
360 ##########################################################################################################################
361 
362 TEST_NAME="test_021"
363 
364 os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
365 
366 #Para conocer los embeddings de las canciones
367 W2V  = np.load('embeddings/w2v/EMB_MATRIX')
```

```
368
369 #Se cargan los datos de train de las canciones para conocer los IDs reales
370 DATA = np.load("datos/Extend/ARRAY_ARRAYS_TRAIN.pkl")
371
372 #Obtener una lista con los ID de canciones que no aestn el la matriz de embeddings
373 ALL = set(list(range(W2V.shape[0])))
374 REAL = set(list(set(list(map(int, DATA)))))
375
376 ##########################################################################################################
377 # TENSORFLOW
378 ##########################################################################################################
379
380 #---------------------------------------------------------------------------------------------------------
381 # Grid Search + Early Stopping
382 #---------------------------------------------------------------------------------------------------------
383 '''
384 X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))
385 rates = [0.000001,1, 0.01, 0.0001]
386
387 for r in rates:
388     config = {"batch_size": 1024, "learning_rate": r, "epochs": 1000, "seed": 100}
389
390     print("Learning Rate: "+str(r))
391     print("-" * 50)
392     trainNet(X, Y, DX, DY, TX, TY, config, user_size, train=True, save=False)
393     print("-"*50)
394
395 exit()
396 '''
397 #---------------------------------------------------------------------------------------------------------
398 # Train
399 #---------------------------------------------------------------------------------------------------------
400 '''
401 X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))
402
403 #Para esta fase, se une TRAIN y DEV
404 X = np.concatenate((X,DX),axis=0)
405 Y = np.concatenate((Y,DY),axis=0)
406
407 DX=[]
408 DY=[]
409
410 config = {"batch_size": 1024, "learning_rate": 0.0001, "epochs": 102, "seed": 100}
411 trainNet(X, Y, DX, DY, TX, TY, config, user_size,train=True,save=True)
412
413 exit()
414 '''
415 #---------------------------------------------------------------------------------------------------------
416 # Test
417 #---------------------------------------------------------------------------------------------------------
418
419
420 X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))
421
422 config = {"batch_size": 1024, "learning_rate": 0.0001, "epochs": 102, "seed": 100}
423 trainNet(X, Y, DX, DY, TX, TY, config, user_size,train=False, save=True)
```

4.2. TEST 02

```
# -*- coding: utf-8 -*-

import os.path
import os
import tensorflow as tf
import numpy as np
import random
import time
import math
import pandas as pd
import pickle
from scipy.stats import linregress
from collections import Counter

#########################################################################################################
# eMtodos
#########################################################################################################

def getData(REAL_SONG_IDS=[],data
    ="datos/Append/ARRAY_ARRAYS_REC.pkl",pastData="datos/Append/ARRAY_ARRAYS_TRAIN_COMPLETE.pkl",shuffl
    seed=100, train=.7):

    DATA = np.load(data)
    PAST_DATA = np.load(pastData)
    REAL_SET = set(REAL_SONG_IDS)

    random.seed(seed)

    #DATA = DATA[:20];print("ELIMINAR FILTRO")

    #User, cancion, +1 o 0
    TRAIN = np.empty([0, 3], dtype=np.int32)
    DEV = np.empty([0, 3], dtype=np.int32)
    TEST = np.empty([0, 3], dtype=np.int32)

    for u in range(len(DATA)):

        # Eliminar canciones escuchadas que no tienen embedding
        USER_PLAYS = DATA[u]
        USER_PLAYS = pd.DataFrame(list(map(int, USER_PLAYS)))
        USER_PLAYS = USER_PLAYS.loc[USER_PLAYS[0].isin(REAL_SONG_IDS)][0].values

        #Separar en train dev test
        TRAIN_ITEMS = int(len(USER_PLAYS)*train)
        DEV_ITEMS = int((len(USER_PLAYS)-TRAIN_ITEMS)/2)

        TRAIN_DATA = USER_PLAYS[:TRAIN_ITEMS]
        DEV_DATA = USER_PLAYS[TRAIN_ITEMS:TRAIN_ITEMS+DEV_ITEMS]
        TEST_DATA = USER_PLAYS[TRAIN_ITEMS+DEV_ITEMS:]

        # Obtener IDS de canciones que no han sido escuchadas en TRAIN_COMPLETE y en TRAIN De
            ahora
        USED_IDS = set(list(map(int, PAST_DATA[u])))
        USED_IDS.update(TRAIN_DATA)
        TRAIN_NOT_USED = list(REAL_SET.difference(USED_IDS))
```

```
53
54          # Obtener IDS de canciones que no han sido escuchadas en TRAIN_COMPLETE , en TRAIN y
                DEV
55          USED_IDS.update(DEV_DATA)
56          DEV_NOT_USED = list(REAL_SET.difference(USED_IDS))
57
58          # Obtener IDS de canciones que no han sido escuchadas en TRAIN_COMPLETE , en TRAIN y
                DEV y TEST
59          USED_IDS.update(TEST_DATA)
60          TEST_NOT_USED = list(REAL_SET.difference(USED_IDS))
61
62          # Crear casos de train
63          TRAIN_RES = np.empty([len(TRAIN_DATA) * 2, 3], dtype=np.int32)
64          TRAIN_RES[:, 0] = u
65          TRAIN_RES[:len(TRAIN_DATA), 1] = TRAIN_DATA
66          TRAIN_RES[:len(TRAIN_DATA), 2] = 1
67
68          TRAIN_RES[len(TRAIN_DATA):, 1] = random.sample(TRAIN_NOT_USED, len(TRAIN_DATA))
69          TRAIN_RES[len(TRAIN_DATA):, 2] = 0
70
71          TRAIN = np.concatenate([TRAIN, TRAIN_RES])
72
73          # Crear casos de dev
74          DEV_RES = np.empty([len(DEV_DATA) * 2, 3], dtype=np.int32)
75          DEV_RES[:, 0] = u
76          DEV_RES[:len(DEV_DATA), 1] = DEV_DATA
77          DEV_RES[:len(DEV_DATA), 2] = 1
78
79          DEV_RES[len(DEV_DATA):, 1] = random.sample(DEV_NOT_USED, len(DEV_DATA))
80          DEV_RES[len(DEV_DATA):, 2] = 0
81
82          DEV = np.concatenate([DEV, DEV_RES])
83
84          # Crear casos de test
85          TEST_RES = np.empty([len(TEST_DATA) * 2, 3], dtype=np.int32)
86          TEST_RES[:, 0] = u
87          TEST_RES[:len(TEST_DATA), 1] = TEST_DATA
88          TEST_RES[:len(TEST_DATA), 2] = 1
89
90          TEST_RES[len(TEST_DATA):, 1] = random.sample(TEST_NOT_USED, len(TEST_DATA))
91          TEST_RES[len(TEST_DATA):, 2] = 0
92
93          TEST = np.concatenate([TEST, TEST_RES])
94
95      #Mezclar
96      if(shuffle):
97          np.random.seed(seed)
98          np.random.shuffle(TRAIN)
99          np.random.shuffle(DEV)
100         np.random.shuffle(TEST)
101
102
103     return
            TRAIN[:,:2],np.matrix(TRAIN[:,-1]).T,DEV[:,:2],np.matrix(DEV[:,-1]).T,TEST[:,:2],np.matrix(TEST
104
105 def getBatchData(X, Y, batch_size, step, ITRS, RES_ITMS):
106
```

```
107     # Si se ha añadi una iteracin, introducir los datos restantes
108     if (step == ITRS - 1 and RES_ITMS > 0):
109         batch_inputs = X[(ITRS - 1) * batch_size:]
110         batch_context = Y[(ITRS - 1) * batch_size:]
111     # Si no, es un batch normal
112     else:
113         st = step * batch_size
114         nd = (step + 1) * batch_size
115 
116         batch_inputs = X[st:nd]
117         batch_context = Y[st:nd]
118     return(batch_inputs,batch_context)
119 
120 def trainNet(X,Y, DX,DY,TX,TY,config, user_size,train=True, save=True):
121 
122     def getSlope(data):
123         r = linregress(range(len(data)), data)
124         return r.slope
125 
126     batch_size = config['batch_size']
127     learning_rate = config['learning_rate']
128     epochs = config['epochs']
129 
130     user_size = user_size
131     song_size = W2V.shape[0]
132 
133     user_emb_size = 128
134     song_emb_size = 64
135     hidden_size = 32
136 
137     num_sampled = 64  # Ejemplos negativos (Para el NSE)
138 
139     model_name = 'model'
140     model_path = 'models/' + TEST_NAME + '/model'
141     complete_path = model_path + "/" + model_name
142 
143     # Se almacena la loss de dev de las 5 ultimas epochs
144     slope_size = 100
145     train_hist=[]
146     dev_hist=[]
147 
148     # Creacin del grafo de TF.
149     graph = tf.Graph()
150 
151     with graph.as_default():
152 
153         if(config["seed"]!=1):
154             tf.set_random_seed(config["seed"])
155 
156         # Nmero global de iteraciones
157         global_step = tf.Variable(0, dtype=tf.int32, trainable=False, name='global_step')
158 
159         # Datos de entrada
160         # Array del tamao del batch con las X
161         train_dataset = tf.placeholder(tf.int32, shape=[None, 2], name="train_inputs")
162 
```

```
163         # Array del tamao del batch con las Y asociadas
164         train_labels = tf.placeholder(tf.float32, shape=[None, 1], name="train_labels")
165 
166         # Embeddings
                ------------------------------------------------------------------------------
167 
168         D2V_EMB_VRB = tf.Variable(D2V_EMB, trainable=False, name="doc_embeddings")
169 
170         W2V_EMB_VRB = tf.Variable(W2V, trainable=False, name="word_embeddings")
171 
172         T1 = tf.Variable(tf.truncated_normal([user_emb_size, hidden_size], mean=0.0,
                stddev=1.0 / math.sqrt(hidden_size)), name="T1")
173         B1 = tf.Variable(tf.zeros([hidden_size]), name="B1")
174 
175         T2 = tf.Variable(tf.truncated_normal([song_emb_size, hidden_size], mean=0.0,
                stddev=1.0 / math.sqrt(hidden_size)), name="T2")
176         B2 = tf.Variable(tf.zeros([hidden_size]), name="B2")
177 
178         # Operaciones
                ------------------------------------------------------------------------------
179 
180         # Embedding de documento
181         ed = tf.nn.embedding_lookup(D2V_EMB_VRB, train_dataset[:, 0])
182 
183         # Embedding de cancion
184         ec = tf.nn.embedding_lookup(W2V_EMB_VRB, train_dataset[:, 1])
185 
186         # Transformar a 32 documento
187         hd = tf.matmul(ed,T1)+B1
188 
189         # Transformar a 32 cancion
190         hw = tf.matmul(ec,T2)+B2
191 
192         # Obtener el producto escalar
193         dot_prod = tf.reduce_sum(tf.multiply(hd, hw), 1, keep_dims=True)
194 
195         # aClculo de LOSS y optimizador
                ------------------------------------------------------------------------------
196 
197         # Obtener la loss
198         softplus_batch = tf.nn.softplus((1 - 2 * train_labels) * dot_prod)
199         loss_softplus = tf.reduce_mean(softplus_batch)
200 
201         # Minimizar la loss
202         train_step =
                tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(loss=loss_softplus,global_step
203 
204         # Obtener la probabilidad
205         prob = tf.sigmoid(dot_prod)
206 
207         # Inicializar variables
208         init = tf.global_variables_initializer()
209 
210         # Crear objeto encargado de almacenar la red
211         saver = tf.train.Saver(max_to_keep=1)
212 
213     config = tf.ConfigProto()
```

```
214     config.gpu_options.allow_growth = True
215     with tf.Session(graph=graph, config=config) as session:
216 
217         # We must initialize all variables before we use them.
218         init.run()
219 
220         if(save):
221             # Obtener el checkpoint
222             ckpt = tf.train.get_checkpoint_state(model_path)
223 
224             # Si existe el checkpoint restaurar
225             if ckpt and ckpt.model_checkpoint_path:
226                 saver.restore(session, ckpt.model_checkpoint_path)
227 
228         #Si se desea entrenar
229         if(train):
230             #-----------------------------------------------------------------------------------------------------------------
231             # TRAIN
232             #-----------------------------------------------------------------------------------------------------------------
233 
234             ITRS= int(len(X)/batch_size)
235             RES_ITMS = len(X) - (ITRS*batch_size)
236             if(RES_ITMS>0):ITRS+=1 #Si no es @divisin exacta, se aade una @iteracin
237 
238             for e in range(epochs):
239                 TRAIN_LOSS = 0.0
240 
241                 print("Epoch "+str(e))
242 
243                 for step in range(ITRS):
244 
245                     batch_inputs, batch_context =
                            getBatchData(X,Y,batch_size,step,ITRS,RES_ITMS)
246 
247                     feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
248 
249                     _, loss_val,softplus, gs = session.run([train_step,
                            loss_softplus,softplus_batch,global_step], feed_dict=feed_dict)
250 
251                     TRAIN_LOSS += np.sum(softplus[:, 0])
252 
253                 if (save): saver.save(session, complete_path, global_step=global_step)
254 
255                 #-------------------------------------------------------------------------------------------------------------
256                 # DEV
257                 #-------------------------------------------------------------------------------------------------------------
258 
259                 if(len(DX)>0):
260                     ITRS_TEST = int(len(DX) / batch_size)
261                     RES_ITMS_TEST = len(DX) - (ITRS_TEST * batch_size)
262                     TEST_LOSS = 0.0
263                     if (RES_ITMS_TEST > 0): ITRS_TEST += 1 # Si no es @divisin exacta, se
                            aade una @iteracin
264 
265                     for step in range(ITRS_TEST):
266 
267                         batch_inputs, batch_context =
```

```
                    getBatchData(DX,DY,batch_size,step,ITRS_TEST,RES_ITMS_TEST)

                feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
                loss_val, softplus = session.run([loss_softplus,softplus_batch],
                    feed_dict=feed_dict)

                TEST_LOSS+=np.sum(softplus[:,0])

            dev_hist.append(TEST_LOSS/(len(DX)*1.0))
            train_hist.append(TRAIN_LOSS/(len(X)*1.0))

        if (len(dev_hist) >= slope_size):
            # Calcular la pendiente
            print(getSlope(dev_hist[-slope_size:]))
            print(dev_hist[-slope_size:])

            if (getSlope(dev_hist[-slope_size:]) > -1e-5 or e==epochs-1):

                def printArray(data,data2):
                    for i in range(len(data)):
                        a=data[i]
                        b=data2[i]
                        print(str(a).replace(".",",")+"\t"+str(b).replace(".",","))

                print("-" * 50)
                print(e)
                printArray(dev_hist,train_hist)
                return

else:
    #-------------------------------------------------------------------------------
    # TEST FINAL
    #-------------------------------------------------------------------------------

    TRUE_POSITIVE = 0
    FALSE_NEGATIVE = 0
    FALSE_POSITIVE = 0
    TRUE_NEGATIVE = 0

    ITRS_TEST = int(len(TX) / batch_size)
    RES_ITMS_TEST = len(TX) - (ITRS_TEST * batch_size)
    TEST_LOSS = 0.0
    # Si no es divisin exacta, se aade una iteracin
    if (RES_ITMS_TEST > 0): ITRS_TEST += 1

    for step in range(ITRS_TEST):
        # Si se aadi una iteracin, introducir los datos restantes
        if (step == ITRS_TEST - 1 and RES_ITMS_TEST > 0):
            batch_inputs = TX[(ITRS_TEST - 1) * batch_size:]
            batch_context = TY[(ITRS_TEST - 1) * batch_size:]
        # Si no, es un batch normal
        else:
            st = step * batch_size
            nd = (step + 1) * batch_size

```

```
                    batch_inputs = TX[st:nd]
                    batch_context = TY[st:nd]

                feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
                loss_val,loss_batch,  p = session.run([loss_softplus,softplus_batch, prob],
                    feed_dict=feed_dict)

                TEST_LOSS += np.sum(loss_batch)

                # aClculo de TP/TN/FN/FP para cada batch

                for i in range(len(p)):
                    real = batch_context[i, 0]
                    pred = p[i, 0]

                    if (real == 1) and (pred > 0.5): TRUE_POSITIVE += 1
                    if (real == 1) and (pred <= 0.5): FALSE_NEGATIVE += 1
                    if (real == 0) and (pred > 0.5): FALSE_POSITIVE += 1
                    if (real == 0) and (pred <= 0.5): TRUE_NEGATIVE += 1

            PRECISION = TRUE_POSITIVE / (TRUE_POSITIVE + FALSE_POSITIVE * 1.0)
            RECALL = TRUE_POSITIVE / (TRUE_POSITIVE + FALSE_NEGATIVE * 1.0)

            F1 = 2.0 * ((PRECISION * RECALL) / (PRECISION + RECALL))

            print("-" * 50)
            print("LOSS:\t" + str(TEST_LOSS / (len(TX) * 1.0)))
            print("PRECISION:\t" + str(PRECISION))
            print("RECALL:\t" + str(RECALL))
            print("F1:\t" + str(F1))
            print("-" * 50)
            print("")
        #------------------------------------------------------------------------------------------

##########################################################################################
# Llamadas
##########################################################################################

TEST_NAME="test_031"

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'

#Se obtinenen cada uno de los embeddings aprendidos (incluidas las canciones)
D2V_DM_COMPLETE   = np.load('embeddings/d2v_dm/complete/EMB_MATRIX')
D2V_DBOW_COMPLETE = np.load('embeddings/d2v_dbow/complete/EMB_MATRIX')
W2V  = np.load('embeddings/w2v/EMB_MATRIX')

#Se concatenan los embeddings de los usuarios para generar el perfil
D2V_EMB = np.concatenate((D2V_DM_COMPLETE,D2V_DBOW_COMPLETE),axis=1) # El consolidado =
    Consolidado DM + Consolidado DBOW


#Se cargan los datos de train de las canciones para conocer los IDs reales
DATA = np.load("datos/Extend/ARRAY_ARRAYS_TRAIN.pkl")

#Obtener una lista con los ID de canciones que no aestn el la matriz de embeddings
ALL = set(list(range(W2V.shape[0])))
```

```
REAL = set(list(set(list(map(int, DATA)))))
NOT_EXISTS = list(ALL.difference(REAL))

#########################################################################################################
# TENSORFLOW
#########################################################################################################

#-------------------------------------------------------------------------------------------------------
# Grid Search + Early Stopping
#-------------------------------------------------------------------------------------------------------
'''
X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))
rates = [0.000001,1, 0.01, 0.0001]

for r in rates:
    config = {"batch_size": 1024, "learning_rate": r, "epochs": 1000, "seed": 100}

    print("Learning Rate: "+str(r))
    print("-" * 50)
    trainNet(X, Y, DX, DY, TX, TY, config, user_size, train=True, save=False)
    print("-"*50)

exit()
'''
#-------------------------------------------------------------------------------------------------------
# Train
#-------------------------------------------------------------------------------------------------------
'''
X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))

#Para esta fase, se une TRAIN y DEV
X = np.concatenate((X,DX),axis=0)
Y = np.concatenate((Y,DY),axis=0)

DX=[]
DY=[]

config = {"batch_size": 1024, "learning_rate": 0.0001, "epochs": 199, "seed": 100}
trainNet(X, Y, DX, DY, TX, TY, config, user_size,train=True,save=True)

exit()
'''
#-------------------------------------------------------------------------------------------------------
# Test
#-------------------------------------------------------------------------------------------------------


X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))

config = {"batch_size": 1024, "learning_rate": 0.0001, "epochs": 199, "seed": 100}
trainNet(X, Y, DX, DY, TX, TY, config, user_size,train=False, save=True)
```

4.3. TEST 03

```
# -*- coding: utf-8 -*-

import os.path
import os
import tensorflow as tf
import numpy as np
import random
import time
import math
import pandas as pd
import pickle
from scipy.stats import linregress
from collections import Counter

################################################################################################################
# eMtodos
################################################################################################################

def getData(REAL_SONG_IDS=[],data
    ="datos/Append/ARRAY_ARRAYS_REC.pkl",pastData="datos/Append/ARRAY_ARRAYS_TRAIN_COMPLETE.pkl",shuffl
    seed=100, train=.7):

    DATA = np.load(data)
    PAST_DATA = np.load(pastData)
    REAL_SET = set(REAL_SONG_IDS)

    random.seed(seed)

    #DATA = DATA[:20];print("ELIMINAR FILTRO")

    #User, cancion, +1 o 0
    TRAIN = np.empty([0, 3], dtype=np.int32)
    DEV = np.empty([0, 3], dtype=np.int32)
    TEST = np.empty([0, 3], dtype=np.int32)

    for u in range(len(DATA)):

        # Eliminar canciones escuchadas que no tienen embedding
        USER_PLAYS = DATA[u]
        USER_PLAYS = pd.DataFrame(list(map(int, USER_PLAYS)))
        USER_PLAYS = USER_PLAYS.loc[USER_PLAYS[0].isin(REAL_SONG_IDS)][0].values

        #Separar en train dev test
        TRAIN_ITEMS = int(len(USER_PLAYS)*train)
        DEV_ITEMS = int((len(USER_PLAYS)-TRAIN_ITEMS)/2)

        TRAIN_DATA = USER_PLAYS[:TRAIN_ITEMS]
        DEV_DATA = USER_PLAYS[TRAIN_ITEMS:TRAIN_ITEMS+DEV_ITEMS]
        TEST_DATA = USER_PLAYS[TRAIN_ITEMS+DEV_ITEMS:]

        # Obtener IDS de canciones que no han sido escuchadas en TRAIN_COMPLETE y en TRAIN De
            ahora
        USED_IDS = set(list(map(int, PAST_DATA[u])))
        USED_IDS.update(TRAIN_DATA)
        TRAIN_NOT_USED = list(REAL_SET.difference(USED_IDS))
```

```
53
54          # Obtener IDS de canciones que no han sido escuchadas en TRAIN_COMPLETE , en TRAIN y
                DEV
55          USED_IDS.update(DEV_DATA)
56          DEV_NOT_USED = list(REAL_SET.difference(USED_IDS))
57
58          # Obtener IDS de canciones que no han sido escuchadas en TRAIN_COMPLETE , en TRAIN y
                DEV y TEST
59          USED_IDS.update(TEST_DATA)
60          TEST_NOT_USED = list(REAL_SET.difference(USED_IDS))
61
62          # Crear casos de train
63          TRAIN_RES = np.empty([len(TRAIN_DATA) * 2, 3], dtype=np.int32)
64          TRAIN_RES[:, 0] = u
65          TRAIN_RES[:len(TRAIN_DATA), 1] = TRAIN_DATA
66          TRAIN_RES[:len(TRAIN_DATA), 2] = 1
67
68          TRAIN_RES[len(TRAIN_DATA):, 1] = random.sample(TRAIN_NOT_USED, len(TRAIN_DATA))
69          TRAIN_RES[len(TRAIN_DATA):, 2] = 0
70
71          TRAIN = np.concatenate([TRAIN, TRAIN_RES])
72
73          # Crear casos de dev
74          DEV_RES = np.empty([len(DEV_DATA) * 2, 3], dtype=np.int32)
75          DEV_RES[:, 0] = u
76          DEV_RES[:len(DEV_DATA), 1] = DEV_DATA
77          DEV_RES[:len(DEV_DATA), 2] = 1
78
79          DEV_RES[len(DEV_DATA):, 1] = random.sample(DEV_NOT_USED, len(DEV_DATA))
80          DEV_RES[len(DEV_DATA):, 2] = 0
81
82          DEV = np.concatenate([DEV, DEV_RES])
83
84          # Crear casos de test
85          TEST_RES = np.empty([len(TEST_DATA) * 2, 3], dtype=np.int32)
86          TEST_RES[:, 0] = u
87          TEST_RES[:len(TEST_DATA), 1] = TEST_DATA
88          TEST_RES[:len(TEST_DATA), 2] = 1
89
90          TEST_RES[len(TEST_DATA):, 1] = random.sample(TEST_NOT_USED, len(TEST_DATA))
91          TEST_RES[len(TEST_DATA):, 2] = 0
92
93          TEST = np.concatenate([TEST, TEST_RES])
94
95      #Mezclar
96      if(shuffle):
97          np.random.seed(seed)
98          np.random.shuffle(TRAIN)
99          np.random.shuffle(DEV)
100         np.random.shuffle(TEST)
101
102
103     return
            TRAIN[:,:2],np.matrix(TRAIN[:,-1]).T,DEV[:,:2],np.matrix(DEV[:,-1]).T,TEST[:,:2],np.matrix(TEST
104
105 def getBatchData(X, Y, batch_size, step, ITRS, RES_ITMS):
106
```

```
    # Si se añaadi una iteracin, introducir los datos restantes
    if (step == ITRS - 1 and RES_ITMS > 0):
        batch_inputs = X[(ITRS - 1) * batch_size:]
        batch_context = Y[(ITRS - 1) * batch_size:]
    # Si no, es un batch normal
    else:
        st = step * batch_size
        nd = (step + 1) * batch_size

        batch_inputs = X[st:nd]
        batch_context = Y[st:nd]
    return(batch_inputs,batch_context)

def trainNet(X,Y, DX,DY,TX,TY,config, user_size,train=True, save=True):

    def getSlope(data):
        r = linregress(range(len(data)), data)
        return r.slope

    batch_size = config['batch_size']
    learning_rate = config['learning_rate']
    epochs = config['epochs']

    user_size = user_size
    song_size = W2V.shape[0]

    user_emb_size = 128
    song_emb_size = 64
    hidden_size = 32

    num_sampled = 64  # Ejemplos negativos (Para el NSE)

    model_name = 'model'
    model_path = 'models/' + TEST_NAME + '/model'
    complete_path = model_path + "/" + model_name

    # Se almacena la loss de dev de las 5 ultimas epochs
    slope_size = 100
    train_hist=[]
    dev_hist=[]

    # Creacin del grafo de TF.
    graph = tf.Graph()

    with graph.as_default():

        if(config["seed"]!=1):
            tf.set_random_seed(config["seed"])

        # Nmero global de iteraciones
        global_step = tf.Variable(0, dtype=tf.int32, trainable=False, name='global_step')

        # Datos de entrada
        # Array del tamao del batch con las X
        train_dataset = tf.placeholder(tf.int32, shape=[None, 2], name="train_inputs")

```

```
163         # Array del tamao del batch con las Y asociadas
164         train_labels = tf.placeholder(tf.float32, shape=[None, 1], name="train_labels")
165 
166         # Embeddings
                ----------------------------------------------------------------------------------------------------
167 
168         D2V_EMB_VRB = tf.Variable(D2V_EMB, trainable=False, name="doc_embeddings")
169 
170         W2V_EMB_VRB = tf.Variable(W2V, trainable=False, name="word_embeddings")
171 
172         T1 = tf.Variable(tf.truncated_normal([user_emb_size, hidden_size], mean=0.0,
                stddev=1.0 / math.sqrt(hidden_size)), name="T1")
173         B1 = tf.Variable(tf.zeros([hidden_size]), name="B1")
174 
175         T2 = tf.Variable(tf.truncated_normal([song_emb_size, hidden_size], mean=0.0,
                stddev=1.0 / math.sqrt(hidden_size)), name="T2")
176         B2 = tf.Variable(tf.zeros([hidden_size]), name="B2")
177 
178         # Operaciones
                ----------------------------------------------------------------------------------------------------
179 
180         # Embedding de documento
181         ed = tf.nn.embedding_lookup(D2V_EMB_VRB, train_dataset[:, 0])
182 
183         # Embedding de cancion
184         ec = tf.nn.embedding_lookup(W2V_EMB_VRB, train_dataset[:, 1])
185 
186         # Transformar a 32 documento
187         hd = tf.matmul(ed,T1)+B1
188 
189         # Transformar a 32 cancion
190         hw = tf.matmul(ec,T2)+B2
191 
192         # Obtener el producto escalar
193         dot_prod = tf.reduce_sum(tf.multiply(hd, hw), 1, keep_dims=True)
194 
195         # aClculo de LOSS y optimizador
                ----------------------------------------------------------------------------------------------------
196 
197         # Obtener la loss
198         softplus_batch = tf.nn.softplus((1 - 2 * train_labels) * dot_prod)
199         loss_softplus = tf.reduce_mean(softplus_batch)
200 
201         # Minimizar la loss
202         train_step =
                tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(loss=loss_softplus,global_step
203 
204         # Obtener la probabilidad
205         prob = tf.sigmoid(dot_prod)
206 
207         # Inicializar variables
208         init = tf.global_variables_initializer()
209 
210         # Crear objeto encargado de almacenar la red
211         saver = tf.train.Saver(max_to_keep=1)
212 
213     config = tf.ConfigProto()
```

```
214     config.gpu_options.allow_growth = True
215     with tf.Session(graph=graph, config=config) as session:
216 
217         # We must initialize all variables before we use them.
218         init.run()
219 
220         if(save):
221             # Obtener el checkpoint
222             ckpt = tf.train.get_checkpoint_state(model_path)
223 
224             # Si existe el checkpoint restaurar
225             if ckpt and ckpt.model_checkpoint_path:
226                 saver.restore(session, ckpt.model_checkpoint_path)
227 
228         #Si se desea entrenar
229         if(train):
230             #-----------------------------------------------------------------------------------------------
231             # TRAIN
232             #-----------------------------------------------------------------------------------------------
233 
234             ITRS= int(len(X)/batch_size)
235             RES_ITMS = len(X) - (ITRS*batch_size)
236             if(RES_ITMS>0):ITRS+=1 #Si no es ◌divisin exacta, se ◌aade una ◌iteracin
237 
238             for e in range(epochs):
239                 TRAIN_LOSS = 0.0
240 
241                 print("Epoch "+str(e))
242 
243                 for step in range(ITRS):
244 
245                     batch_inputs, batch_context =
                            getBatchData(X,Y,batch_size,step,ITRS,RES_ITMS)
246 
247                     feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
248 
249                     _, loss_val,softplus, gs = session.run([train_step,
                            loss_softplus,softplus_batch,global_step], feed_dict=feed_dict)
250 
251                     TRAIN_LOSS += np.sum(softplus[:, 0])
252 
253                 if (save): saver.save(session, complete_path, global_step=global_step)
254 
255                 #-----------------------------------------------------------------------------------------------
256                 # DEV
257                 #-----------------------------------------------------------------------------------------------
258 
259                 if(len(DX)>0):
260                     ITRS_TEST = int(len(DX) / batch_size)
261                     RES_ITMS_TEST = len(DX) - (ITRS_TEST * batch_size)
262                     TEST_LOSS = 0.0
263                     if (RES_ITMS_TEST > 0): ITRS_TEST += 1 # Si no es ◌divisin exacta, se
                            ◌aade una ◌iteracin
264 
265                     for step in range(ITRS_TEST):
266 
267                         batch_inputs, batch_context =
```

```
                    getBatchData(DX,DY,batch_size,step,ITRS_TEST,RES_ITMS_TEST)

                feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
                loss_val, softplus = session.run([loss_softplus,softplus_batch],
                    feed_dict=feed_dict)

                TEST_LOSS+=np.sum(softplus[:,0])

            dev_hist.append(TEST_LOSS/(len(DX)*1.0))
            train_hist.append(TRAIN_LOSS/(len(X)*1.0))

        if (len(dev_hist) >= slope_size):
            # Calcular la pendiente
            print(getSlope(dev_hist[-slope_size:]))
            print(dev_hist[-slope_size:])

            if (getSlope(dev_hist[-slope_size:]) > -1e-5 or e==epochs-1):

                def printArray(data,data2):
                    for i in range(len(data)):
                        a=data[i]
                        b=data2[i]
                        print(str(a).replace(".",",")+"\t"+str(b).replace(".",","))

                print("-" * 50)
                print(e)
                printArray(dev_hist,train_hist)
                return

else:
    #-------------------------------------------------------------------------------
    # TEST FINAL
    #-------------------------------------------------------------------------------

    TRUE_POSITIVE = 0
    FALSE_NEGATIVE = 0
    FALSE_POSITIVE = 0
    TRUE_NEGATIVE = 0

    ITRS_TEST = int(len(TX) / batch_size)
    RES_ITMS_TEST = len(TX) - (ITRS_TEST * batch_size)
    TEST_LOSS = 0.0
    # Si no es divisin exacta, se naade una iteracin
    if (RES_ITMS_TEST > 0): ITRS_TEST += 1

    for step in range(ITRS_TEST):
        # Si se naadi una iteracin, introducir los datos restantes
        if (step == ITRS_TEST - 1 and RES_ITMS_TEST > 0):
            batch_inputs = TX[(ITRS_TEST - 1) * batch_size:]
            batch_context = TY[(ITRS_TEST - 1) * batch_size:]
        # Si no, es un batch normal
        else:
            st = step * batch_size
            nd = (step + 1) * batch_size
```

```
                    batch_inputs = TX[st:nd]
                    batch_context = TY[st:nd]

                feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
                loss_val,loss_batch,  p = session.run([loss_softplus,softplus_batch, prob],
                    feed_dict=feed_dict)

                TEST_LOSS += np.sum(loss_batch)

                # aClculo de TP/TN/FN/FP para cada batch

                for i in range(len(p)):
                    real = batch_context[i, 0]
                    pred = p[i, 0]

                    if (real == 1) and (pred > 0.5): TRUE_POSITIVE += 1
                    if (real == 1) and (pred <= 0.5): FALSE_NEGATIVE += 1
                    if (real == 0) and (pred > 0.5): FALSE_POSITIVE += 1
                    if (real == 0) and (pred <= 0.5): TRUE_NEGATIVE += 1

            PRECISION = TRUE_POSITIVE / (TRUE_POSITIVE + FALSE_POSITIVE * 1.0)
            RECALL = TRUE_POSITIVE / (TRUE_POSITIVE + FALSE_NEGATIVE * 1.0)

            F1 = 2.0 * ((PRECISION * RECALL) / (PRECISION + RECALL))

            print("-" * 50)
            print("LOSS:\t" + str(TEST_LOSS / (len(TX) * 1.0)))
            print("PRECISION:\t" + str(PRECISION))
            print("RECALL:\t" + str(RECALL))
            print("F1:\t" + str(F1))
            print("-" * 50)
            print("")
        #------------------------------------------------------------------------------------------

###############################################################################################
# Llamadas
###############################################################################################

TEST_NAME="test_041"

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'

#Se obtinenen cada uno de los embeddings aprendidos (incluidas las canciones)
D2V_DM_PRESENT   = np.load('embeddings/d2v_dm/present/EMB_MATRIX')
D2V_DBOW_PRESENT  = np.load('embeddings/d2v_dbow/present/EMB_MATRIX')
W2V  = np.load('embeddings/w2v/EMB_MATRIX')

#Se concatenan los embeddings de los usuarios para generar el perfil
D2V_EMB = np.concatenate((D2V_DM_PRESENT,D2V_DBOW_PRESENT),axis=1) # El reciente = Reciente
    DM + Reciente DBOW


#Se cargan los datos de train de las canciones para conocer los IDs reales
DATA = np.load("datos/Extend/ARRAY_ARRAYS_TRAIN.pkl")

#Obtener una lista con los ID de canciones que no aestn el la matriz de embeddings
ALL = set(list(range(W2V.shape[0])))
```

```
REAL = set(list(set(list(map(int, DATA)))))
NOT_EXISTS = list(ALL.difference(REAL))

#########################################################################################################
# TENSORFLOW
#########################################################################################################

#-------------------------------------------------------------------------------------------------------
# Grid Search + Early Stopping
#-------------------------------------------------------------------------------------------------------
'''
X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))
rates = [0.000001,1, 0.01, 0.0001]

for r in rates:
    config = {"batch_size": 1024, "learning_rate": r, "epochs": 1000, "seed": 100}

    print("Learning Rate: "+str(r))
    print("-" * 50)
    trainNet(X, Y, DX, DY, TX, TY, config, user_size, train=True, save=False)
    print("-"*50)

exit()
'''
#-------------------------------------------------------------------------------------------------------
# Train
#-------------------------------------------------------------------------------------------------------
'''
X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))

#Para esta fase, se une TRAIN y DEV
X = np.concatenate((X,DX),axis=0)
Y = np.concatenate((Y,DY),axis=0)

DX=[]
DY=[]

config = {"batch_size": 1024, "learning_rate": 0.0001, "epochs": 202, "seed": 100}
trainNet(X, Y, DX, DY, TX, TY, config, user_size,train=True,save=True)

exit()
'''
#-------------------------------------------------------------------------------------------------------
# Test
#-------------------------------------------------------------------------------------------------------


X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))

config = {"batch_size": 1024, "learning_rate": 0.0001, "epochs": 202, "seed": 100}
trainNet(X, Y, DX, DY, TX, TY, config, user_size,train=False, save=True)
```

4.4. TEST 04

```
# -*- coding: utf-8 -*-

import os.path
import os
import tensorflow as tf
import numpy as np
import random
import time
import math
import pandas as pd
import pickle
from scipy.stats import linregress
from collections import Counter

#########################################################################################################
# eMtodos
#########################################################################################################

def getData(REAL_SONG_IDS=[],data
    ="datos/Append/ARRAY_ARRAYS_REC.pkl",pastData="datos/Append/ARRAY_ARRAYS_TRAIN_COMPLETE.pkl",shuffl
    seed=100, train=.7):

    DATA = np.load(data)
    PAST_DATA = np.load(pastData)
    REAL_SET = set(REAL_SONG_IDS)

    random.seed(seed)

    #DATA = DATA[:20];print("ELIMINAR FILTRO")

    #User, cancion, +1 o 0
    TRAIN = np.empty([0, 3], dtype=np.int32)
    DEV = np.empty([0, 3], dtype=np.int32)
    TEST = np.empty([0, 3], dtype=np.int32)

    for u in range(len(DATA)):

        # Eliminar canciones escuchadas que no tienen embedding
        USER_PLAYS = DATA[u]
        USER_PLAYS = pd.DataFrame(list(map(int, USER_PLAYS)))
        USER_PLAYS = USER_PLAYS.loc[USER_PLAYS[0].isin(REAL_SONG_IDS)][0].values

        #Separar en train dev test
        TRAIN_ITEMS = int(len(USER_PLAYS)*train)
        DEV_ITEMS = int((len(USER_PLAYS)-TRAIN_ITEMS)/2)

        TRAIN_DATA = USER_PLAYS[:TRAIN_ITEMS]
        DEV_DATA = USER_PLAYS[TRAIN_ITEMS:TRAIN_ITEMS+DEV_ITEMS]
        TEST_DATA = USER_PLAYS[TRAIN_ITEMS+DEV_ITEMS:]

        # Obtener IDS de canciones que no han sido escuchadas en TRAIN_COMPLETE y en TRAIN De
            ahora
        USED_IDS = set(list(map(int, PAST_DATA[u])))
        USED_IDS.update(TRAIN_DATA)
        TRAIN_NOT_USED = list(REAL_SET.difference(USED_IDS))
```

```
53
54          # Obtener IDS de canciones que no han sido escuchadas en TRAIN_COMPLETE , en TRAIN y
                DEV
55          USED_IDS.update(DEV_DATA)
56          DEV_NOT_USED = list(REAL_SET.difference(USED_IDS))
57
58          # Obtener IDS de canciones que no han sido escuchadas en TRAIN_COMPLETE , en TRAIN y
                DEV y TEST
59          USED_IDS.update(TEST_DATA)
60          TEST_NOT_USED = list(REAL_SET.difference(USED_IDS))
61
62          # Crear casos de train
63          TRAIN_RES = np.empty([len(TRAIN_DATA) * 2, 3], dtype=np.int32)
64          TRAIN_RES[:, 0] = u
65          TRAIN_RES[:len(TRAIN_DATA), 1] = TRAIN_DATA
66          TRAIN_RES[:len(TRAIN_DATA), 2] = 1
67
68          TRAIN_RES[len(TRAIN_DATA):, 1] = random.sample(TRAIN_NOT_USED, len(TRAIN_DATA))
69          TRAIN_RES[len(TRAIN_DATA):, 2] = 0
70
71          TRAIN = np.concatenate([TRAIN, TRAIN_RES])
72
73          # Crear casos de dev
74          DEV_RES = np.empty([len(DEV_DATA) * 2, 3], dtype=np.int32)
75          DEV_RES[:, 0] = u
76          DEV_RES[:len(DEV_DATA), 1] = DEV_DATA
77          DEV_RES[:len(DEV_DATA), 2] = 1
78
79          DEV_RES[len(DEV_DATA):, 1] = random.sample(DEV_NOT_USED, len(DEV_DATA))
80          DEV_RES[len(DEV_DATA):, 2] = 0
81
82          DEV = np.concatenate([DEV, DEV_RES])
83
84          # Crear casos de test
85          TEST_RES = np.empty([len(TEST_DATA) * 2, 3], dtype=np.int32)
86          TEST_RES[:, 0] = u
87          TEST_RES[:len(TEST_DATA), 1] = TEST_DATA
88          TEST_RES[:len(TEST_DATA), 2] = 1
89
90          TEST_RES[len(TEST_DATA):, 1] = random.sample(TEST_NOT_USED, len(TEST_DATA))
91          TEST_RES[len(TEST_DATA):, 2] = 0
92
93          TEST = np.concatenate([TEST, TEST_RES])
94
95      #Mezclar
96      if(shuffle):
97          np.random.seed(seed)
98          np.random.shuffle(TRAIN)
99          np.random.shuffle(DEV)
100         np.random.shuffle(TEST)
101
102
103     return
            TRAIN[:,:2],np.matrix(TRAIN[:,-1]).T,DEV[:,:2],np.matrix(DEV[:,-1]).T,TEST[:,:2],np.matrix(TEST
104
105 def getBatchData(X, Y, batch_size, step, ITRS, RES_ITMS):
106
```

```
107     # Si se ha aadi una iteracin, introducir los datos restantes
108     if (step == ITRS - 1 and RES_ITMS > 0):
109         batch_inputs = X[(ITRS - 1) * batch_size:]
110         batch_context = Y[(ITRS - 1) * batch_size:]
111     # Si no, es un batch normal
112     else:
113         st = step * batch_size
114         nd = (step + 1) * batch_size
115 
116         batch_inputs = X[st:nd]
117         batch_context = Y[st:nd]
118     return(batch_inputs,batch_context)
119 
120 def trainNet(X,Y, DX,DY,TX,TY,config, user_size,train=True, save=True):
121 
122     def getSlope(data):
123         r = linregress(range(len(data)), data)
124         return r.slope
125 
126     batch_size = config['batch_size']
127     learning_rate = config['learning_rate']
128     epochs = config['epochs']
129 
130     user_size = user_size
131     song_size = W2V.shape[0]
132 
133     user_emb_size = 128
134     user_concat_size = user_emb_size * 2
135     song_emb_size = 64
136     hidden_size = 32
137 
138     num_sampled = 64  # Ejemplos negativos (Para el NSE)
139 
140     model_name = 'model'
141     model_path = 'models/' + TEST_NAME + '/model'
142     complete_path = model_path + "/" + model_name
143 
144     # Se almacena la loss de dev de las 5 ultimas epochs
145     slope_size = 100
146     train_hist=[]
147     dev_hist=[]
148 
149     # Creacin del grafo de TF.
150     graph = tf.Graph()
151 
152     with graph.as_default():
153 
154         if(config["seed"]!=1):
155             tf.set_random_seed(config["seed"])
156 
157         # Nmero global de iteraciones
158         global_step = tf.Variable(0, dtype=tf.int32, trainable=False, name='global_step')
159 
160         # Datos de entrada
161         # Array del tamao del batch con las X
162         train_dataset = tf.placeholder(tf.int32, shape=[None, 2], name="train_inputs")
```

```
163
164         # Array del tamao del batch con las Y asociadas
165         train_labels = tf.placeholder(tf.float32, shape=[None, 1], name="train_labels")
166
167         # Embeddings
            ------------------------------------------------------------------------------
168
169         D2V_EMB_VRB = tf.Variable(D2V_EMB, trainable=False, name="doc_embeddings")
170
171         T0 = tf.Variable(tf.truncated_normal([user_concat_size, user_emb_size], mean=0.0,
                stddev=1.0 / math.sqrt(hidden_size)), name="T0")
172         B0 = tf.Variable(tf.zeros([user_emb_size]), name="B0")
173
174         W2V_EMB_VRB = tf.Variable(W2V, trainable=False, name="word_embeddings")
175
176         T1 = tf.Variable(tf.truncated_normal([user_emb_size, hidden_size], mean=0.0,
                stddev=1.0 / math.sqrt(hidden_size)), name="T1")
177         B1 = tf.Variable(tf.zeros([hidden_size]), name="B1")
178
179         T2 = tf.Variable(tf.truncated_normal([song_emb_size, hidden_size], mean=0.0,
                stddev=1.0 / math.sqrt(hidden_size)), name="T2")
180         B2 = tf.Variable(tf.zeros([hidden_size]), name="B2")
181
182         # Operaciones
            ------------------------------------------------------------------------------
183
184         # Embedding de documento
185         ed = tf.nn.embedding_lookup(D2V_EMB_VRB, train_dataset[:, 0])
186
187         #Transformar concat documento de 256 a 128
188         ed = tf.matmul(ed,T0)+B0
189
190         # Embedding de cancion
191         ec = tf.nn.embedding_lookup(W2V_EMB_VRB, train_dataset[:, 1])
192
193         # Transformar a 32 documento
194         hd = tf.matmul(ed,T1)+B1
195
196         # Transformar a 32 cancion
197         hw = tf.matmul(ec,T2)+B2
198
199         # Obtener el producto escalar
200         dot_prod = tf.reduce_sum(tf.multiply(hd, hw), 1, keep_dims=True)
201
202         # aClculo de LOSS y optimizador
            ------------------------------------------------------------------------------
203
204         # Obtener la loss
205         softplus_batch = tf.nn.softplus((1 - 2 * train_labels) * dot_prod)
206         loss_softplus = tf.reduce_mean(softplus_batch)
207
208         # Minimizar la loss
209         train_step =
                tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(loss=loss_softplus,global_step
210
211         # Obtener la probabilidad
212         prob = tf.sigmoid(dot_prod)
```

```
213
214         # Inicializar variables
215         init = tf.global_variables_initializer()
216
217         # Crear objeto encargado de almacenar la red
218         saver = tf.train.Saver(max_to_keep=1)
219
220     #config = tf.ConfigProto()
221     #config.gpu_options.allow_growth = True
222     #with tf.Session(graph=graph,config=config) as session:
223     with tf.Session(graph=graph) as session:
224
225         # We must initialize all variables before we use them.
226         init.run()
227
228         if(save):
229             # Obtener el checkpoint
230             ckpt = tf.train.get_checkpoint_state(model_path)
231
232             # Si existe el checkpoint restaurar
233             if ckpt and ckpt.model_checkpoint_path:
234                 saver.restore(session, ckpt.model_checkpoint_path)
235
236         #Si se desea entrenar
237         if(train):
238             #-----------------------------------------------------------------------------------------------------------------------
239             # TRAIN
240             #-----------------------------------------------------------------------------------------------------------------------
241
242             ITRS= int(len(X)/batch_size)
243             RES_ITMS = len(X) - (ITRS*batch_size)
244             if(RES_ITMS>0):ITRS+=1 #Si no es división exacta, se añade una iteración
245
246             for e in range(epochs):
247                 TRAIN_LOSS = 0.0
248
249                 print("Epoch "+str(e))
250
251                 for step in range(ITRS):
252
253                     batch_inputs, batch_context =
                            getBatchData(X,Y,batch_size,step,ITRS,RES_ITMS)
254
255                     feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
256
257                     _, loss_val,softplus, gs = session.run([train_step,
                            loss_softplus,softplus_batch,global_step], feed_dict=feed_dict)
258
259                     TRAIN_LOSS += np.sum(softplus[:, 0])
260
261                 if (save): saver.save(session, complete_path, global_step=global_step)
262
263                 #-----------------------------------------------------------------------------------------------------------------------
264                 # DEV
265                 #-----------------------------------------------------------------------------------------------------------------------
266
267                 if(len(DX)>0):
```

```
268                 ITRS_TEST = int(len(DX) / batch_size)
269                 RES_ITMS_TEST = len(DX) - (ITRS_TEST * batch_size)
270                 TEST_LOSS = 0.0
271                 if (RES_ITMS_TEST > 0): ITRS_TEST += 1 # Si no es ødivisin exacta, se
                        ñaade una øiteracin
272
273                 for step in range(ITRS_TEST):
274
275                     batch_inputs, batch_context =
                            getBatchData(DX,DY,batch_size,step,ITRS_TEST,RES_ITMS_TEST)
276
277                     feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
278                     loss_val, softplus = session.run([loss_softplus,softplus_batch],
                            feed_dict=feed_dict)
279
280                     TEST_LOSS+=np.sum(softplus[:,0])
281
282                 dev_hist.append(TEST_LOSS/(len(DX)*1.0))
283                 train_hist.append(TRAIN_LOSS/(len(X)*1.0))
284
285             if (len(dev_hist) >= slope_size):
286                 # Calcular la pendiente
287                 print(getSlope(dev_hist[-slope_size:]))
288                 print(dev_hist[-slope_size:])
289
290                 if (getSlope(dev_hist[-slope_size:]) > -1e-5 or e==epochs-1):
291
292                     def printArray(data,data2):
293                         for i in range(len(data)):
294                             a=data[i]
295                             b=data2[i]
296                             print(str(a).replace(".",",")+"\t"+str(b).replace(".",","))
297
298                     print("-" * 50)
299                     print(e)
300                     printArray(dev_hist,train_hist)
301                     return
302
303         else:
304             #
                ------------------------------------------------------------------------------
305             # TEST FINAL
306             #
                ------------------------------------------------------------------------------
307
308             TRUE_POSITIVE = 0
309             FALSE_NEGATIVE = 0
310             FALSE_POSITIVE = 0
311             TRUE_NEGATIVE = 0
312
313             ITRS_TEST = int(len(TX) / batch_size)
314             RES_ITMS_TEST = len(TX) - (ITRS_TEST * batch_size)
315             TEST_LOSS = 0.0
316             # Si no es ødivisin exacta, se ñaade una øiteracin
317             if (RES_ITMS_TEST > 0): ITRS_TEST += 1
318
319             for step in range(ITRS_TEST):
```

```
320                 # Si se no aadi una iteracin, introducir los datos restantes
321                 if (step == ITRS_TEST - 1 and RES_ITMS_TEST > 0):
322                     batch_inputs = TX[(ITRS_TEST - 1) * batch_size:]
323                     batch_context = TY[(ITRS_TEST - 1) * batch_size:]
324                 # Si no, es un batch normal
325                 else:
326                     st = step * batch_size
327                     nd = (step + 1) * batch_size
328 
329                     batch_inputs = TX[st:nd]
330                     batch_context = TY[st:nd]
331 
332                 feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
333                 loss_val,loss_batch,  p = session.run([loss_softplus,softplus_batch, prob],
                        feed_dict=feed_dict)
334 
335                 TEST_LOSS += np.sum(loss_batch)
336 
337                 # aClculo de TP/TN/FN/FP para cada batch
338 
339                 for i in range(len(p)):
340                     real = batch_context[i, 0]
341                     pred = p[i, 0]
342 
343                     if (real == 1) and (pred > 0.5): TRUE_POSITIVE += 1
344                     if (real == 1) and (pred <= 0.5): FALSE_NEGATIVE += 1
345                     if (real == 0) and (pred > 0.5): FALSE_POSITIVE += 1
346                     if (real == 0) and (pred <= 0.5): TRUE_NEGATIVE += 1
347 
348             PRECISION = TRUE_POSITIVE / (TRUE_POSITIVE + FALSE_POSITIVE * 1.0)
349             RECALL = TRUE_POSITIVE / (TRUE_POSITIVE + FALSE_NEGATIVE * 1.0)
350 
351             F1 = 2.0 * ((PRECISION * RECALL) / (PRECISION + RECALL))
352 
353             print("-" * 50)
354             print("LOSS:\t" + str(TEST_LOSS / (len(TX) * 1.0)))
355             print("PRECISION:\t" + str(PRECISION))
356             print("RECALL:\t" + str(RECALL))
357             print("F1:\t" + str(F1))
358             print("-" * 50)
359             print("")
360         #----------------------------------------------------------------------------------------------------
361 
362 ##########################################################################################################
363 # Llamadas
364 ##########################################################################################################
365 
366 TEST_NAME="test_051"
367 
368 os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
369 
370 #Se obtinenen cada uno de los embeddings aprendidos (incluidas las canciones)
371 D2V_DM_COMPLETE   = np.load('embeddings/d2v_dm/complete/EMB_MATRIX')
372 D2V_DBOW_COMPLETE = np.load('embeddings/d2v_dbow/complete/EMB_MATRIX')
373 
374 #Se obtinenen cada uno de los embeddings aprendidos (incluidas las canciones)
375 D2V_DM_PRESENT  = np.load('embeddings/d2v_dm/present/EMB_MATRIX')
```

```
D2V_DBOW_PRESENT = np.load('embeddings/d2v_dbow/present/EMB_MATRIX')

W2V  = np.load('embeddings/w2v/EMB_MATRIX')

#Se concatenan los embeddings de los usuarios para generar el perfil
D2V_CONS = np.concatenate((D2V_DM_COMPLETE,D2V_DBOW_COMPLETE),axis=1) # El consolidado =
    Consolidado DM + Consolidado DBOW
D2V_PRES = np.concatenate((D2V_DM_PRESENT,D2V_DBOW_PRESENT),axis=1) # El presente = Presente
    DM + Presente DBOW

D2V_EMB = np.concatenate((D2V_CONS,D2V_PRES),axis=1) # El perfil completo CONS + PRES

#Se cargan los datos de train de las canciones para conocer los IDs reales
DATA = np.load("datos/Extend/ARRAY_ARRAYS_TRAIN.pkl")

#Obtener una lista con los ID de canciones que no aestn el la matriz de embeddings
ALL = set(list(range(W2V.shape[0])))
REAL = set(list(set(list(map(int, DATA)))))
NOT_EXISTS = list(ALL.difference(REAL))

#########################################################################################################
# TENSORFLOW
#########################################################################################################

#-------------------------------------------------------------------------------------------------------
# Grid Search + Early Stopping
#-------------------------------------------------------------------------------------------------------
'''
X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))
rates = [0.000001,1, 0.01, 0.0001]

for r in rates:
    config = {"batch_size": 1024, "learning_rate": r, "epochs": 1000, "seed": 100}

    print("Learning Rate: "+str(r))
    print("-" * 50)
    trainNet(X, Y, DX, DY, TX, TY, config, user_size, train=True, save=False)
    print("-"*50)

exit()
'''
#-------------------------------------------------------------------------------------------------------
# Train
#-------------------------------------------------------------------------------------------------------
'''
X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))

#Para esta fase, se une TRAIN y DEV
X = np.concatenate((X,DX),axis=0)
Y = np.concatenate((Y,DY),axis=0)

DX=[]
DY=[]

config = {"batch_size": 1024, "learning_rate": 0.0001, "epochs": 194, "seed": 100}
trainNet(X, Y, DX, DY, TX, TY, config, user_size,train=True,save=True)

```

```
exit()
'''
#-----------------------------------------------------------------------------------------------------------------------
# Test
#-----------------------------------------------------------------------------------------------------------------------


X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))

config = {"batch_size": 1024, "learning_rate": 0.0001, "epochs": 194, "seed": 100}
trainNet(X, Y, DX, DY, TX, TY, config, user_size,train=False, save=True)
```

4.5. TEST 05

```
# -*- coding: utf-8 -*-

import os.path
import os
import tensorflow as tf
import numpy as np
import random
import time
import math
import pandas as pd
import pickle
from scipy.stats import linregress
from collections import Counter

#########################################################################################################
# eMtodos
#########################################################################################################

def getData(REAL_SONG_IDS=[],data
    ="datos/Append/ARRAY_ARRAYS_REC.pkl",pastData="datos/Append/ARRAY_ARRAYS_TRAIN_COMPLETE.pkl",shuffl
    seed=100, train=.7):

    DATA = np.load(data)
    PAST_DATA = np.load(pastData)
    REAL_SET = set(REAL_SONG_IDS)

    random.seed(seed)

    #DATA = DATA[:20];print("ELIMINAR FILTRO")

    #User, cancion, +1 o 0
    TRAIN = np.empty([0, 3], dtype=np.int32)
    DEV = np.empty([0, 3], dtype=np.int32)
    TEST = np.empty([0, 3], dtype=np.int32)

    for u in range(len(DATA)):

        # Eliminar canciones escuchadas que no tienen embedding
        USER_PLAYS = DATA[u]
        USER_PLAYS = pd.DataFrame(list(map(int, USER_PLAYS)))
        USER_PLAYS = USER_PLAYS.loc[USER_PLAYS[0].isin(REAL_SONG_IDS)][0].values

        #Separar en train dev test
        TRAIN_ITEMS = int(len(USER_PLAYS)*train)
        DEV_ITEMS = int((len(USER_PLAYS)-TRAIN_ITEMS)/2)

        TRAIN_DATA = USER_PLAYS[:TRAIN_ITEMS]
        DEV_DATA = USER_PLAYS[TRAIN_ITEMS:TRAIN_ITEMS+DEV_ITEMS]
        TEST_DATA = USER_PLAYS[TRAIN_ITEMS+DEV_ITEMS:]

        # Obtener IDS de canciones que no han sido escuchadas en TRAIN_COMPLETE y en TRAIN De
            ahora
        USED_IDS = set(list(map(int, PAST_DATA[u])))
        USED_IDS.update(TRAIN_DATA)
        TRAIN_NOT_USED = list(REAL_SET.difference(USED_IDS))
```

```
53
54          # Obtener IDS de canciones que no han sido escuchadas en TRAIN_COMPLETE , en TRAIN y
                DEV
55          USED_IDS.update(DEV_DATA)
56          DEV_NOT_USED = list(REAL_SET.difference(USED_IDS))
57
58          # Obtener IDS de canciones que no han sido escuchadas en TRAIN_COMPLETE , en TRAIN y
                DEV y TEST
59          USED_IDS.update(TEST_DATA)
60          TEST_NOT_USED = list(REAL_SET.difference(USED_IDS))
61
62          # Crear casos de train
63          TRAIN_RES = np.empty([len(TRAIN_DATA) * 2, 3], dtype=np.int32)
64          TRAIN_RES[:, 0] = u
65          TRAIN_RES[:len(TRAIN_DATA), 1] = TRAIN_DATA
66          TRAIN_RES[:len(TRAIN_DATA), 2] = 1
67
68          TRAIN_RES[len(TRAIN_DATA):, 1] = random.sample(TRAIN_NOT_USED, len(TRAIN_DATA))
69          TRAIN_RES[len(TRAIN_DATA):, 2] = 0
70
71          TRAIN = np.concatenate([TRAIN, TRAIN_RES])
72
73          # Crear casos de dev
74          DEV_RES = np.empty([len(DEV_DATA) * 2, 3], dtype=np.int32)
75          DEV_RES[:, 0] = u
76          DEV_RES[:len(DEV_DATA), 1] = DEV_DATA
77          DEV_RES[:len(DEV_DATA), 2] = 1
78
79          DEV_RES[len(DEV_DATA):, 1] = random.sample(DEV_NOT_USED, len(DEV_DATA))
80          DEV_RES[len(DEV_DATA):, 2] = 0
81
82          DEV = np.concatenate([DEV, DEV_RES])
83
84          # Crear casos de test
85          TEST_RES = np.empty([len(TEST_DATA) * 2, 3], dtype=np.int32)
86          TEST_RES[:, 0] = u
87          TEST_RES[:len(TEST_DATA), 1] = TEST_DATA
88          TEST_RES[:len(TEST_DATA), 2] = 1
89
90          TEST_RES[len(TEST_DATA):, 1] = random.sample(TEST_NOT_USED, len(TEST_DATA))
91          TEST_RES[len(TEST_DATA):, 2] = 0
92
93          TEST = np.concatenate([TEST, TEST_RES])
94
95      #Mezclar
96      if(shuffle):
97          np.random.seed(seed)
98          np.random.shuffle(TRAIN)
99          np.random.shuffle(DEV)
100         np.random.shuffle(TEST)
101
102
103     return
            TRAIN[:,:2],np.matrix(TRAIN[:,-1]).T,DEV[:,:2],np.matrix(DEV[:,-1]).T,TEST[:,:2],np.matrix(TEST
104
105 def getBatchData(X, Y, batch_size, step, ITRS, RES_ITMS):
106
```

```
107     # Si se ha añadi una iteracin, introducir los datos restantes
108     if (step == ITRS - 1 and RES_ITMS > 0):
109         batch_inputs = X[(ITRS - 1) * batch_size:]
110         batch_context = Y[(ITRS - 1) * batch_size:]
111     # Si no, es un batch normal
112     else:
113         st = step * batch_size
114         nd = (step + 1) * batch_size
115 
116         batch_inputs = X[st:nd]
117         batch_context = Y[st:nd]
118     return(batch_inputs,batch_context)
119 
120 def trainNet(X,Y, DX,DY,TX,TY,config, user_size,train=True, save=True):
121 
122     def getSlope(data):
123         r = linregress(range(len(data)), data)
124         return r.slope
125 
126     batch_size = config['batch_size']
127     learning_rate = config['learning_rate']
128     epochs = config['epochs']
129 
130     user_size = user_size
131     song_size = W2V.shape[0]
132 
133     user_emb_size = 128
134     song_emb_size = 64
135     hidden_size = 32
136 
137     num_sampled = 64  # Ejemplos negativos (Para el NSE)
138 
139     model_name = 'model'
140     model_path = 'models/' + TEST_NAME + '/model'
141     complete_path = model_path + "/" + model_name
142 
143     # Se almacena la loss de dev de las 5 ultimas epochs
144     slope_size = 100
145     train_hist=[]
146     dev_hist=[]
147 
148     # Creacin del grafo de TF.
149     graph = tf.Graph()
150 
151     with graph.as_default():
152 
153         if(config["seed"]!=1):
154             tf.set_random_seed(config["seed"])
155 
156         # Nmero global de iteraciones
157         global_step = tf.Variable(0, dtype=tf.int32, trainable=False, name='global_step')
158 
159         # Datos de entrada
            ------------------------------------------------------------------------------
160         # Array del tamao del batch con las X
161         train_dataset = tf.placeholder(tf.int32, shape=[None, 2], name="train_inputs")
162 
```

```
163         # Array del tamao del batch con las Y asociadas
164         train_labels = tf.placeholder(tf.float32, shape=[None, 1], name="train_labels")
165
166         # Embeddings
                ---------------------------------------------------------------------------------------------
167
168         # Matriz W de embeddigs de las canciones obtenida en el W2V previo
169         #word_embeddings = tf.Variable(W2V, trainable=False, name="word_embeddings")  #
                FUNCIONA???
                https://stackoverflow.com/questions/37326002/is-it-possible-to-make-a-trainable-variable-no
170
171         O1 = tf.Variable(tf.truncated_normal([user_size, user_emb_size], mean=0.0, stddev=1.0
                / math.sqrt(user_emb_size)), name="O1")
172         BO1 = tf.Variable(tf.zeros([user_emb_size]), name="BO1")
173
174         O2 = tf.Variable(tf.truncated_normal([song_size, song_emb_size], mean=0.0, stddev=1.0
                / math.sqrt(song_emb_size)), name="O2")
175         BO2 = tf.Variable(tf.zeros([song_emb_size]), name="BO2")
176
177         T1 = tf.Variable(tf.truncated_normal([user_emb_size, hidden_size], mean=0.0,
                stddev=1.0 / math.sqrt(hidden_size)), name="T1")
178         B1 = tf.Variable(tf.zeros([hidden_size]), name="B1")
179
180         T2 = tf.Variable(tf.truncated_normal([song_emb_size, hidden_size], mean=0.0,
                stddev=1.0 / math.sqrt(hidden_size)), name="T2")
181         B2 = tf.Variable(tf.zeros([hidden_size]), name="B2")
182
183         # Operaciones
                ---------------------------------------------------------------------------------------------
184
185         # Embedding de documento
186         ed = tf.nn.embedding_lookup(O1, train_dataset[:, 0])
187         ed = ed + BO1
188
189         # Embedding de cancion
190         ec = tf.nn.embedding_lookup(O2, train_dataset[:, 1])
191         ec = ec + BO2
192
193         # Transformar a 32 documento
194         hd = tf.matmul(ed,T1)+B1
195
196         # Transformar a 32 cancion
197         hw = tf.matmul(ec,T2)+B2
198
199         # Obtener el producto escalar
200         dot_prod = tf.reduce_sum(tf.multiply(hd, hw), 1, keep_dims=True)
201
202         # aClculo de LOSS y optimizador-
                ---------------------------------------------------------------------------------------------
203
204         # Obtener la loss
205         softplus_batch = tf.nn.softplus((1 - 2 * train_labels) * dot_prod)
206         loss_softplus = tf.reduce_mean(softplus_batch)
207
208         # Minimizar la loss
209         train_step =
                tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(loss=loss_softplus,global_step
```

```
210
211         # Obtener la probabilidad
212         prob = tf.sigmoid(dot_prod)
213
214         # Inicializar variables
215         init = tf.global_variables_initializer()
216
217         # Crear objeto encargado de almacenar la red
218         saver = tf.train.Saver(max_to_keep=1)
219
220     config = tf.ConfigProto()
221     config.gpu_options.allow_growth = True
222     with tf.Session(graph=graph, config=config) as session:
223
224         # We must initialize all variables before we use them.
225         init.run()
226
227         if(save):
228             # Obtener el checkpoint
229             ckpt = tf.train.get_checkpoint_state(model_path)
230
231             # Si existe el checkpoint restaurar
232             if ckpt and ckpt.model_checkpoint_path:
233                 saver.restore(session, ckpt.model_checkpoint_path)
234
235         #Si se desea entrenar
236         if(train):
237             #-----------------------------------------------------------------------------------------------------------
238             # TRAIN
239             #-----------------------------------------------------------------------------------------------------------
240
241             ITRS= int(len(X)/batch_size)
242             RES_ITMS = len(X) - (ITRS*batch_size)
243             if(RES_ITMS>0):ITRS+=1 #Si no es divisin exacta, se aade una iteracin
244
245             for e in range(epochs):
246                 TRAIN_LOSS = 0.0
247
248                 print("Epoch "+str(e))
249
250                 for step in range(ITRS):
251
252                     batch_inputs, batch_context =
                         getBatchData(X,Y,batch_size,step,ITRS,RES_ITMS)
253
254                     feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
255
256                     _, loss_val,softplus, gs = session.run([train_step,
                         loss_softplus,softplus_batch,global_step], feed_dict=feed_dict)
257
258                     TRAIN_LOSS += np.sum(softplus[:, 0])
259
260                 if (save): saver.save(session, complete_path, global_step=global_step)
261
262                 #-----------------------------------------------------------------------------------------------------------
263                 # DEV
264                 #-----------------------------------------------------------------------------------------------------------
```

```
265
266             if(len(DX)>0):
267                 ITRS_TEST = int(len(DX) / batch_size)
268                 RES_ITMS_TEST = len(DX) - (ITRS_TEST * batch_size)
269                 TEST_LOSS = 0.0
270                 if (RES_ITMS_TEST > 0): ITRS_TEST += 1 # Si no es @divisin exacta, se
                        naade una @iteracin
271
272                 for step in range(ITRS_TEST):
273
274                     batch_inputs, batch_context =
                            getBatchData(DX,DY,batch_size,step,ITRS_TEST,RES_ITMS_TEST)
275
276                     feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
277                     loss_val, softplus = session.run([loss_softplus,softplus_batch],
                            feed_dict=feed_dict)
278
279                     TEST_LOSS+=np.sum(softplus[:,0])
280
281                 dev_hist.append(TEST_LOSS/(len(DX)*1.0))
282                 train_hist.append(TRAIN_LOSS/(len(X)*1.0))
283
284
285             if (len(dev_hist) >= slope_size):
286                 # Calcular la pendiente
287                 print(getSlope(dev_hist[-slope_size:]))
288                 print(dev_hist[-slope_size:])
289
290                 if (getSlope(dev_hist[-slope_size:]) > -1e-5 or e==epochs-1):
291
292                     def printArray(data,data2):
293                         for i in range(len(data)):
294                             a=data[i]
295                             b=data2[i]
296                             print(str(a).replace(".",",")+"\t"+str(b).replace(".",","))
297
298                     print("-" * 50)
299                     print(e)
300                     printArray(dev_hist,train_hist)
301                     return
302
303         else:
304             #
                ------------------------------------------------------------------------
305             # TEST FINAL
306             #
                ------------------------------------------------------------------------
307
308             TRUE_POSITIVE = 0
309             FALSE_NEGATIVE = 0
310             FALSE_POSITIVE = 0
311             TRUE_NEGATIVE = 0
312
313             ITRS_TEST = int(len(TX) / batch_size)
314             RES_ITMS_TEST = len(TX) - (ITRS_TEST * batch_size)
315             TEST_LOSS = 0.0
316             # Si no es @divisin exacta, se naade una @iteracin
```

```
317             if (RES_ITMS_TEST > 0): ITRS_TEST += 1
318 
319             for step in range(ITRS_TEST):
320                 # Si se ne aadi una iteracin, introducir los datos restantes
321                 if (step == ITRS_TEST - 1 and RES_ITMS_TEST > 0):
322                     batch_inputs = TX[(ITRS_TEST - 1) * batch_size:]
323                     batch_context = TY[(ITRS_TEST - 1) * batch_size:]
324                 # Si no, es un batch normal
325                 else:
326                     st = step * batch_size
327                     nd = (step + 1) * batch_size
328 
329                     batch_inputs = TX[st:nd]
330                     batch_context = TY[st:nd]
331 
332                 feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
333                 loss_val,loss_batch,  p = session.run([loss_softplus,softplus_batch, prob],
                        feed_dict=feed_dict)
334 
335                 TEST_LOSS += np.sum(loss_batch)
336 
337                 # aClculo de TP/TN/FN/FP para cada batch
338 
339                 for i in range(len(p)):
340                     real = batch_context[i, 0]
341                     pred = p[i, 0]
342 
343                     if (real == 1) and (pred > 0.5): TRUE_POSITIVE += 1
344                     if (real == 1) and (pred <= 0.5): FALSE_NEGATIVE += 1
345                     if (real == 0) and (pred > 0.5): FALSE_POSITIVE += 1
346                     if (real == 0) and (pred <= 0.5): TRUE_NEGATIVE += 1
347 
348             PRECISION = TRUE_POSITIVE / (TRUE_POSITIVE + FALSE_POSITIVE * 1.0)
349             RECALL = TRUE_POSITIVE / (TRUE_POSITIVE + FALSE_NEGATIVE * 1.0)
350 
351             F1 = 2.0 * ((PRECISION * RECALL) / (PRECISION + RECALL))
352 
353             print("-" * 50)
354             print("LOSS:\t" + str(TEST_LOSS / (len(TX) * 1.0)))
355             print("PRECISION:\t" + str(PRECISION))
356             print("RECALL:\t" + str(RECALL))
357             print("F1:\t" + str(F1))
358             print("-" * 50)
359             print("")
360         #----------------------------------------------------------------------------------------------------------------
361 
362 ##################################################################################################################
363 # Llamadas
364 ##################################################################################################################
365 
366 TEST_NAME="test_011"
367 
368 os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
369 
370 #Para conocer los embeddings de las canciones
371 W2V  = np.load('embeddings/w2v/EMB_MATRIX')
372 
```

```
#Se cargan los datos de train de las canciones para conocer los IDs reales
DATA = np.load("datos/Extend/ARRAY_ARRAYS_TRAIN.pkl")

#Obtener una lista con los ID de canciones que no aestn el la matriz de embeddings
ALL = set(list(range(W2V.shape[0])))
REAL = set(list(set(list(map(int, DATA)))))

##########################################################################################################
# TENSORFLOW
##########################################################################################################

#---------------------------------------------------------------------------------------------------------
# Grid Search + Early Stopping
#---------------------------------------------------------------------------------------------------------
'''
X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))
rates = [0.000001,1, 0.01, 0.0001]

for r in rates:
    config = {"batch_size": 1024, "learning_rate": r, "epochs": 1000, "seed": 100}

    print("Learning Rate: "+str(r))
    print("-" * 50)
    trainNet(X, Y, DX, DY, TX, TY, config, user_size, train=True, save=False)
    print("-"*50)

exit()
'''
#---------------------------------------------------------------------------------------------------------
# Train
#---------------------------------------------------------------------------------------------------------
'''
X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=REAL)

#Para esta fase, se une TRAIN y DEV
X = np.concatenate((X,DX),axis=0)
Y = np.concatenate((Y,DY),axis=0)

DX=[]
DY=[]

config = {"batch_size": 1024, "learning_rate": 0.000001, "epochs": 410, "seed": 100}
trainNet(X, Y, DX, DY, TX, TY, config, user_size,train=True,save=True)

exit()
'''
#---------------------------------------------------------------------------------------------------------
# Test
#---------------------------------------------------------------------------------------------------------


X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))

config = {"batch_size": 1024, "learning_rate": 0.000001, "epochs": 410, "seed": 100}
trainNet(X, Y, DX, DY, TX, TY, config, user_size,train=False, save=True)
```

4.6. TEST 06

```
# -*- coding: utf-8 -*-

import os.path
import os
import tensorflow as tf
import numpy as np
import random
import time
import math
import pandas as pd
import pickle
from scipy.stats import linregress
from collections import Counter

#########################################################################################################
# eMtodos
#########################################################################################################

def getData(REAL_SONG_IDS=[],data
    ="datos/Append/ARRAY_ARRAYS_REC.pkl",pastData="datos/Append/ARRAY_ARRAYS_TRAIN_COMPLETE.pkl",shuffl
    seed=100, train=.7):

    DATA = np.load(data)
    PAST_DATA = np.load(pastData)
    REAL_SET = set(REAL_SONG_IDS)

    random.seed(seed)

    #DATA = DATA[:20];print("ELIMINAR FILTRO")

    #User, cancion, +1 o 0
    TRAIN = np.empty([0, 3], dtype=np.int32)
    DEV = np.empty([0, 3], dtype=np.int32)
    TEST = np.empty([0, 3], dtype=np.int32)

    for u in range(len(DATA)):

        # Eliminar canciones escuchadas que no tienen embedding
        USER_PLAYS = DATA[u]
        USER_PLAYS = pd.DataFrame(list(map(int, USER_PLAYS)))
        USER_PLAYS = USER_PLAYS.loc[USER_PLAYS[0].isin(REAL_SONG_IDS)][0].values

        #Separar en train dev test
        TRAIN_ITEMS = int(len(USER_PLAYS)*train)
        DEV_ITEMS = int((len(USER_PLAYS)-TRAIN_ITEMS)/2)

        TRAIN_DATA = USER_PLAYS[:TRAIN_ITEMS]
        DEV_DATA = USER_PLAYS[TRAIN_ITEMS:TRAIN_ITEMS+DEV_ITEMS]
        TEST_DATA = USER_PLAYS[TRAIN_ITEMS+DEV_ITEMS:]

        # Obtener IDS de canciones que no han sido escuchadas en TRAIN_COMPLETE y en TRAIN De
            ahora
        USED_IDS = set(list(map(int, PAST_DATA[u])))
        USED_IDS.update(TRAIN_DATA)
        TRAIN_NOT_USED = list(REAL_SET.difference(USED_IDS))
```

```
53
54          # Obtener IDS de canciones que no han sido escuchadas en TRAIN_COMPLETE , en TRAIN y
                DEV
55          USED_IDS.update(DEV_DATA)
56          DEV_NOT_USED = list(REAL_SET.difference(USED_IDS))
57
58          # Obtener IDS de canciones que no han sido escuchadas en TRAIN_COMPLETE , en TRAIN y
                DEV y TEST
59          USED_IDS.update(TEST_DATA)
60          TEST_NOT_USED = list(REAL_SET.difference(USED_IDS))
61
62          # Crear casos de train
63          TRAIN_RES = np.empty([len(TRAIN_DATA) * 2, 3], dtype=np.int32)
64          TRAIN_RES[:, 0] = u
65          TRAIN_RES[:len(TRAIN_DATA), 1] = TRAIN_DATA
66          TRAIN_RES[:len(TRAIN_DATA), 2] = 1
67
68          TRAIN_RES[len(TRAIN_DATA):, 1] = random.sample(TRAIN_NOT_USED, len(TRAIN_DATA))
69          TRAIN_RES[len(TRAIN_DATA):, 2] = 0
70
71          TRAIN = np.concatenate([TRAIN, TRAIN_RES])
72
73          # Crear casos de dev
74          DEV_RES = np.empty([len(DEV_DATA) * 2, 3], dtype=np.int32)
75          DEV_RES[:, 0] = u
76          DEV_RES[:len(DEV_DATA), 1] = DEV_DATA
77          DEV_RES[:len(DEV_DATA), 2] = 1
78
79          DEV_RES[len(DEV_DATA):, 1] = random.sample(DEV_NOT_USED, len(DEV_DATA))
80          DEV_RES[len(DEV_DATA):, 2] = 0
81
82          DEV = np.concatenate([DEV, DEV_RES])
83
84          # Crear casos de test
85          TEST_RES = np.empty([len(TEST_DATA) * 2, 3], dtype=np.int32)
86          TEST_RES[:, 0] = u
87          TEST_RES[:len(TEST_DATA), 1] = TEST_DATA
88          TEST_RES[:len(TEST_DATA), 2] = 1
89
90          TEST_RES[len(TEST_DATA):, 1] = random.sample(TEST_NOT_USED, len(TEST_DATA))
91          TEST_RES[len(TEST_DATA):, 2] = 0
92
93          TEST = np.concatenate([TEST, TEST_RES])
94
95      #Mezclar
96      if(shuffle):
97          np.random.seed(seed)
98          np.random.shuffle(TRAIN)
99          np.random.shuffle(DEV)
100         np.random.shuffle(TEST)
101
102
103     return
            TRAIN[:,:2],np.matrix(TRAIN[:,-1]).T,DEV[:,:2],np.matrix(DEV[:,-1]).T,TEST[:,:2],np.matrix(TEST
104
105 def getBatchData(X, Y, batch_size, step, ITRS, RES_ITMS):
106
```

```
107     # Si se ha añadi una iteracin, introducir los datos restantes
108     if (step == ITRS - 1 and RES_ITMS > 0):
109         batch_inputs = X[(ITRS - 1) * batch_size:]
110         batch_context = Y[(ITRS - 1) * batch_size:]
111     # Si no, es un batch normal
112     else:
113         st = step * batch_size
114         nd = (step + 1) * batch_size
115 
116         batch_inputs = X[st:nd]
117         batch_context = Y[st:nd]
118     return(batch_inputs,batch_context)
119 
120 def trainNet(X,Y, DX,DY,TX,TY,config, user_size,train=True, save=True):
121 
122     def getSlope(data):
123         r = linregress(range(len(data)), data)
124         return r.slope
125 
126     batch_size = config['batch_size']
127     learning_rate = config['learning_rate']
128     epochs = config['epochs']
129 
130     user_size = user_size
131     song_size = W2V.shape[0]
132 
133     user_emb_size = 128
134     song_emb_size = 64
135     hidden_size = 32
136 
137     num_sampled = 64  # Ejemplos negativos (Para el NSE)
138 
139     model_name = 'model'
140     model_path = 'models/' + TEST_NAME + '/model'
141     complete_path = model_path + "/" + model_name
142 
143     # Se almacena la loss de dev de las 5 ultimas epochs
144     slope_size = 100
145     train_hist=[]
146     dev_hist=[]
147 
148     # Creacin del grafo de TF.
149     graph = tf.Graph()
150 
151     with graph.as_default():
152 
153         if(config["seed"]!=1):
154             tf.set_random_seed(config["seed"])
155 
156         # Nmero global de iteraciones
157         global_step = tf.Variable(0, dtype=tf.int32, trainable=False, name='global_step')
158 
159         # Datos de entrada
160         # Array del tamao del batch con las X
161         train_dataset = tf.placeholder(tf.int32, shape=[None, 2], name="train_inputs")
162 
```

```
163         # Array del tamao del batch con las Y asociadas
164         train_labels = tf.placeholder(tf.float32, shape=[None, 1], name="train_labels")
165
166         # Embeddings
            ------------------------------------------------------------------------------
167
168         D2V_EMB_VRB = tf.Variable(D2V_EMB, trainable=False, name="doc_embeddings")
169
170         O2 = tf.Variable(tf.truncated_normal([song_size, song_emb_size], mean=0.0, stddev=1.0
                / math.sqrt(song_emb_size)), name="O2")
171         BO2 = tf.Variable(tf.zeros([song_emb_size]), name="BO2")
172
173         T1 = tf.Variable(tf.truncated_normal([user_emb_size, hidden_size], mean=0.0,
                stddev=1.0 / math.sqrt(hidden_size)), name="T1")
174         B1 = tf.Variable(tf.zeros([hidden_size]), name="B1")
175
176         T2 = tf.Variable(tf.truncated_normal([song_emb_size, hidden_size], mean=0.0,
                stddev=1.0 / math.sqrt(hidden_size)), name="T2")
177         B2 = tf.Variable(tf.zeros([hidden_size]), name="B2")
178
179         # Operaciones
            ------------------------------------------------------------------------------
180
181         # Embedding de documento
182         ed = tf.nn.embedding_lookup(D2V_EMB_VRB, train_dataset[:, 0])
183
184         # Embedding de cancion
185         ec = tf.nn.embedding_lookup(O2, train_dataset[:, 1])
186         ec = ec + BO2
187
188         # Transformar a 32 documento
189         hd = tf.matmul(ed,T1)+B1
190
191         # Transformar a 32 cancion
192         hw = tf.matmul(ec,T2)+B2
193
194         # Obtener el producto escalar
195         dot_prod = tf.reduce_sum(tf.multiply(hd, hw), 1, keep_dims=True)
196
197         # aClculo de LOSS y optimizador
            ------------------------------------------------------------------------------
198
199         # Obtener la loss
200         softplus_batch = tf.nn.softplus((1 - 2 * train_labels) * dot_prod)
201         loss_softplus = tf.reduce_mean(softplus_batch)
202
203         # Minimizar la loss
204         train_step =
                tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(loss=loss_softplus,global_step
205
206         # Obtener la probabilidad
207         prob = tf.sigmoid(dot_prod)
208
209         # Inicializar variables
210         init = tf.global_variables_initializer()
211
212         # Crear objeto encargado de almacenar la red
```

```
213         saver = tf.train.Saver(max_to_keep=1)
214
215     config = tf.ConfigProto()
216     config.gpu_options.allow_growth = True
217     with tf.Session(graph=graph, config=config) as session:
218
219         # We must initialize all variables before we use them.
220         init.run()
221
222         if(save):
223             # Obtener el checkpoint
224             ckpt = tf.train.get_checkpoint_state(model_path)
225
226             # Si existe el checkpoint restaurar
227             if ckpt and ckpt.model_checkpoint_path:
228                 saver.restore(session, ckpt.model_checkpoint_path)
229
230         #Si se desea entrenar
231         if(train):
232             #-----------------------------------------------------------------------------------------------------------------
233             # TRAIN
234             #-----------------------------------------------------------------------------------------------------------------
235
236             ITRS= int(len(X)/batch_size)
237             RES_ITMS = len(X) - (ITRS*batch_size)
238             if(RES_ITMS>0):ITRS+=1 #Si no es divisin exacta, se naade una iteracin
239
240             for e in range(epochs):
241                 TRAIN_LOSS = 0.0
242
243                 print("Epoch "+str(e))
244
245                 for step in range(ITRS):
246
247                     batch_inputs, batch_context =
                            getBatchData(X,Y,batch_size,step,ITRS,RES_ITMS)
248
249                     feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
250
251                     _, loss_val,softplus, gs = session.run([train_step,
                            loss_softplus,softplus_batch,global_step], feed_dict=feed_dict)
252
253                     TRAIN_LOSS += np.sum(softplus[:, 0])
254
255                 if (save): saver.save(session, complete_path, global_step=global_step)
256
257                 #-----------------------------------------------------------------------------------------------------------
258                 # DEV
259                 #-----------------------------------------------------------------------------------------------------------
260
261                 if(len(DX)>0):
262                     ITRS_TEST = int(len(DX) / batch_size)
263                     RES_ITMS_TEST = len(DX) - (ITRS_TEST * batch_size)
264                     TEST_LOSS = 0.0
265                     if (RES_ITMS_TEST > 0): ITRS_TEST += 1 # Si no es divisin exacta, se
                            naade una iteracin
266
```

```
267                 for step in range(ITRS_TEST):
268
269                     batch_inputs, batch_context =
                            getBatchData(DX,DY,batch_size,step,ITRS_TEST,RES_ITMS_TEST)
270
271                     feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
272                     loss_val, softplus = session.run([loss_softplus,softplus_batch],
                            feed_dict=feed_dict)
273
274                     TEST_LOSS+=np.sum(softplus[:,0])
275
276                 dev_hist.append(TEST_LOSS/(len(DX)*1.0))
277                 train_hist.append(TRAIN_LOSS/(len(X)*1.0))
278
279             if (len(dev_hist) >= slope_size):
280                 # Calcular la pendiente
281                 print(getSlope(dev_hist[-slope_size:]))
282                 print(dev_hist[-slope_size:])
283
284                 if (getSlope(dev_hist[-slope_size:]) > -1e-5 or e==epochs-1):
285
286                     def printArray(data,data2):
287                         for i in range(len(data)):
288                             a=data[i]
289                             b=data2[i]
290                             print(str(a).replace(".",",")+"\t"+str(b).replace(".",","))
291
292                     print("-" * 50)
293                     print(e)
294                     printArray(dev_hist,train_hist)
295                     return
296
297         else:
298             #
                ------------------------------------------------------------------------
299             # TEST FINAL
300             #
                ------------------------------------------------------------------------
301
302             TRUE_POSITIVE = 0
303             FALSE_NEGATIVE = 0
304             FALSE_POSITIVE = 0
305             TRUE_NEGATIVE = 0
306
307             ITRS_TEST = int(len(TX) / batch_size)
308             RES_ITMS_TEST = len(TX) - (ITRS_TEST * batch_size)
309             TEST_LOSS = 0.0
310             # Si no es divisin exacta, se naade una iteracin
311             if (RES_ITMS_TEST > 0): ITRS_TEST += 1
312
313             for step in range(ITRS_TEST):
314                 # Si se naadi una iteracin, introducir los datos restantes
315                 if (step == ITRS_TEST - 1 and RES_ITMS_TEST > 0):
316                     batch_inputs = TX[(ITRS_TEST - 1) * batch_size:]
317                     batch_context = TY[(ITRS_TEST - 1) * batch_size:]
318                 # Si no, es un batch normal
319                 else:
```

```
320                     st = step * batch_size
321                     nd = (step + 1) * batch_size
322 
323                     batch_inputs = TX[st:nd]
324                     batch_context = TY[st:nd]
325 
326                 feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
327                 loss_val,loss_batch,  p = session.run([loss_softplus,softplus_batch, prob],
                        feed_dict=feed_dict)
328 
329                 TEST_LOSS += np.sum(loss_batch)
330 
331                 # aClculo de TP/TN/FN/FP para cada batch
332 
333                 for i in range(len(p)):
334                     real = batch_context[i, 0]
335                     pred = p[i, 0]
336 
337                     if (real == 1) and (pred > 0.5): TRUE_POSITIVE += 1
338                     if (real == 1) and (pred <= 0.5): FALSE_NEGATIVE += 1
339                     if (real == 0) and (pred > 0.5): FALSE_POSITIVE += 1
340                     if (real == 0) and (pred <= 0.5): TRUE_NEGATIVE += 1
341 
342             PRECISION = TRUE_POSITIVE / (TRUE_POSITIVE + FALSE_POSITIVE * 1.0)
343             RECALL = TRUE_POSITIVE / (TRUE_POSITIVE + FALSE_NEGATIVE * 1.0)
344 
345             F1 = 2.0 * ((PRECISION * RECALL) / (PRECISION + RECALL))
346 
347             print("-" * 50)
348             print("LOSS:\t" + str(TEST_LOSS / (len(TX) * 1.0)))
349             print("PRECISION:\t" + str(PRECISION))
350             print("RECALL:\t" + str(RECALL))
351             print("F1:\t" + str(F1))
352             print("-" * 50)
353             print("")
354         #----------------------------------------------------------------------------------------------------
355 
356 ##########################################################################################################
357 # Llamadas
358 ##########################################################################################################
359 
360 TEST_NAME="test_061"
361 
362 os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
363 
364 #Se obtinenen cada uno de los embeddings aprendidos (incluidas las canciones)
365 D2V_DM_COMPLETE   = np.load('embeddings/d2v_dm/complete/EMB_MATRIX')
366 D2V_DBOW_COMPLETE = np.load('embeddings/d2v_dbow/complete/EMB_MATRIX')
367 W2V  = np.load('embeddings/w2v/EMB_MATRIX')
368 
369 #Se concatenan los embeddings de los usuarios para generar el perfil
370 D2V_EMB = np.concatenate((D2V_DM_COMPLETE,D2V_DBOW_COMPLETE),axis=1) # El consolidado =
        Consolidado DM + Consolidado DBOW
371 
372 #Se cargan los datos de train de las canciones para conocer los IDs reales
373 DATA = np.load("datos/Extend/ARRAY_ARRAYS_TRAIN.pkl")
374 
```

```
#Obtener una lista con los ID de canciones que no aestn el la matriz de embeddings
ALL = set(list(range(W2V.shape[0])))
REAL = set(list(set(list(map(int, DATA)))))
NOT_EXISTS = list(ALL.difference(REAL))

################################################################################################################
# TENSORFLOW
################################################################################################################

#---------------------------------------------------------------------------------------------------------------
# Grid Search + Early Stopping
#---------------------------------------------------------------------------------------------------------------
'''
X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))
rates = [0.000001,1, 0.01, 0.0001]

for r in rates:
    config = {"batch_size": 1024, "learning_rate": r, "epochs": 1000, "seed": 100}

    print("Learning Rate: "+str(r))
    print("-" * 50)
    trainNet(X, Y, DX, DY, TX, TY, config, user_size, train=True, save=False)
    print("-"*50)

exit()
'''
#---------------------------------------------------------------------------------------------------------------
# Train
#---------------------------------------------------------------------------------------------------------------
'''
X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))

#Para esta fase, se une TRAIN y DEV
X = np.concatenate((X,DX),axis=0)
Y = np.concatenate((Y,DY),axis=0)

DX=[]
DY=[]

config = {"batch_size": 1024, "learning_rate": 0.000001, "epochs": 464, "seed": 100}
trainNet(X, Y, DX, DY, TX, TY, config, user_size,train=True,save=True)

exit()
'''
#---------------------------------------------------------------------------------------------------------------
# Test
#---------------------------------------------------------------------------------------------------------------


X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))

config = {"batch_size": 1024, "learning_rate": 0.000001, "epochs": 464, "seed": 100}
trainNet(X, Y, DX, DY, TX, TY, config, user_size,train=False, save=True)
```

4.7. TEST 07

```
# -*- coding: utf-8 -*-

import os.path
import os
import tensorflow as tf
import numpy as np
import random
import time
import math
import pandas as pd
import pickle
from scipy.stats import linregress
from collections import Counter

################################################################################################################
# eMtodos
################################################################################################################

def getData(REAL_SONG_IDS=[],data
    ="datos/Append/ARRAY_ARRAYS_REC.pkl",pastData="datos/Append/ARRAY_ARRAYS_TRAIN_COMPLETE.pkl",shuffl
    seed=100, train=.7):

    DATA = np.load(data)
    PAST_DATA = np.load(pastData)
    REAL_SET = set(REAL_SONG_IDS)

    random.seed(seed)

    #DATA = DATA[:20];print("ELIMINAR FILTRO")

    #User, cancion, +1 o 0
    TRAIN = np.empty([0, 3], dtype=np.int32)
    DEV = np.empty([0, 3], dtype=np.int32)
    TEST = np.empty([0, 3], dtype=np.int32)

    for u in range(len(DATA)):

        # Eliminar canciones escuchadas que no tienen embedding
        USER_PLAYS = DATA[u]
        USER_PLAYS = pd.DataFrame(list(map(int, USER_PLAYS)))
        USER_PLAYS = USER_PLAYS.loc[USER_PLAYS[0].isin(REAL_SONG_IDS)][0].values

        #Separar en train dev test
        TRAIN_ITEMS = int(len(USER_PLAYS)*train)
        DEV_ITEMS = int((len(USER_PLAYS)-TRAIN_ITEMS)/2)

        TRAIN_DATA = USER_PLAYS[:TRAIN_ITEMS]
        DEV_DATA = USER_PLAYS[TRAIN_ITEMS:TRAIN_ITEMS+DEV_ITEMS]
        TEST_DATA = USER_PLAYS[TRAIN_ITEMS+DEV_ITEMS:]

        # Obtener IDS de canciones que no han sido escuchadas en TRAIN_COMPLETE y en TRAIN De
            ahora
        USED_IDS = set(list(map(int, PAST_DATA[u])))
        USED_IDS.update(TRAIN_DATA)
        TRAIN_NOT_USED = list(REAL_SET.difference(USED_IDS))
```

```
53
54          # Obtener IDS de canciones que no han sido escuchadas en TRAIN_COMPLETE , en TRAIN y
                DEV
55          USED_IDS.update(DEV_DATA)
56          DEV_NOT_USED = list(REAL_SET.difference(USED_IDS))
57
58          # Obtener IDS de canciones que no han sido escuchadas en TRAIN_COMPLETE , en TRAIN y
                DEV y TEST
59          USED_IDS.update(TEST_DATA)
60          TEST_NOT_USED = list(REAL_SET.difference(USED_IDS))
61
62          # Crear casos de train
63          TRAIN_RES = np.empty([len(TRAIN_DATA) * 2, 3], dtype=np.int32)
64          TRAIN_RES[:, 0] = u
65          TRAIN_RES[:len(TRAIN_DATA), 1] = TRAIN_DATA
66          TRAIN_RES[:len(TRAIN_DATA), 2] = 1
67
68          TRAIN_RES[len(TRAIN_DATA):, 1] = random.sample(TRAIN_NOT_USED, len(TRAIN_DATA))
69          TRAIN_RES[len(TRAIN_DATA):, 2] = 0
70
71          TRAIN = np.concatenate([TRAIN, TRAIN_RES])
72
73          # Crear casos de dev
74          DEV_RES = np.empty([len(DEV_DATA) * 2, 3], dtype=np.int32)
75          DEV_RES[:, 0] = u
76          DEV_RES[:len(DEV_DATA), 1] = DEV_DATA
77          DEV_RES[:len(DEV_DATA), 2] = 1
78
79          DEV_RES[len(DEV_DATA):, 1] = random.sample(DEV_NOT_USED, len(DEV_DATA))
80          DEV_RES[len(DEV_DATA):, 2] = 0
81
82          DEV = np.concatenate([DEV, DEV_RES])
83
84          # Crear casos de test
85          TEST_RES = np.empty([len(TEST_DATA) * 2, 3], dtype=np.int32)
86          TEST_RES[:, 0] = u
87          TEST_RES[:len(TEST_DATA), 1] = TEST_DATA
88          TEST_RES[:len(TEST_DATA), 2] = 1
89
90          TEST_RES[len(TEST_DATA):, 1] = random.sample(TEST_NOT_USED, len(TEST_DATA))
91          TEST_RES[len(TEST_DATA):, 2] = 0
92
93          TEST = np.concatenate([TEST, TEST_RES])
94
95      #Mezclar
96      if(shuffle):
97          np.random.seed(seed)
98          np.random.shuffle(TRAIN)
99          np.random.shuffle(DEV)
100         np.random.shuffle(TEST)
101
102
103     return
            TRAIN[:,:2],np.matrix(TRAIN[:,-1]).T,DEV[:,:2],np.matrix(DEV[:,-1]).T,TEST[:,:2],np.matrix(TEST
104
105 def getBatchData(X, Y, batch_size, step, ITRS, RES_ITMS):
106
```

```
107     # Si se ha añadi una iteracin, introducir los datos restantes
108     if (step == ITRS - 1 and RES_ITMS > 0):
109         batch_inputs = X[(ITRS - 1) * batch_size:]
110         batch_context = Y[(ITRS - 1) * batch_size:]
111     # Si no, es un batch normal
112     else:
113         st = step * batch_size
114         nd = (step + 1) * batch_size
115 
116         batch_inputs = X[st:nd]
117         batch_context = Y[st:nd]
118     return(batch_inputs,batch_context)
119 
120 def trainNet(X,Y, DX,DY,TX,TY,config, user_size,train=True, save=True):
121 
122     def getSlope(data):
123         r = linregress(range(len(data)), data)
124         return r.slope
125 
126     batch_size = config['batch_size']
127     learning_rate = config['learning_rate']
128     epochs = config['epochs']
129 
130     user_size = user_size
131     song_size = W2V.shape[0]
132 
133     user_emb_size = 128
134     song_emb_size = 64
135     hidden_size = 32
136 
137     num_sampled = 64  # Ejemplos negativos (Para el NSE)
138 
139     model_name = 'model'
140     model_path = 'models/' + TEST_NAME + '/model'
141     complete_path = model_path + "/" + model_name
142 
143     # Se almacena la loss de dev de las 5 ultimas epochs
144     slope_size = 100
145     train_hist=[]
146     dev_hist=[]
147 
148     # Creacin del grafo de TF.
149     graph = tf.Graph()
150 
151     with graph.as_default():
152 
153         if(config["seed"]!=1):
154             tf.set_random_seed(config["seed"])
155 
156         # Nmero global de iteraciones
157         global_step = tf.Variable(0, dtype=tf.int32, trainable=False, name='global_step')
158 
159         # Datos de entrada
160         # Array del tamao del batch con las X
161         train_dataset = tf.placeholder(tf.int32, shape=[None, 2], name="train_inputs")
162 
```

```
163         # Array del tamao del batch con las Y asociadas
164         train_labels = tf.placeholder(tf.float32, shape=[None, 1], name="train_labels")
165
166         # Embeddings
            ------------------------------------------------------------------------------
167
168         D2V_EMB_VRB = tf.Variable(D2V_EMB, trainable=False, name="doc_embeddings")
169
170         O2 = tf.Variable(tf.truncated_normal([song_size, song_emb_size], mean=0.0, stddev=1.0
                / math.sqrt(song_emb_size)), name="O2")
171         BO2 = tf.Variable(tf.zeros([song_emb_size]), name="BO2")
172
173         T1 = tf.Variable(tf.truncated_normal([user_emb_size, hidden_size], mean=0.0,
                stddev=1.0 / math.sqrt(hidden_size)), name="T1")
174         B1 = tf.Variable(tf.zeros([hidden_size]), name="B1")
175
176         T2 = tf.Variable(tf.truncated_normal([song_emb_size, hidden_size], mean=0.0,
                stddev=1.0 / math.sqrt(hidden_size)), name="T2")
177         B2 = tf.Variable(tf.zeros([hidden_size]), name="B2")
178
179         # Operaciones
            ------------------------------------------------------------------------------
180
181         # Embedding de documento
182         ed = tf.nn.embedding_lookup(D2V_EMB_VRB, train_dataset[:, 0])
183
184         # Embedding de cancion
185         ec = tf.nn.embedding_lookup(O2, train_dataset[:, 1])
186         ec = ec + BO2
187
188         # Transformar a 32 documento
189         hd = tf.matmul(ed,T1)+B1
190
191         # Transformar a 32 cancion
192         hw = tf.matmul(ec,T2)+B2
193
194         # Obtener el producto escalar
195         dot_prod = tf.reduce_sum(tf.multiply(hd, hw), 1, keep_dims=True)
196
197         # aClculo de LOSS y optimizador
            ------------------------------------------------------------------------------
198
199         # Obtener la loss
200         softplus_batch = tf.nn.softplus((1 - 2 * train_labels) * dot_prod)
201         loss_softplus = tf.reduce_mean(softplus_batch)
202
203         # Minimizar la loss
204         train_step =
                tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(loss=loss_softplus,global_step
205
206         # Obtener la probabilidad
207         prob = tf.sigmoid(dot_prod)
208
209         # Inicializar variables
210         init = tf.global_variables_initializer()
211
212         # Crear objeto encargado de almacenar la red
```

```
213         saver = tf.train.Saver(max_to_keep=1)
214
215     config = tf.ConfigProto()
216     config.gpu_options.allow_growth = True
217     with tf.Session(graph=graph, config=config) as session:
218
219         # We must initialize all variables before we use them.
220         init.run()
221
222         if(save):
223             # Obtener el checkpoint
224             ckpt = tf.train.get_checkpoint_state(model_path)
225
226             # Si existe el checkpoint restaurar
227             if ckpt and ckpt.model_checkpoint_path:
228                 saver.restore(session, ckpt.model_checkpoint_path)
229
230         #Si se desea entrenar
231         if(train):
232             #-----------------------------------------------------------------------------------------------
233             # TRAIN
234             #-----------------------------------------------------------------------------------------------
235
236             ITRS= int(len(X)/batch_size)
237             RES_ITMS = len(X) - (ITRS*batch_size)
238             if(RES_ITMS>0):ITRS+=1 #Si no es divisin exacta, se naade una iteracin
239
240             for e in range(epochs):
241                 TRAIN_LOSS = 0.0
242
243                 print("Epoch "+str(e))
244
245                 for step in range(ITRS):
246
247                     batch_inputs, batch_context =
                            getBatchData(X,Y,batch_size,step,ITRS,RES_ITMS)
248
249                     feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
250
251                     _, loss_val,softplus, gs = session.run([train_step,
                            loss_softplus,softplus_batch,global_step], feed_dict=feed_dict)
252
253                     TRAIN_LOSS += np.sum(softplus[:, 0])
254
255                 if (save): saver.save(session, complete_path, global_step=global_step)
256
257                 #-----------------------------------------------------------------------------------------------
258                 # DEV
259                 #-----------------------------------------------------------------------------------------------
260
261                 if(len(DX)>0):
262                     ITRS_TEST = int(len(DX) / batch_size)
263                     RES_ITMS_TEST = len(DX) - (ITRS_TEST * batch_size)
264                     TEST_LOSS = 0.0
265                     if (RES_ITMS_TEST > 0): ITRS_TEST += 1 # Si no es divisin exacta, se
                            naade una iteracin
266
```

```
267                 for step in range(ITRS_TEST):
268
269                     batch_inputs, batch_context =
                            getBatchData(DX,DY,batch_size,step,ITRS_TEST,RES_ITMS_TEST)
270
271                     feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
272                     loss_val, softplus = session.run([loss_softplus,softplus_batch],
                            feed_dict=feed_dict)
273
274                     TEST_LOSS+=np.sum(softplus[:,0])
275
276                 dev_hist.append(TEST_LOSS/(len(DX)*1.0))
277                 train_hist.append(TRAIN_LOSS/(len(X)*1.0))
278
279             if (len(dev_hist) >= slope_size):
280                 # Calcular la pendiente
281                 print(getSlope(dev_hist[-slope_size:]))
282                 print(dev_hist[-slope_size:])
283
284                 if (getSlope(dev_hist[-slope_size:]) > -1e-5 or e==epochs-1):
285
286                     def printArray(data,data2):
287                         for i in range(len(data)):
288                             a=data[i]
289                             b=data2[i]
290                             print(str(a).replace(".",",")+"\t"+str(b).replace(".",","))
291
292                     print("-" * 50)
293                     print(e)
294                     printArray(dev_hist,train_hist)
295                     return
296
297         else:
298             #
                ------------------------------------------------------------------------
299             # TEST FINAL
300             #
                ------------------------------------------------------------------------
301
302             TRUE_POSITIVE = 0
303             FALSE_NEGATIVE = 0
304             FALSE_POSITIVE = 0
305             TRUE_NEGATIVE = 0
306
307             ITRS_TEST = int(len(TX) / batch_size)
308             RES_ITMS_TEST = len(TX) - (ITRS_TEST * batch_size)
309             TEST_LOSS = 0.0
310             # Si no es divisin exacta, se naade una iteracin
311             if (RES_ITMS_TEST > 0): ITRS_TEST += 1
312
313             for step in range(ITRS_TEST):
314                 # Si se naadi una iteracin, introducir los datos restantes
315                 if (step == ITRS_TEST - 1 and RES_ITMS_TEST > 0):
316                     batch_inputs = TX[(ITRS_TEST - 1) * batch_size:]
317                     batch_context = TY[(ITRS_TEST - 1) * batch_size:]
318                 # Si no, es un batch normal
319                 else:
```

```
320                     st = step * batch_size
321                     nd = (step + 1) * batch_size
322 
323                     batch_inputs = TX[st:nd]
324                     batch_context = TY[st:nd]
325 
326                 feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
327                 loss_val,loss_batch,  p = session.run([loss_softplus,softplus_batch, prob],
                        feed_dict=feed_dict)
328 
329                 TEST_LOSS += np.sum(loss_batch)
330 
331                 # aClculo de TP/TN/FN/FP para cada batch
332 
333                 for i in range(len(p)):
334                     real = batch_context[i, 0]
335                     pred = p[i, 0]
336 
337                     if (real == 1) and (pred > 0.5): TRUE_POSITIVE += 1
338                     if (real == 1) and (pred <= 0.5): FALSE_NEGATIVE += 1
339                     if (real == 0) and (pred > 0.5): FALSE_POSITIVE += 1
340                     if (real == 0) and (pred <= 0.5): TRUE_NEGATIVE += 1
341 
342             PRECISION = TRUE_POSITIVE / (TRUE_POSITIVE + FALSE_POSITIVE * 1.0)
343             RECALL = TRUE_POSITIVE / (TRUE_POSITIVE + FALSE_NEGATIVE * 1.0)
344 
345             F1 = 2.0 * ((PRECISION * RECALL) / (PRECISION + RECALL))
346 
347             print("-" * 50)
348             print("LOSS:\t" + str(TEST_LOSS / (len(TX) * 1.0)))
349             print("PRECISION:\t" + str(PRECISION))
350             print("RECALL:\t" + str(RECALL))
351             print("F1:\t" + str(F1))
352             print("-" * 50)
353             print("")
354         #----------------------------------------------------------------------------------------------------
355 
356 ##########################################################################################################
357 # Llamadas
358 ##########################################################################################################
359 
360 TEST_NAME="test_071"
361 
362 os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
363 
364 #Se obtinenen cada uno de los embeddings aprendidos (incluidas las canciones)
365 D2V_DM_PRESENT   = np.load('embeddings/d2v_dm/present/EMB_MATRIX')
366 D2V_DBOW_PRESENT  = np.load('embeddings/d2v_dbow/present/EMB_MATRIX')
367 W2V  = np.load('embeddings/w2v/EMB_MATRIX')
368 
369 #Se concatenan los embeddings de los usuarios para generar el perfil
370 D2V_EMB = np.concatenate((D2V_DM_PRESENT,D2V_DBOW_PRESENT),axis=1) # El reciente = Reciente
    DM + Reciente DBOW
371 
372 
373 #Se cargan los datos de train de las canciones para conocer los IDs reales
374 DATA = np.load("datos/Extend/ARRAY_ARRAYS_TRAIN.pkl")
```

```
375
376 #Obtener una lista con los ID de canciones que no aestn el la matriz de embeddings
377 ALL = set(list(range(W2V.shape[0])))
378 REAL = set(list(set(list(map(int, DATA)))))
379 NOT_EXISTS = list(ALL.difference(REAL))
380
381 ##########################################################################################################
382 # TENSORFLOW
383 ##########################################################################################################
384
385 #---------------------------------------------------------------------------------------------------------
386 # Grid Search + Early Stopping
387 #---------------------------------------------------------------------------------------------------------
388 '''
389 X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))
390 rates = [0.000001,1, 0.01, 0.0001]
391
392 for r in rates:
393     config = {"batch_size": 1024, "learning_rate": r, "epochs": 1000, "seed": 100}
394
395     print("Learning Rate: "+str(r))
396     print("-" * 50)
397     trainNet(X, Y, DX, DY, TX, TY, config, user_size, train=True, save=False)
398     print("-"*50)
399
400 exit()
401 '''
402 #---------------------------------------------------------------------------------------------------------
403 # Train
404 #---------------------------------------------------------------------------------------------------------
405 '''
406 X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))
407
408 #Para esta fase, se une TRAIN y DEV
409 X = np.concatenate((X,DX),axis=0)
410 Y = np.concatenate((Y,DY),axis=0)
411
412 DX=[]
413 DY=[]
414
415 config = {"batch_size": 1024, "learning_rate": 0.000001, "epochs": 401, "seed": 100}
416 trainNet(X, Y, DX, DY, TX, TY, config, user_size,train=True,save=True)
417
418 exit()
419 '''
420 #---------------------------------------------------------------------------------------------------------
421 # Test
422 #---------------------------------------------------------------------------------------------------------
423
424
425 X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))
426
427 config = {"batch_size": 1024, "learning_rate": 0.000001, "epochs": 401, "seed": 100}
428 trainNet(X, Y, DX, DY, TX, TY, config, user_size,train=False, save=True)
```

4.8. TEST 08

```
# -*- coding: utf-8 -*-

import os.path
import os
import tensorflow as tf
import numpy as np
import random
import time
import math
import pandas as pd
import pickle
from scipy.stats import linregress
from collections import Counter

#########################################################################################################
# eMtodos
#########################################################################################################

def getData(REAL_SONG_IDS=[],data
    ="datos/Append/ARRAY_ARRAYS_REC.pkl",pastData="datos/Append/ARRAY_ARRAYS_TRAIN_COMPLETE.pkl",shuffl
    seed=100, train=.7):

    DATA = np.load(data)
    PAST_DATA = np.load(pastData)
    REAL_SET = set(REAL_SONG_IDS)

    random.seed(seed)

    #DATA = DATA[:20];print("ELIMINAR FILTRO")

    #User, cancion, +1 o 0
    TRAIN = np.empty([0, 3], dtype=np.int32)
    DEV = np.empty([0, 3], dtype=np.int32)
    TEST = np.empty([0, 3], dtype=np.int32)

    for u in range(len(DATA)):

        # Eliminar canciones escuchadas que no tienen embedding
        USER_PLAYS = DATA[u]
        USER_PLAYS = pd.DataFrame(list(map(int, USER_PLAYS)))
        USER_PLAYS = USER_PLAYS.loc[USER_PLAYS[0].isin(REAL_SONG_IDS)][0].values

        #Separar en train dev test
        TRAIN_ITEMS = int(len(USER_PLAYS)*train)
        DEV_ITEMS = int((len(USER_PLAYS)-TRAIN_ITEMS)/2)

        TRAIN_DATA = USER_PLAYS[:TRAIN_ITEMS]
        DEV_DATA = USER_PLAYS[TRAIN_ITEMS:TRAIN_ITEMS+DEV_ITEMS]
        TEST_DATA = USER_PLAYS[TRAIN_ITEMS+DEV_ITEMS:]

        # Obtener IDS de canciones que no han sido escuchadas en TRAIN_COMPLETE y en TRAIN De
            ahora
        USED_IDS = set(list(map(int, PAST_DATA[u])))
        USED_IDS.update(TRAIN_DATA)
        TRAIN_NOT_USED = list(REAL_SET.difference(USED_IDS))
```

```
53
54          # Obtener IDS de canciones que no han sido escuchadas en TRAIN_COMPLETE , en TRAIN y
                DEV
55          USED_IDS.update(DEV_DATA)
56          DEV_NOT_USED = list(REAL_SET.difference(USED_IDS))
57
58          # Obtener IDS de canciones que no han sido escuchadas en TRAIN_COMPLETE , en TRAIN y
                DEV y TEST
59          USED_IDS.update(TEST_DATA)
60          TEST_NOT_USED = list(REAL_SET.difference(USED_IDS))
61
62          # Crear casos de train
63          TRAIN_RES = np.empty([len(TRAIN_DATA) * 2, 3], dtype=np.int32)
64          TRAIN_RES[:, 0] = u
65          TRAIN_RES[:len(TRAIN_DATA), 1] = TRAIN_DATA
66          TRAIN_RES[:len(TRAIN_DATA), 2] = 1
67
68          TRAIN_RES[len(TRAIN_DATA):, 1] = random.sample(TRAIN_NOT_USED, len(TRAIN_DATA))
69          TRAIN_RES[len(TRAIN_DATA):, 2] = 0
70
71          TRAIN = np.concatenate([TRAIN, TRAIN_RES])
72
73          # Crear casos de dev
74          DEV_RES = np.empty([len(DEV_DATA) * 2, 3], dtype=np.int32)
75          DEV_RES[:, 0] = u
76          DEV_RES[:len(DEV_DATA), 1] = DEV_DATA
77          DEV_RES[:len(DEV_DATA), 2] = 1
78
79          DEV_RES[len(DEV_DATA):, 1] = random.sample(DEV_NOT_USED, len(DEV_DATA))
80          DEV_RES[len(DEV_DATA):, 2] = 0
81
82          DEV = np.concatenate([DEV, DEV_RES])
83
84          # Crear casos de test
85          TEST_RES = np.empty([len(TEST_DATA) * 2, 3], dtype=np.int32)
86          TEST_RES[:, 0] = u
87          TEST_RES[:len(TEST_DATA), 1] = TEST_DATA
88          TEST_RES[:len(TEST_DATA), 2] = 1
89
90          TEST_RES[len(TEST_DATA):, 1] = random.sample(TEST_NOT_USED, len(TEST_DATA))
91          TEST_RES[len(TEST_DATA):, 2] = 0
92
93          TEST = np.concatenate([TEST, TEST_RES])
94
95      #Mezclar
96      if(shuffle):
97          np.random.seed(seed)
98          np.random.shuffle(TRAIN)
99          np.random.shuffle(DEV)
100         np.random.shuffle(TEST)
101
102
103     return
            TRAIN[:,:2],np.matrix(TRAIN[:,-1]).T,DEV[:,:2],np.matrix(DEV[:,-1]).T,TEST[:,:2],np.matrix(TEST
104
105 def getBatchData(X, Y, batch_size, step, ITRS, RES_ITMS):
106
```

```
107     # Si se ha añadi una iteracin, introducir los datos restantes
108     if (step == ITRS - 1 and RES_ITMS > 0):
109         batch_inputs = X[(ITRS - 1) * batch_size:]
110         batch_context = Y[(ITRS - 1) * batch_size:]
111     # Si no, es un batch normal
112     else:
113         st = step * batch_size
114         nd = (step + 1) * batch_size
115 
116         batch_inputs = X[st:nd]
117         batch_context = Y[st:nd]
118     return(batch_inputs,batch_context)
119 
120 def trainNet(X,Y, DX,DY,TX,TY,config, user_size,train=True, save=True):
121 
122     def getSlope(data):
123         r = linregress(range(len(data)), data)
124         return r.slope
125 
126     batch_size = config['batch_size']
127     learning_rate = config['learning_rate']
128     epochs = config['epochs']
129 
130     user_size = user_size
131     song_size = W2V.shape[0]
132 
133     user_emb_size = 128
134     user_concat_size = user_emb_size * 2
135     song_emb_size = 64
136     hidden_size = 32
137 
138     num_sampled = 64  # Ejemplos negativos (Para el NSE)
139 
140     model_name = 'model'
141     model_path = 'models/' + TEST_NAME + '/model'
142     complete_path = model_path + "/" + model_name
143 
144     # Se almacena la loss de dev de las 5 ultimas epochs
145     slope_size = 100
146     train_hist=[]
147     dev_hist=[]
148 
149     # Creacin del grafo de TF.
150     graph = tf.Graph()
151 
152     with graph.as_default():
153 
154         if(config["seed"]!=1):
155             tf.set_random_seed(config["seed"])
156 
157         # Nmero global de iteraciones
158         global_step = tf.Variable(0, dtype=tf.int32, trainable=False, name='global_step')
159 
160         # Datos de entrada
161         # Array del tamao del batch con las X
162         train_dataset = tf.placeholder(tf.int32, shape=[None, 2], name="train_inputs")
```

```
163
164         # Array del tamao del batch con las Y asociadas
165         train_labels = tf.placeholder(tf.float32, shape=[None, 1], name="train_labels")
166
167         # Embeddings
            ------------------------------------------------------------------------------
168
169         D2V_EMB_VRB = tf.Variable(D2V_EMB, trainable=False, name="doc_embeddings")
170
171         T0 = tf.Variable(tf.truncated_normal([user_concat_size, user_emb_size], mean=0.0,
                stddev=1.0 / math.sqrt(hidden_size)), name="T0")
172         B0 = tf.Variable(tf.zeros([user_emb_size]), name="B0")
173
174         O2 = tf.Variable(tf.truncated_normal([song_size, song_emb_size], mean=0.0, stddev=1.0
                / math.sqrt(song_emb_size)), name="O2")
175         BO2 = tf.Variable(tf.zeros([song_emb_size]), name="BO2")
176
177         T1 = tf.Variable(tf.truncated_normal([user_emb_size, hidden_size], mean=0.0,
                stddev=1.0 / math.sqrt(hidden_size)), name="T1")
178         B1 = tf.Variable(tf.zeros([hidden_size]), name="B1")
179
180         T2 = tf.Variable(tf.truncated_normal([song_emb_size, hidden_size], mean=0.0,
                stddev=1.0 / math.sqrt(hidden_size)), name="T2")
181         B2 = tf.Variable(tf.zeros([hidden_size]), name="B2")
182
183         # Operaciones
            ------------------------------------------------------------------------------
184
185         # Embedding de documento
186         ed = tf.nn.embedding_lookup(D2V_EMB_VRB, train_dataset[:, 0])
187
188         #Transformar concat documento de 256 a 128
189         ed = tf.matmul(ed,T0)+B0
190
191         # Embedding de cancion
192         ec = tf.nn.embedding_lookup(O2, train_dataset[:, 1])
193         ec = ec + BO2
194
195         # Transformar a 32 documento
196         hd = tf.matmul(ed,T1)+B1
197
198         # Transformar a 32 cancion
199         hw = tf.matmul(ec,T2)+B2
200
201         # Obtener el producto escalar
202         dot_prod = tf.reduce_sum(tf.multiply(hd, hw), 1, keep_dims=True)
203
204         # aClculo de LOSS y optimizador
            ------------------------------------------------------------------------------
205
206         # Obtener la loss
207         softplus_batch = tf.nn.softplus((1 - 2 * train_labels) * dot_prod)
208         loss_softplus = tf.reduce_mean(softplus_batch)
209
210         # Minimizar la loss
211         train_step =
                tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(loss=loss_softplus,global_step
```

```
212
213         # Obtener la probabilidad
214         prob = tf.sigmoid(dot_prod)
215
216         # Inicializar variables
217         init = tf.global_variables_initializer()
218
219         # Crear objeto encargado de almacenar la red
220         saver = tf.train.Saver(max_to_keep=1)
221
222     config = tf.ConfigProto()
223     config.gpu_options.allow_growth = True
224     with tf.Session(graph=graph,config=config) as session:
225     #with tf.Session(graph=graph) as session:
226
227         # We must initialize all variables before we use them.
228         init.run()
229
230         if(save):
231             # Obtener el checkpoint
232             ckpt = tf.train.get_checkpoint_state(model_path)
233
234             # Si existe el checkpoint restaurar
235             if ckpt and ckpt.model_checkpoint_path:
236                 saver.restore(session, ckpt.model_checkpoint_path)
237
238         #Si se desea entrenar
239         if(train):
240             #-----------------------------------------------------------------------------------------------------------------
241             # TRAIN
242             #-----------------------------------------------------------------------------------------------------------------
243
244             ITRS= int(len(X)/batch_size)
245             RES_ITMS = len(X) - (ITRS*batch_size)
246             if(RES_ITMS>0):ITRS+=1 #Si no es divisin exacta, se aade una iteracin
247
248             for e in range(epochs):
249                 TRAIN_LOSS = 0.0
250
251                 print("Epoch "+str(e))
252
253                 for step in range(ITRS):
254
255                     batch_inputs, batch_context =
                            getBatchData(X,Y,batch_size,step,ITRS,RES_ITMS)
256
257                     feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
258
259                     _, loss_val,softplus, gs = session.run([train_step,
                            loss_softplus,softplus_batch,global_step], feed_dict=feed_dict)
260
261                     TRAIN_LOSS += np.sum(softplus[:, 0])
262
263                 if (save): saver.save(session, complete_path, global_step=global_step)
264
265                 #-----------------------------------------------------------------------------------------------------------------
266                 # DEV
```

```
267             #-----------------------------------------------------------------------------------------------------
268
269             if(len(DX)>0):
270                 ITRS_TEST = int(len(DX) / batch_size)
271                 RES_ITMS_TEST = len(DX) - (ITRS_TEST * batch_size)
272                 TEST_LOSS = 0.0
273                 if (RES_ITMS_TEST > 0): ITRS_TEST += 1 # Si no es divisin exacta, se
                        naade una iteracin
274
275                 for step in range(ITRS_TEST):
276
277                     batch_inputs, batch_context =
                            getBatchData(DX,DY,batch_size,step,ITRS_TEST,RES_ITMS_TEST)
278
279                     feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
280                     loss_val, softplus = session.run([loss_softplus,softplus_batch],
                            feed_dict=feed_dict)
281
282                     TEST_LOSS+=np.sum(softplus[:,0])
283
284                 dev_hist.append(TEST_LOSS/(len(DX)*1.0))
285                 train_hist.append(TRAIN_LOSS/(len(X)*1.0))
286
287             if (len(dev_hist) >= slope_size):
288                 # Calcular la pendiente
289                 print(getSlope(dev_hist[-slope_size:]))
290                 print(dev_hist[-slope_size:])
291
292                 if (getSlope(dev_hist[-slope_size:]) > -1e-5 or e==epochs-1):
293
294                     def printArray(data,data2):
295                         for i in range(len(data)):
296                             a=data[i]
297                             b=data2[i]
298                             print(str(a).replace(".",",")+"\t"+str(b).replace(".",","))
299
300                     print("-" * 50)
301                     print(e)
302                     printArray(dev_hist,train_hist)
303                     return
304
305         else:
306             #-----------------------------------------------------------------------------------------------------
307             # TEST FINAL
308             #-----------------------------------------------------------------------------------------------------
309
310             TRUE_POSITIVE = 0
311             FALSE_NEGATIVE = 0
312             FALSE_POSITIVE = 0
313             TRUE_NEGATIVE = 0
314
315             ITRS_TEST = int(len(TX) / batch_size)
316             RES_ITMS_TEST = len(TX) - (ITRS_TEST * batch_size)
317             TEST_LOSS = 0.0
318             # Si no es divisin exacta, se naade una iteracin
```

```
319             if (RES_ITMS_TEST > 0): ITRS_TEST += 1
320 
321             for step in range(ITRS_TEST):
322                 # Si se ne aadi una iteracin, introducir los datos restantes
323                 if (step == ITRS_TEST - 1 and RES_ITMS_TEST > 0):
324                     batch_inputs = TX[(ITRS_TEST - 1) * batch_size:]
325                     batch_context = TY[(ITRS_TEST - 1) * batch_size:]
326                 # Si no, es un batch normal
327                 else:
328                     st = step * batch_size
329                     nd = (step + 1) * batch_size
330 
331                     batch_inputs = TX[st:nd]
332                     batch_context = TY[st:nd]
333 
334                 feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
335                 loss_val,loss_batch,  p = session.run([loss_softplus,softplus_batch, prob],
                        feed_dict=feed_dict)
336 
337                 TEST_LOSS += np.sum(loss_batch)
338 
339                 # aClculo de TP/TN/FN/FP para cada batch
340 
341                 for i in range(len(p)):
342                     real = batch_context[i, 0]
343                     pred = p[i, 0]
344 
345                     if (real == 1) and (pred > 0.5): TRUE_POSITIVE += 1
346                     if (real == 1) and (pred <= 0.5): FALSE_NEGATIVE += 1
347                     if (real == 0) and (pred > 0.5): FALSE_POSITIVE += 1
348                     if (real == 0) and (pred <= 0.5): TRUE_NEGATIVE += 1
349 
350             PRECISION = TRUE_POSITIVE / (TRUE_POSITIVE + FALSE_POSITIVE * 1.0)
351             RECALL = TRUE_POSITIVE / (TRUE_POSITIVE + FALSE_NEGATIVE * 1.0)
352 
353             F1 = 2.0 * ((PRECISION * RECALL) / (PRECISION + RECALL))
354 
355             print("-" * 50)
356             print("LOSS:\t" + str(TEST_LOSS / (len(TX) * 1.0)))
357             print("PRECISION:\t" + str(PRECISION))
358             print("RECALL:\t" + str(RECALL))
359             print("F1:\t" + str(F1))
360             print("-" * 50)
361             print("")
362         #----------------------------------------------------------------------------------------------------
363 
364 ##########################################################################################################
365 # Llamadas
366 ##########################################################################################################
367 
368 TEST_NAME="test_081"
369 
370 os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
371 
372 #Se obtinenen cada uno de los embeddings aprendidos (incluidas las canciones)
373 D2V_DM_COMPLETE   = np.load('embeddings/d2v_dm/complete/EMB_MATRIX')
374 D2V_DBOW_COMPLETE = np.load('embeddings/d2v_dbow/complete/EMB_MATRIX')
```

```
375
376 #Se obtinenen cada uno de los embeddings aprendidos (incluidas las canciones)
377 D2V_DM_PRESENT  = np.load('embeddings/d2v_dm/present/EMB_MATRIX')
378 D2V_DBOW_PRESENT = np.load('embeddings/d2v_dbow/present/EMB_MATRIX')
379
380 W2V  = np.load('embeddings/w2v/EMB_MATRIX')
381
382 #Se concatenan los embeddings de los usuarios para generar el perfil
383 D2V_CONS = np.concatenate((D2V_DM_COMPLETE,D2V_DBOW_COMPLETE),axis=1) # El consolidado =
    Consolidado DM + Consolidado DBOW
384 D2V_PRES = np.concatenate((D2V_DM_PRESENT,D2V_DBOW_PRESENT),axis=1) # El presente = Presente
    DM + Presente DBOW
385
386 D2V_EMB = np.concatenate((D2V_CONS,D2V_PRES),axis=1) # El perfil completo CONS + PRES
387
388 #Se cargan los datos de train de las canciones para conocer los IDs reales
389 DATA = np.load("datos/Extend/ARRAY_ARRAYS_TRAIN.pkl")
390
391 #Obtener una lista con los ID de canciones que no aestn el la matriz de embeddings
392 ALL = set(list(range(W2V.shape[0])))
393 REAL = set(list(set(list(map(int, DATA)))))
394 NOT_EXISTS = list(ALL.difference(REAL))
395
396 ##########################################################################################################
397 # TENSORFLOW
398 ##########################################################################################################
399
400 #---------------------------------------------------------------------------------------------------------
401 # Grid Search + Early Stopping
402 #---------------------------------------------------------------------------------------------------------
403 '''
404 X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))
405 rates = [0.000001,1, 0.01, 0.0001]
406
407 for r in rates:
408     config = {"batch_size": 1024, "learning_rate": r, "epochs": 1000, "seed": 100}
409
410     print("Learning Rate: "+str(r))
411     print("-" * 50)
412     trainNet(X, Y, DX, DY, TX, TY, config, user_size, train=True, save=False)
413     print("-"*50)
414
415 exit()
416 '''
417 #---------------------------------------------------------------------------------------------------------
418 # Train
419 #---------------------------------------------------------------------------------------------------------
420 '''
421 X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))
422
423 #Para esta fase, se une TRAIN y DEV
424 X = np.concatenate((X,DX),axis=0)
425 Y = np.concatenate((Y,DY),axis=0)
426
427 DX=[]
428 DY=[]
429
```

```
config = {"batch_size": 1024, "learning_rate": 0.000001, "epochs": 435, "seed": 100}
trainNet(X, Y, DX, DY, TX, TY, config, user_size,train=True,save=True)

exit()
'''
#-----------------------------------------------------------------------------------------------------------------------
# Test
#-----------------------------------------------------------------------------------------------------------------------


X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))

config = {"batch_size": 1024, "learning_rate": 0.000001, "epochs": 435, "seed": 100}
trainNet(X, Y, DX, DY, TX, TY, config, user_size,train=False, save=True)
```

4.9. TEST 09

```
# -*- coding: utf-8 -*-

import os.path
import os
import tensorflow as tf
import numpy as np
import random
import time
import math
import pandas as pd
import pickle
from collections import Counter
from scipy.stats import linregress

#########################################################################################################
# eMtodos
#########################################################################################################

def getData(REAL_SONG_IDS=[],data ="datos/Append/ARRAY_ARRAYS_REC.pkl",shuffle=True,
    seed=100, train=.7):

    DATA = np.load(data)

    #DATA = DATA[:20];print("ELIMINAR FILTRO")

    #User, cancion 1, cancion 2
    TRAIN = np.empty([0, 3], dtype=np.int32)
    DEV = np.empty([0, 3], dtype=np.int32)
    TEST = np.empty([0, 3], dtype=np.int32)

    for u in range(len(DATA)):

        # Eliminar canciones escuchadas que no tienen embedding
        USER_PLAYS = DATA[u]
        USER_PLAYS = pd.DataFrame(list(map(int, USER_PLAYS)))
        USER_PLAYS = USER_PLAYS.loc[USER_PLAYS[0].isin(REAL_SONG_IDS)][0].values

        #Separar en train dev test
        TRAIN_ITEMS = int(len(USER_PLAYS)*train)
        DEV_ITEMS = int((len(USER_PLAYS)-TRAIN_ITEMS)/2)

        TRAIN_DATA = USER_PLAYS[:TRAIN_ITEMS]
        DEV_DATA = USER_PLAYS[TRAIN_ITEMS:TRAIN_ITEMS+DEV_ITEMS]
        TEST_DATA = USER_PLAYS[TRAIN_ITEMS+DEV_ITEMS:]

        # Crear ejemplos
        USR_TRAIN = np.empty([len(TRAIN_DATA)-1, 3], dtype=np.int32)
        USR_TRAIN[:,0] = u
        USR_TRAIN[:,1] = TRAIN_DATA[:len(TRAIN_DATA)-1]
        USR_TRAIN[:,2] = TRAIN_DATA[1:]

        USR_DEV = np.empty([len(DEV_DATA) - 1, 3], dtype=np.int32)
        USR_DEV[:, 0] = u
        USR_DEV[:, 1] = DEV_DATA[:len(DEV_DATA) - 1]
        USR_DEV[:, 2] = DEV_DATA[1:]
```

```
55
56          USR_TEST = np.empty([len(TEST_DATA) - 1, 3], dtype=np.int32)
57          USR_TEST[:, 0] = u
58          USR_TEST[:, 1] = TEST_DATA[:len(TEST_DATA) - 1]
59          USR_TEST[:, 2] = TEST_DATA[1:]
60
61
62          # nAadir al global
63          TRAIN = np.concatenate([TRAIN,USR_TRAIN])
64          DEV = np.concatenate([DEV,USR_DEV])
65          TEST = np.concatenate([TEST,USR_TEST])
66
67
68      #Mezclar
69      if(shuffle):
70          np.random.seed(seed)
71          np.random.shuffle(TRAIN)
72          np.random.shuffle(DEV)
73          np.random.shuffle(TEST)
74
75
76      return
            TRAIN[:,:2],np.matrix(TRAIN[:,-1]).T,DEV[:,:2],np.matrix(DEV[:,-1]).T,TEST[:,:2],np.matrix(TEST
77
78  def getBatchData(X, Y, batch_size, step, ITRS, RES_ITMS):
79
80      # Si se noaadi una oiteracin, introducir los datos restantes
81      if (step == ITRS - 1 and RES_ITMS > 0):
82          batch_inputs = X[(ITRS - 1) * batch_size:]
83          batch_context = Y[(ITRS - 1) * batch_size:]
84      # Si no, es un batch normal
85      else:
86          st = step * batch_size
87          nd = (step + 1) * batch_size
88
89          batch_inputs = X[st:nd]
90          batch_context = Y[st:nd]
91      return(batch_inputs,batch_context)
92
93  def trainNet(X,Y, DX,DY,TX,TY,config, user_size, train=True, save=True):
94
95      def getSlope(data):
96          r = linregress(range(len(data)), data)
97          return r.slope
98
99      batch_size = config['batch_size']
100     learning_rate = config['learning_rate']
101     epochs = config['epochs']
102
103     user_size = user_size
104     song_size = W2V.shape[0]
105
106     user_emb_size = 128
107     song_emb_size = 64
108     hidden_size = 32
109
110     profile_size = user_emb_size + song_emb_size
```

```
111
112     num_sampled = 64  # Ejemplos negativos (Para el NSE)
113
114     model_name = 'model'
115     model_path = 'models/' + TEST_NAME + '/model'
116     complete_path = model_path + "/" + model_name
117
118     # Se almacena la loss de dev de las 5 ultimas epochs
119     slope_size = 100
120     train_hist = []
121     dev_hist = []
122
123     # Creacin del grafo de TF.
124     graph = tf.Graph()
125
126     with graph.as_default():
127
128         if(config["seed"]!=1):
129             tf.set_random_seed(config["seed"])
130
131         # Nmero global de iteraciones
132         global_step = tf.Variable(0, dtype=tf.int32, trainable=False, name='global_step')
133
134         # Datos de entrada
            ------------------------------------------------------------------------------------
135         # Array del tamao del batch con las X
136         train_dataset = tf.placeholder(tf.int32, shape=[None, 2], name="train_inputs")
137
138         # Array del tamao del batch con las Y asociadas
139         train_labels = tf.placeholder(tf.float32, shape=[None, 1], name="train_labels")
140
141         # Embeddings
            ------------------------------------------------------------------------------------
142
143         O1 = tf.Variable(tf.truncated_normal([user_size, user_emb_size], mean=0.0, stddev=1.0
                / math.sqrt(user_emb_size)), name="O1")
144         BO1 = tf.Variable(tf.zeros([user_emb_size]), name="BO1")
145
146         W2V_EMB_VRB = tf.Variable(W2V, trainable=False, name="word_embeddings")
147
148         # Operaciones
            ------------------------------------------------------------------------------------
149
150         # Embedding de documento
151         ed = tf.nn.embedding_lookup(O1, train_dataset[:, 0])
152         ed = ed + BO1
153
154         # Embedding de cancion
155         ec = tf.nn.embedding_lookup(W2V_EMB_VRB, train_dataset[:, 1])
156
157         # Clculo de LOSS y optimizador
            ------------------------------------------------------------------------------------
158
159         embed = []
160         embed.append(ed)
161         embed.append(ec)
162         embed = tf.concat(embed, 1)
```

```
163
164         # softmax weights, W and D vectors should be concatenated before applying softmax
165         weights = tf.Variable(tf.truncated_normal([song_size, profile_size], stddev=1.0 /
                math.sqrt(song_size)),
166                               name="weights")
167         # softmax biases
168         biases = tf.Variable(tf.zeros([song_size]), name="biases")
169
170         # aClculo de LOSS y optimizador—
                ------------------------------------------------------------------------------
171         nce_loss = tf.nn.nce_loss(weights=weights,
172                                   biases=biases,
173                                   labels=train_labels,
174                                   inputs=embed,
175                                   num_sampled=num_sampled,
176                                   num_classes=song_size,
177                                   name="nce_loss")
178
179         batch_loss = nce_loss
180         nce_loss = tf.reduce_mean(nce_loss)
181
182         logits = tf.matmul(embed, tf.transpose(weights))
183         logits = tf.nn.bias_add(logits, biases)
184         # logits = tf.nn.softmax(logits,axis=0)
185         top_v,top_i = tf.nn.top_k(logits,k=song_size)
186
187         # optimizer =
                tf.train.AdamOptimizer(learning_rate).minimize(nce_loss,global_step=global_step)
188         train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(nce_loss,
                global_step=global_step)
189
190         # Inicializar variables
191         init = tf.global_variables_initializer()
192
193         # Crear objeto encargado de almacenar la red
194         saver = tf.train.Saver(max_to_keep=1)
195
196     #config = tf.ConfigProto()
197     #config.gpu_options.allow_growth = True
198     #with tf.Session(graph=graph, config=config) as session:
199     with tf.Session(graph=graph) as session:
200
201         # We must initialize all variables before we use them.
202         init.run()
203
204
205         if(save):
206             # Obtener el checkpoint
207             ckpt = tf.train.get_checkpoint_state(model_path)
208
209             # Si existe el checkpoint restaurar
210             if ckpt and ckpt.model_checkpoint_path:
211                 saver.restore(session, ckpt.model_checkpoint_path)
212
213         #Si se desea entrenar
214         if(train):
215             #------------------------------------------------------------------------------
```

```
216             # TRAIN
217             #--------------------------------------------------------------------------------
218
219             ITRS= int(len(X)/batch_size)
220             RES_ITMS = len(X) - (ITRS*batch_size)
221             if(RES_ITMS>0):ITRS+=1 #Si no es divisin exacta, se aade una iteracin
222
223             for e in range(epochs):
224                 TRAIN_LOSS = 0.0
225
226                 print("Epoch "+str(e))
227
228                 for step in range(ITRS):
229
230                     batch_inputs, batch_context =
                            getBatchData(X,Y,batch_size,step,ITRS,RES_ITMS)
231
232                     feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
233
234                     _, loss_val, batch_lss, gs = session.run([train_step,
                            nce_loss,batch_loss,global_step], feed_dict=feed_dict)
235
236                     TRAIN_LOSS += np.sum(batch_lss)
237
238                 #print('Epoch '+str(e)+', last batch loss: '+str(loss_val))
239
240                 if (save): saver.save(session, complete_path, global_step=global_step)
241
242                 #--------------------------------------------------------------------------------
243                 # DEV
244                 #--------------------------------------------------------------------------------
245
246                 if(len(DX)>0):
247
248                     ITRS_TEST = int(len(DX) / batch_size)
249                     RES_ITMS_TEST = len(DX) - (ITRS_TEST * batch_size)
250                     TEST_LOSS = 0.0
251                     if (RES_ITMS_TEST > 0): ITRS_TEST += 1 # Si no es divisin exacta, se
                            aade una iteracin
252
253
254                     for step in range(ITRS_TEST):
255
256                         batch_inputs, batch_context =
                                getBatchData(DX,DY,batch_size,step,ITRS_TEST,RES_ITMS_TEST)
257
258                         feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
259                         loss_val,batch_lss = session.run([nce_loss,batch_loss],
                                feed_dict=feed_dict)
260
261                         TEST_LOSS+=np.sum(batch_lss)
262
263                     dev_hist.append(TEST_LOSS / (len(DX) * 1.0))
264                     train_hist.append(TRAIN_LOSS / (len(X) * 1.0))
265
266                 if (len(dev_hist) >= slope_size):
267                     # Calcular la pendiente
```

```
268                 print(getSlope(dev_hist[-slope_size:]))
269                 print(dev_hist[-slope_size:])
270 
271                 if (getSlope(dev_hist[-slope_size:]) > -1e-5 or e==epochs-1):
272 
273                     def printArray(data,data2):
274                         for i in range(len(data)):
275                             a=data[i]
276                             b=data2[i]
277                             print(str(a).replace(".",",")+"\t"+str(b).replace(".",","))
278 
279                     print("-" * 50)
280                     print(e)
281                     printArray(dev_hist,train_hist)
282                     return
283         else:
284             #
                ------------------------------------------------------------------------------
285             # TEST FINAL
286             #
                ------------------------------------------------------------------------------
287             batch_size = 300
288 
289             IN_TOP_5 = 0
290             IN_TOP_10 = 0
291             IN_TOP_20 = 0
292             IN_TOP_50 = 0
293             IN_TOP_100 = 0
294             IN_TOP_1000 = 0
295             real_positions = []
296 
297             items = 0
298 
299             ITRS_TEST = int(len(TX) / batch_size)
300             RES_ITMS_TEST = len(TX) - (ITRS_TEST * batch_size)
301             if (RES_ITMS_TEST > 0): ITRS_TEST += 1  # Si no es @divisin exacta, se ñaade una
                    @iteracin
302 
303             for step in range(ITRS_TEST):
304                 # tss = time.time()
305 
306                 batch_inputs, batch_context = getBatchData(TX, TY, batch_size, step,
                        ITRS_TEST, RES_ITMS_TEST)
307                 top_values, top_indx = session.run([top_v, top_i],
308                                                    feed_dict={train_dataset: batch_inputs,
                                                        train_labels: batch_context})
309 
310                 for j in range(len(batch_context)):
311                     real_position = int(np.where(top_indx[j, :] == batch_context[j, 0])[0])
312                     real_positions.append(real_position)
313 
314                     IN_TOP_5 += 1 if real_position <= 5 else 0
315                     IN_TOP_10 += 1 if real_position <= 10 else 0
316                     IN_TOP_20 += 1 if real_position <= 20 else 0
317                     IN_TOP_50 += 1 if real_position <= 50 else 0
318                     IN_TOP_100 += 1 if real_position <= 100 else 0
319                     IN_TOP_1000 += 1 if real_position <= 1000 else 0
```

```
320
321                 items += len(batch_inputs) * 1.0
322
323                 PTF = IN_TOP_5 / items
324                 PTT = IN_TOP_10 / items
325                 PTTW = IN_TOP_20 / items
326                 PTFT = IN_TOP_50 / items
327                 PTCIEN = IN_TOP_100 / items
328                 PTMIL = IN_TOP_1000 / items
329
330                 print(np.mean(real_positions), np.std(real_positions),
                        np.median(real_positions))
331                 print(str(PTF).replace(".", ",") + "\t" + str(PTT).replace(".", ",") + "\t"
332                       + str(PTTW).replace(".", ",") + "\t" + str(PTFT).replace(".", ",") +
                            "\t"
333                       + str(PTCIEN).replace(".", ",") + "\t" + str(PTMIL).replace(".", ","))
334
335                 # print(time.time()-tss)
336
337             print(np.mean(real_positions), np.std(real_positions), np.median(real_positions))
338
339         #-------------------------------------------------------------------------------------------------------------------------
340
341 ##########################################################################################################################
342 # Llamadas
343 ##########################################################################################################################
344
345 TEST_NAME="test_022"
346
347 os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
348
349 #Solamente se carga para conocer el unmero de canciones
350 W2V  = np.load('embeddings/w2v/EMB_MATRIX')
351
352 #Se cargan los datos de train de las canciones para conocer los IDs reales
353 DATA = np.load("datos/Extend/ARRAY_ARRAYS_TRAIN.pkl")
354
355 #Obtener una lista con los ID de canciones que no aestn el la matriz de embeddings
356 ALL = set(list(range(W2V.shape[0])))
357 REAL = set(list(set(list(map(int, DATA)))))
358
359 ##########################################################################################################################
360 # TENSORFLOW
361 ##########################################################################################################################
362 '''
363 #-------------------------------------------------------------------------------------------------------------------------
364 # Grid Search + Early Stopping
365 #-------------------------------------------------------------------------------------------------------------------------
366
367 X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))
368 rates = [0.000001,1, 0.01, 0.0001]
369
370 for r in rates:
371     config = {"batch_size": 1024, "learning_rate": r, "epochs": 1000, "seed": 100}
372
373     print("Learning Rate: "+str(r))
374     print("-" * 50)
```

```
375     trainNet(X, Y, DX, DY, TX, TY, config, user_size, train=True, save=False)
376     print("-"*50)
377 
378 exit()
379 '''
380 #-----------------------------------------------------------------------------------------------------------------------
381 # Train
382 #-----------------------------------------------------------------------------------------------------------------------
383 '''
384 X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))
385 
386 #Para esta fase, se une TRAIN y DEV
387 X = np.concatenate((X,DX),axis=0)
388 Y = np.concatenate((Y,DY),axis=0)
389 
390 DX=[]
391 DY=[]
392 
393 config = {"batch_size": 1024, "learning_rate": 1, "epochs":172, "seed":100}
394 trainNet(X, Y, DX, DY, TX, TY, config, user_size, train=True, save=True)
395 
396 exit()
397 '''
398 #-----------------------------------------------------------------------------------------------------------------------
399 # Test
400 #-----------------------------------------------------------------------------------------------------------------------
401 
402 X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))
403 
404 config = {"batch_size": 1024, "learning_rate": 1, "epochs":172, "seed":100}
405 trainNet(X, Y, DX, DY, TX, TY, config, user_size,train=False, save=True)
```

4.10. TEST 10

```
# —*— coding: utf—8 —*—

import os.path
import os
import tensorflow as tf
import numpy as np
import random
import time
import math
import pandas as pd
import pickle
from collections import Counter
from scipy.stats import linregress

#########################################################################################################################
# eMtodos
#########################################################################################################################

def getData(REAL_SONG_IDS=[],data ="datos/Append/ARRAY_ARRAYS_REC.pkl",shuffle=True,
    seed=100, train=.7):

    DATA = np.load(data)

    #DATA = DATA[:20];print("ELIMINAR FILTRO")

    #User, cancion 1, cancion 2
    TRAIN = np.empty([0, 3], dtype=np.int32)
    DEV = np.empty([0, 3], dtype=np.int32)
    TEST = np.empty([0, 3], dtype=np.int32)

    for u in range(len(DATA)):

        # Eliminar canciones escuchadas que no tienen embedding
        USER_PLAYS = DATA[u]
        USER_PLAYS = pd.DataFrame(list(map(int, USER_PLAYS)))
        USER_PLAYS = USER_PLAYS.loc[USER_PLAYS[0].isin(REAL_SONG_IDS)][0].values

        #Separar en train dev test
        TRAIN_ITEMS = int(len(USER_PLAYS)*train)
        DEV_ITEMS = int((len(USER_PLAYS)—TRAIN_ITEMS)/2)

        TRAIN_DATA = USER_PLAYS[:TRAIN_ITEMS]
        DEV_DATA = USER_PLAYS[TRAIN_ITEMS:TRAIN_ITEMS+DEV_ITEMS]
        TEST_DATA = USER_PLAYS[TRAIN_ITEMS+DEV_ITEMS:]

        # Crear ejemplos
        USR_TRAIN = np.empty([len(TRAIN_DATA)—1, 3], dtype=np.int32)
        USR_TRAIN[:,0] = u
        USR_TRAIN[:,1] = TRAIN_DATA[:len(TRAIN_DATA)—1]
        USR_TRAIN[:,2] = TRAIN_DATA[1:]

        USR_DEV = np.empty([len(DEV_DATA) — 1, 3], dtype=np.int32)
        USR_DEV[:, 0] = u
        USR_DEV[:, 1] = DEV_DATA[:len(DEV_DATA) — 1]
        USR_DEV[:, 2] = DEV_DATA[1:]
```

```
55
56          USR_TEST = np.empty([len(TEST_DATA) - 1, 3], dtype=np.int32)
57          USR_TEST[:, 0] = u
58          USR_TEST[:, 1] = TEST_DATA[:len(TEST_DATA) - 1]
59          USR_TEST[:, 2] = TEST_DATA[1:]
60
61
62          # Aadir al global
63          TRAIN = np.concatenate([TRAIN,USR_TRAIN])
64          DEV = np.concatenate([DEV,USR_DEV])
65          TEST = np.concatenate([TEST,USR_TEST])
66
67
68      #Mezclar
69      if(shuffle):
70          np.random.seed(seed)
71          np.random.shuffle(TRAIN)
72          np.random.shuffle(DEV)
73          np.random.shuffle(TEST)
74
75
76      return
            TRAIN[:,:2],np.matrix(TRAIN[:,-1]).T,DEV[:,:2],np.matrix(DEV[:,-1]).T,TEST[:,:2],np.matrix(TEST
77
78  def getBatchData(X, Y, batch_size, step, ITRS, RES_ITMS):
79
80      # Si se aadi una iteracin, introducir los datos restantes
81      if (step == ITRS - 1 and RES_ITMS > 0):
82          batch_inputs = X[(ITRS - 1) * batch_size:]
83          batch_context = Y[(ITRS - 1) * batch_size:]
84      # Si no, es un batch normal
85      else:
86          st = step * batch_size
87          nd = (step + 1) * batch_size
88
89          batch_inputs = X[st:nd]
90          batch_context = Y[st:nd]
91      return(batch_inputs,batch_context)
92
93  def trainNet(X,Y, DX,DY,TX,TY,config, user_size, train=True, save=True):
94
95      def getSlope(data):
96          r = linregress(range(len(data)), data)
97          return r.slope
98
99      batch_size = config['batch_size']
100     learning_rate = config['learning_rate']
101     epochs = config['epochs']
102
103     user_size = user_size
104     song_size = W2V.shape[0]
105
106     user_emb_size = 128
107     song_emb_size = 64
108     hidden_size = 32
109
110     profile_size = user_emb_size + song_emb_size
```

```
111
112     num_sampled = 64  # Ejemplos negativos (Para el NSE)
113
114     model_name = 'model'
115     model_path = 'models/' + TEST_NAME + '/model'
116     complete_path = model_path + "/" + model_name
117
118     # Se almacena la loss de dev de las 5 ultimas epochs
119     slope_size = 100
120     train_hist = []
121     dev_hist = []
122
123     # Creacin del grafo de TF.
124     graph = tf.Graph()
125
126     with graph.as_default():
127
128         if(config["seed"]!=1):
129             tf.set_random_seed(config["seed"])
130
131         # Nmero global de iteraciones
132         global_step = tf.Variable(0, dtype=tf.int32, trainable=False, name='global_step')
133
134         # Datos de entrada
            ------------------------------------------------------------------------------------------
135         # Array del tamao del batch con las X
136         train_dataset = tf.placeholder(tf.int32, shape=[None, 2], name="train_inputs")
137
138         # Array del tamao del batch con las Y asociadas
139         train_labels = tf.placeholder(tf.float32, shape=[None, 1], name="train_labels")
140
141         # Embeddings
            ------------------------------------------------------------------------------------------
142
143         D2V_EMB_VRB = tf.Variable(D2V_EMB, trainable=False, name="doc_embeddings")
144
145         W2V_EMB_VRB = tf.Variable(W2V, trainable=False, name="word_embeddings")
146
147         # Operaciones
            ------------------------------------------------------------------------------------------
148
149         # Embedding de documento
150         ed = tf.nn.embedding_lookup(D2V_EMB_VRB, train_dataset[:, 0])
151
152         # Embedding de cancion
153         ec = tf.nn.embedding_lookup(W2V_EMB_VRB, train_dataset[:, 1])
154
155         # Clculo de LOSS y optimizador
            ------------------------------------------------------------------------------------------
156
157         embed = []
158         embed.append(ed)
159         embed.append(ec)
160         embed = tf.concat(embed, 1)
161
162         # softmax weights, W and D vectors should be concatenated before applying softmax
163         weights = tf.Variable(tf.truncated_normal([song_size, profile_size], stddev=1.0 /
```

```
            math.sqrt(song_size)),
                                name="weights")
        # softmax biases
        biases = tf.Variable(tf.zeros([song_size]), name="biases")

        # aClculo de LOSS y optimizador—
        ------------------------------------------------------------------------------------------
        nce_loss = tf.nn.nce_loss(weights=weights,
                                  biases=biases,
                                  labels=train_labels,
                                  inputs=embed,
                                  num_sampled=num_sampled,
                                  num_classes=song_size,
                                  name="nce_loss")

        batch_loss = nce_loss
        nce_loss = tf.reduce_mean(nce_loss)

        logits = tf.matmul(embed, tf.transpose(weights))
        logits = tf.nn.bias_add(logits, biases)
        # logits = tf.nn.softmax(logits,axis=0)
        top_v,top_i = tf.nn.top_k(logits,k=song_size)

        # optimizer =
            tf.train.AdamOptimizer(learning_rate).minimize(nce_loss,global_step=global_step)
        train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(nce_loss,
            global_step=global_step)

        # Inicializar variables
        init = tf.global_variables_initializer()

        # Crear objeto encargado de almacenar la red
        saver = tf.train.Saver(max_to_keep=1)

    #config = tf.ConfigProto()
    #config.gpu_options.allow_growth = True
    #with tf.Session(graph=graph, config=config) as session:
    with tf.Session(graph=graph) as session:

        # We must initialize all variables before we use them.
        init.run()


        if(save):
            # Obtener el checkpoint
            ckpt = tf.train.get_checkpoint_state(model_path)

            # Si existe el checkpoint restaurar
            if ckpt and ckpt.model_checkpoint_path:
                saver.restore(session, ckpt.model_checkpoint_path)

        #Si se desea entrenar
        if(train):
            #------------------------------------------------------------------------------------
            # TRAIN
            #------------------------------------------------------------------------------------

```

```
217             ITRS= int(len(X)/batch_size)
218             RES_ITMS = len(X) - (ITRS*batch_size)
219             if(RES_ITMS>0):ITRS+=1 #Si no es divisin exacta, se aade una iteracin
220
221             for e in range(epochs):
222                 TRAIN_LOSS = 0.0
223
224                 print("Epoch "+str(e))
225
226                 for step in range(ITRS):
227
228                     batch_inputs, batch_context =
                            getBatchData(X,Y,batch_size,step,ITRS,RES_ITMS)
229
230                     feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
231
232                     _, loss_val, batch_lss, gs = session.run([train_step,
                            nce_loss,batch_loss,global_step], feed_dict=feed_dict)
233
234                     TRAIN_LOSS += np.sum(batch_lss)
235
236                 #print('Epoch '+str(e)+', last batch loss: '+str(loss_val))
237
238                 if (save): saver.save(session, complete_path, global_step=global_step)
239
240                 #-----------------------------------------------------------------------------
241                 # DEV
242                 #-----------------------------------------------------------------------------
243
244                 if(len(DX)>0):
245
246                     ITRS_TEST = int(len(DX) / batch_size)
247                     RES_ITMS_TEST = len(DX) - (ITRS_TEST * batch_size)
248                     TEST_LOSS = 0.0
249                     if (RES_ITMS_TEST > 0): ITRS_TEST += 1 # Si no es divisin exacta, se
                            aade una iteracin
250
251
252                     for step in range(ITRS_TEST):
253
254                         batch_inputs, batch_context =
                                getBatchData(DX,DY,batch_size,step,ITRS_TEST,RES_ITMS_TEST)
255
256                         feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
257                         loss_val,batch_lss = session.run([nce_loss,batch_loss],
                                feed_dict=feed_dict)
258
259                         TEST_LOSS+=np.sum(batch_lss)
260
261                     dev_hist.append(TEST_LOSS / (len(DX) * 1.0))
262                     train_hist.append(TRAIN_LOSS / (len(X) *1.0))
263
264                 if (len(dev_hist) >= slope_size):
265                     # Calcular la pendiente
266                     print(getSlope(dev_hist[-slope_size:]))
267                     print(dev_hist[-slope_size:])
268
```

```
269                 if (getSlope(dev_hist[-slope_size:]) > -1e-5 or e==epochs-1):
270
271                     def printArray(data,data2):
272                         for i in range(len(data)):
273                             a=data[i]
274                             b=data2[i]
275                             print(str(a).replace(".",",")+"\t"+str(b).replace(".",","))
276
277                     print("-" * 50)
278                     print(e)
279                     printArray(dev_hist,train_hist)
280                     return
281         else:
282             #
                ------------------------------------------------------------------------------
283             # TEST FINAL
284             #
                ------------------------------------------------------------------------------
285             batch_size = 300
286
287             IN_TOP_5 = 0
288             IN_TOP_10 = 0
289             IN_TOP_20 = 0
290             IN_TOP_50 = 0
291             IN_TOP_100 = 0
292             IN_TOP_1000 = 0
293             real_positions = []
294
295             items = 0
296
297             ITRS_TEST = int(len(TX) / batch_size)
298             RES_ITMS_TEST = len(TX) - (ITRS_TEST * batch_size)
299             if (RES_ITMS_TEST > 0): ITRS_TEST += 1  # Si no es divisin exacta, se aade una
                    iteracin
300
301             for step in range(ITRS_TEST):
302                 #tss = time.time()
303
304                 batch_inputs, batch_context = getBatchData(TX, TY, batch_size, step,
                        ITRS_TEST, RES_ITMS_TEST)
305                 top_values,top_indx = session.run([top_v,top_i], feed_dict={train_dataset:
                        batch_inputs, train_labels: batch_context})
306
307                 for j in range(len(batch_context)):
308                     real_position = int(np.where(top_indx[j, :] == batch_context[j, 0])[0])
309                     real_positions.append(real_position)
310
311                     IN_TOP_5 += 1 if real_position <= 5 else 0
312                     IN_TOP_10 += 1 if real_position <= 10 else 0
313                     IN_TOP_20 += 1 if real_position <= 20 else 0
314                     IN_TOP_50 += 1 if real_position <= 50 else 0
315                     IN_TOP_100 += 1 if real_position <= 100 else 0
316                     IN_TOP_1000 += 1 if real_position <= 1000 else 0
317
318                 items += len(batch_inputs)*1.0
319
320                 PTF = IN_TOP_5 / items
```

```
                PTT = IN_TOP_10 / items
                PTTW = IN_TOP_20 / items
                PTFT = IN_TOP_50 / items
                PTCIEN = IN_TOP_100 / items
                PTMIL = IN_TOP_1000 / items

                print(np.mean(real_positions), np.std(real_positions),
                    np.median(real_positions))
                print(str(PTF).replace(".", ",") + "\t" + str(PTT).replace(".", ",") + "\t"
                      + str(PTTW).replace(".",",") + "\t" + str(PTFT).replace(".", ",") + "\t"
                      + str(PTCIEN).replace(".", ",") + "\t" + str(PTMIL).replace(".",","))

                #print(time.time()-tss)

            print(np.mean(real_positions), np.std(real_positions), np.median(real_positions))

        #-------------------------------------------------------------------------------------------------------------

##########################################################################################################################
# Llamadas
##########################################################################################################################

TEST_NAME="test_032"

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'

#Se obtinenen cada uno de los embeddings aprendidos (incluidas las canciones)
D2V_DM_COMPLETE   = np.load('embeddings/d2v_dm/complete/EMB_MATRIX')
D2V_DBOW_COMPLETE = np.load('embeddings/d2v_dbow/complete/EMB_MATRIX')
W2V  = np.load('embeddings/w2v/EMB_MATRIX')

#Se concatenan los embeddings de los usuarios para generar el perfil
D2V_EMB = np.concatenate((D2V_DM_COMPLETE,D2V_DBOW_COMPLETE),axis=1) # El consolidado =
    Consolidado DM + Consolidado DBOW

#Se cargan los datos de train de las canciones para conocer los IDs reales
DATA = np.load("datos/Extend/ARRAY_ARRAYS_TRAIN.pkl")

#Obtener una lista con los ID de canciones que no aestn el la matriz de embeddings
ALL = set(list(range(W2V.shape[0])))
REAL = set(list(set(list(map(int, DATA)))))
NOT_EXISTS = list(ALL.difference(REAL))

##########################################################################################################################
# TENSORFLOW
##########################################################################################################################
'''
#-------------------------------------------------------------------------------------------------------------
# Grid Search + Early Stopping
#-------------------------------------------------------------------------------------------------------------

X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))
rates = [0.000001,1, 0.01, 0.0001]

for r in rates:
    config = {"batch_size": 1024, "learning_rate": r, "epochs": 1000, "seed": 100}

```

```
376     print("Learning Rate: "+str(r))
377     print("-" * 50)
378     trainNet(X, Y, DX, DY, TX, TY, config, user_size, train=True, save=False)
379     print("-"*50)
380 
381 exit()
382 '''
383 #------------------------------------------------------------------------------
384 # Train
385 #------------------------------------------------------------------------------
386 '''
387 X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))
388 
389 #Para esta fase, se une TRAIN y DEV
390 X = np.concatenate((X,DX),axis=0)
391 Y = np.concatenate((Y,DY),axis=0)
392 
393 DX=[]
394 DY=[]
395 
396 config = {"batch_size": 1024, "learning_rate": 0.01, "epochs":1000, "seed":100}
397 trainNet(X, Y, DX, DY, TX, TY, config, user_size, train=True, save=True)
398 
399 exit()
400 '''
401 #------------------------------------------------------------------------------
402 # Test
403 #------------------------------------------------------------------------------
404 
405 X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))
406 
407 config = {"batch_size": 1024, "learning_rate": 0.01, "epochs":1000, "seed":100}
408 trainNet(X, Y, DX, DY, TX, TY, config, user_size,train=False, save=True)
```

4.11. TEST 11

```
# -*- coding: utf-8 -*-

import os.path
import os
import tensorflow as tf
import numpy as np
import random
import time
import math
import pandas as pd
import pickle
from collections import Counter
from scipy.stats import linregress

###############################################################################################################
# eMtodos
###############################################################################################################

def getData(REAL_SONG_IDS=[],data ="datos/Append/ARRAY_ARRAYS_REC.pkl",shuffle=True,
    seed=100, train=.7):

    DATA = np.load(data)

    #DATA = DATA[:20];print("ELIMINAR FILTRO")

    #User, cancion 1, cancion 2
    TRAIN = np.empty([0, 3], dtype=np.int32)
    DEV = np.empty([0, 3], dtype=np.int32)
    TEST = np.empty([0, 3], dtype=np.int32)

    for u in range(len(DATA)):

        # Eliminar canciones escuchadas que no tienen embedding
        USER_PLAYS = DATA[u]
        USER_PLAYS = pd.DataFrame(list(map(int, USER_PLAYS)))
        USER_PLAYS = USER_PLAYS.loc[USER_PLAYS[0].isin(REAL_SONG_IDS)][0].values

        #Separar en train dev test
        TRAIN_ITEMS = int(len(USER_PLAYS)*train)
        DEV_ITEMS = int((len(USER_PLAYS)-TRAIN_ITEMS)/2)

        TRAIN_DATA = USER_PLAYS[:TRAIN_ITEMS]
        DEV_DATA = USER_PLAYS[TRAIN_ITEMS:TRAIN_ITEMS+DEV_ITEMS]
        TEST_DATA = USER_PLAYS[TRAIN_ITEMS+DEV_ITEMS:]

        # Crear ejemplos
        USR_TRAIN = np.empty([len(TRAIN_DATA)-1, 3], dtype=np.int32)
        USR_TRAIN[:,0] = u
        USR_TRAIN[:,1] = TRAIN_DATA[:len(TRAIN_DATA)-1]
        USR_TRAIN[:,2] = TRAIN_DATA[1:]

        USR_DEV = np.empty([len(DEV_DATA) - 1, 3], dtype=np.int32)
        USR_DEV[:, 0] = u
        USR_DEV[:, 1] = DEV_DATA[:len(DEV_DATA) - 1]
        USR_DEV[:, 2] = DEV_DATA[1:]
```

```
55
56          USR_TEST = np.empty([len(TEST_DATA) - 1, 3], dtype=np.int32)
57          USR_TEST[:, 0] = u
58          USR_TEST[:, 1] = TEST_DATA[:len(TEST_DATA) - 1]
59          USR_TEST[:, 2] = TEST_DATA[1:]
60
61
62          # Aadir al global
63          TRAIN = np.concatenate([TRAIN,USR_TRAIN])
64          DEV = np.concatenate([DEV,USR_DEV])
65          TEST = np.concatenate([TEST,USR_TEST])
66
67
68      #Mezclar
69      if(shuffle):
70          np.random.seed(seed)
71          np.random.shuffle(TRAIN)
72          np.random.shuffle(DEV)
73          np.random.shuffle(TEST)
74
75
76      return
            TRAIN[:,:2],np.matrix(TRAIN[:,-1]).T,DEV[:,:2],np.matrix(DEV[:,-1]).T,TEST[:,:2],np.matrix(TEST
77
78  def getBatchData(X, Y, batch_size, step, ITRS, RES_ITMS):
79
80      # Si se aadi una iteracin, introducir los datos restantes
81      if (step == ITRS - 1 and RES_ITMS > 0):
82          batch_inputs = X[(ITRS - 1) * batch_size:]
83          batch_context = Y[(ITRS - 1) * batch_size:]
84      # Si no, es un batch normal
85      else:
86          st = step * batch_size
87          nd = (step + 1) * batch_size
88
89          batch_inputs = X[st:nd]
90          batch_context = Y[st:nd]
91      return(batch_inputs,batch_context)
92
93  def trainNet(X,Y, DX,DY,TX,TY,config, user_size, train=True, save=True):
94
95      def getSlope(data):
96          r = linregress(range(len(data)), data)
97          return r.slope
98
99      batch_size = config['batch_size']
100     learning_rate = config['learning_rate']
101     epochs = config['epochs']
102
103     user_size = user_size
104     song_size = W2V.shape[0]
105
106     user_emb_size = 128
107     song_emb_size = 64
108     hidden_size = 32
109
110     profile_size = user_emb_size + song_emb_size
```

```
111
112     num_sampled = 64  # Ejemplos negativos (Para el NSE)
113
114     model_name = 'model'
115     model_path = 'models/' + TEST_NAME + '/model'
116     complete_path = model_path + "/" + model_name
117
118     # Se almacena la loss de dev de las 5 ultimas epochs
119     slope_size = 100
120     train_hist = []
121     dev_hist = []
122
123     # Creacin del grafo de TF.
124     graph = tf.Graph()
125
126     with graph.as_default():
127
128         if(config["seed"]!=1):
129             tf.set_random_seed(config["seed"])
130
131         # Nmero global de iteraciones
132         global_step = tf.Variable(0, dtype=tf.int32, trainable=False, name='global_step')
133
134         # Datos de entrada
            ------------------------------------------------------------------------------------------
135         # Array del tamao del batch con las X
136         train_dataset = tf.placeholder(tf.int32, shape=[None, 2], name="train_inputs")
137
138         # Array del tamao del batch con las Y asociadas
139         train_labels = tf.placeholder(tf.float32, shape=[None, 1], name="train_labels")
140
141         # Embeddings
            ------------------------------------------------------------------------------------------
142
143         D2V_EMB_VRB = tf.Variable(D2V_EMB, trainable=False, name="doc_embeddings")
144
145         W2V_EMB_VRB = tf.Variable(W2V, trainable=False, name="word_embeddings")
146
147         # Operaciones
            ------------------------------------------------------------------------------------------
148
149         # Embedding de documento
150         ed = tf.nn.embedding_lookup(D2V_EMB_VRB, train_dataset[:, 0])
151
152         # Embedding de cancion
153         ec = tf.nn.embedding_lookup(W2V_EMB_VRB, train_dataset[:, 1])
154
155         # Clculo de LOSS y optimizador
            ------------------------------------------------------------------------------------------
156
157         embed = []
158         embed.append(ed)
159         embed.append(ec)
160         embed = tf.concat(embed, 1)
161
162         # softmax weights, W and D vectors should be concatenated before applying softmax
163         weights = tf.Variable(tf.truncated_normal([song_size, profile_size], stddev=1.0 /
```

```
            math.sqrt(song_size)),
                                name="weights")
        # softmax biases
        biases = tf.Variable(tf.zeros([song_size]), name="biases")

        # aClculo de LOSS y optimizador-
        ----------------------------------------------------------------------------------------------------
        nce_loss = tf.nn.nce_loss(weights=weights,
                                  biases=biases,
                                  labels=train_labels,
                                  inputs=embed,
                                  num_sampled=num_sampled,
                                  num_classes=song_size,
                                  name="nce_loss")

        batch_loss = nce_loss
        nce_loss = tf.reduce_mean(nce_loss)

        logits = tf.matmul(embed, tf.transpose(weights))
        logits = tf.nn.bias_add(logits, biases)
        # logits = tf.nn.softmax(logits,axis=0)
        top_v,top_i = tf.nn.top_k(logits,k=song_size)

        # optimizer =
            tf.train.AdamOptimizer(learning_rate).minimize(nce_loss,global_step=global_step)
        train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(nce_loss,
            global_step=global_step)

        # Inicializar variables
        init = tf.global_variables_initializer()

        # Crear objeto encargado de almacenar la red
        saver = tf.train.Saver(max_to_keep=1)

    #config = tf.ConfigProto()
    #config.gpu_options.allow_growth = True
    #with tf.Session(graph=graph, config=config) as session:
    with tf.Session(graph=graph) as session:

        # We must initialize all variables before we use them.
        init.run()


        if(save):
            # Obtener el checkpoint
            ckpt = tf.train.get_checkpoint_state(model_path)

            # Si existe el checkpoint restaurar
            if ckpt and ckpt.model_checkpoint_path:
                saver.restore(session, ckpt.model_checkpoint_path)

        #Si se desea entrenar
        if(train):
            #----------------------------------------------------------------------------------------------
            # TRAIN
            #----------------------------------------------------------------------------------------------

```

```
217             ITRS= int(len(X)/batch_size)
218             RES_ITMS = len(X) - (ITRS*batch_size)
219             if(RES_ITMS>0):ITRS+=1 #Si no es @divisin exacta, se ñaade una @iteracin
220
221             for e in range(epochs):
222                 TRAIN_LOSS = 0.0
223
224                 print("Epoch "+str(e))
225
226                 for step in range(ITRS):
227
228                     batch_inputs, batch_context =
                            getBatchData(X,Y,batch_size,step,ITRS,RES_ITMS)
229
230                     feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
231
232                     _, loss_val, batch_lss, gs = session.run([train_step,
                            nce_loss,batch_loss,global_step], feed_dict=feed_dict)
233
234                     TRAIN_LOSS += np.sum(batch_lss)
235
236                 #print('Epoch '+str(e)+', last batch loss: '+str(loss_val))
237
238                 if (save): saver.save(session, complete_path, global_step=global_step)
239
240                 #-----------------------------------------------------------------------------
241                 # DEV
242                 #-----------------------------------------------------------------------------
243
244                 if(len(DX)>0):
245
246                     ITRS_TEST = int(len(DX) / batch_size)
247                     RES_ITMS_TEST = len(DX) - (ITRS_TEST * batch_size)
248                     TEST_LOSS = 0.0
249                     if (RES_ITMS_TEST > 0): ITRS_TEST += 1 # Si no es @divisin exacta, se
                            ñaade una @iteracin
250
251
252                     for step in range(ITRS_TEST):
253
254                         batch_inputs, batch_context =
                                getBatchData(DX,DY,batch_size,step,ITRS_TEST,RES_ITMS_TEST)
255
256                         feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
257                         loss_val,batch_lss = session.run([nce_loss,batch_loss],
                                feed_dict=feed_dict)
258
259                         TEST_LOSS+=np.sum(batch_lss)
260
261                     dev_hist.append(TEST_LOSS / (len(DX) * 1.0))
262                     train_hist.append(TRAIN_LOSS / (len(X) * 1.0))
263
264                 if (len(dev_hist) >= slope_size):
265                     # Calcular la pendiente
266                     print(getSlope(dev_hist[-slope_size:]))
267                     print(dev_hist[-slope_size:])
268
```

```
269                 if (getSlope(dev_hist[-slope_size:]) > -1e-5 or e==epochs-1):
270 
271                     def printArray(data,data2):
272                         for i in range(len(data)):
273                             a=data[i]
274                             b=data2[i]
275                             print(str(a).replace(".",",")+"\t"+str(b).replace(".",","))
276 
277                     print("-" * 50)
278                     print(e)
279                     printArray(dev_hist,train_hist)
280                     return
281         else:
282             #
                ------------------------------------------------------------------------------
283             # TEST FINAL
284             #
                ------------------------------------------------------------------------------
285             batch_size = 300
286 
287             IN_TOP_5 = 0
288             IN_TOP_10 = 0
289             IN_TOP_20 = 0
290             IN_TOP_50 = 0
291             IN_TOP_100 = 0
292             IN_TOP_1000 = 0
293             real_positions = []
294 
295             items = 0
296 
297             ITRS_TEST = int(len(TX) / batch_size)
298             RES_ITMS_TEST = len(TX) - (ITRS_TEST * batch_size)
299             if (RES_ITMS_TEST > 0): ITRS_TEST += 1  # Si no es @divisin exacta, se ñaade una
                    @iteracin
300 
301             for step in range(ITRS_TEST):
302                 #tss = time.time()
303 
304                 batch_inputs, batch_context = getBatchData(TX, TY, batch_size, step,
                        ITRS_TEST, RES_ITMS_TEST)
305                 top_values,top_indx = session.run([top_v,top_i], feed_dict={train_dataset:
                        batch_inputs, train_labels: batch_context})
306 
307                 for j in range(len(batch_context)):
308                     real_position = int(np.where(top_indx[j, :] == batch_context[j, 0])[0])
309                     real_positions.append(real_position)
310 
311                     IN_TOP_5 += 1 if real_position <= 5 else 0
312                     IN_TOP_10 += 1 if real_position <= 10 else 0
313                     IN_TOP_20 += 1 if real_position <= 20 else 0
314                     IN_TOP_50 += 1 if real_position <= 50 else 0
315                     IN_TOP_100 += 1 if real_position <= 100 else 0
316                     IN_TOP_1000 += 1 if real_position <= 1000 else 0
317 
318                 items += len(batch_inputs)*1.0
319 
320                 PTF = IN_TOP_5 / items
```

```
                PTT = IN_TOP_10 / items
                PTTW = IN_TOP_20 / items
                PTFT = IN_TOP_50 / items
                PTCIEN = IN_TOP_100 / items
                PTMIL = IN_TOP_1000 / items

                print(np.mean(real_positions), np.std(real_positions),
                    np.median(real_positions))
                print(str(PTF).replace(".", ",") + "\t" + str(PTT).replace(".", ",") + "\t"
                      + str(PTTW).replace(".",",") + "\t" + str(PTFT).replace(".", ",") + "\t"
                      + str(PTCIEN).replace(".", ",") + "\t" + str(PTMIL).replace(".",","))

                #print(time.time()-tss)

            print(np.mean(real_positions), np.std(real_positions), np.median(real_positions))

        #-------------------------------------------------------------------------------------------------------------

##########################################################################################################################
# Llamadas
##########################################################################################################################

TEST_NAME="test_042"

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'

#Se obtinenen cada uno de los embeddings aprendidos (incluidas las canciones)
D2V_DM_PRESENT   = np.load('embeddings/d2v_dm/present/EMB_MATRIX')
D2V_DBOW_PRESENT  = np.load('embeddings/d2v_dbow/present/EMB_MATRIX')
W2V  = np.load('embeddings/w2v/EMB_MATRIX')

#Se concatenan los embeddings de los usuarios para generar el perfil
D2V_EMB = np.concatenate((D2V_DM_PRESENT,D2V_DBOW_PRESENT),axis=1) # El reciente = Reciente
    DM + Reciente DBOW

#Se cargan los datos de train de las canciones para conocer los IDs reales
DATA = np.load("datos/Extend/ARRAY_ARRAYS_TRAIN.pkl")

#Obtener una lista con los ID de canciones que no aestn el la matriz de embeddings
ALL = set(list(range(W2V.shape[0])))
REAL = set(list(set(list(map(int, DATA)))))
NOT_EXISTS = list(ALL.difference(REAL))

##########################################################################################################################
# TENSORFLOW
##########################################################################################################################

#-------------------------------------------------------------------------------------------------------------
# Grid Search + Early Stopping
#-------------------------------------------------------------------------------------------------------------
'''
X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))
rates = [0.000001,1, 0.01, 0.0001]

for r in rates:
    config = {"batch_size": 1024, "learning_rate": r, "epochs": 1000, "seed": 100}

```

```
376     print("Learning Rate: "+str(r))
377     print("-" * 50)
378     trainNet(X, Y, DX, DY, TX, TY, config, user_size, train=True, save=False)
379     print("-"*50)
380 
381 exit()
382 '''
383 #------------------------------------------------------------------------------
384 # Train
385 #------------------------------------------------------------------------------
386 '''
387 X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))
388 
389 #Para esta fase, se une TRAIN y DEV
390 X = np.concatenate((X,DX),axis=0)
391 Y = np.concatenate((Y,DY),axis=0)
392 
393 DX=[]
394 DY=[]
395 
396 config = {"batch_size": 1024, "learning_rate": 0.01, "epochs":1000, "seed":100}
397 trainNet(X, Y, DX, DY, TX, TY, config, user_size, train=True, save=True)
398 
399 exit()
400 '''
401 #------------------------------------------------------------------------------
402 # Test
403 #------------------------------------------------------------------------------
404 
405 X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))
406 
407 config = {"batch_size": 1024, "learning_rate": 0.01, "epochs":1000, "seed":100}
408 trainNet(X, Y, DX, DY, TX, TY, config, user_size,train=False, save=True)
```

4.12. TEST 12

```
# -*- coding: utf-8 -*-

import os.path
import os
import tensorflow as tf
import numpy as np
import random
import time
import math
import pandas as pd
import pickle
from collections import Counter
from scipy.stats import linregress

###############################################################################################################
# eMtodos
###############################################################################################################

def getData(REAL_SONG_IDS=[],data ="datos/Append/ARRAY_ARRAYS_REC.pkl",shuffle=True,
    seed=100, train=.7):

    DATA = np.load(data)

    #DATA = DATA[:20];print("ELIMINAR FILTRO")

    #User, cancion 1, cancion 2
    TRAIN = np.empty([0, 3], dtype=np.int32)
    DEV = np.empty([0, 3], dtype=np.int32)
    TEST = np.empty([0, 3], dtype=np.int32)

    for u in range(len(DATA)):

        # Eliminar canciones escuchadas que no tienen embedding
        USER_PLAYS = DATA[u]
        USER_PLAYS = pd.DataFrame(list(map(int, USER_PLAYS)))
        USER_PLAYS = USER_PLAYS.loc[USER_PLAYS[0].isin(REAL_SONG_IDS)][0].values

        #Separar en train dev test
        TRAIN_ITEMS = int(len(USER_PLAYS)*train)
        DEV_ITEMS = int((len(USER_PLAYS)-TRAIN_ITEMS)/2)

        TRAIN_DATA = USER_PLAYS[:TRAIN_ITEMS]
        DEV_DATA = USER_PLAYS[TRAIN_ITEMS:TRAIN_ITEMS+DEV_ITEMS]
        TEST_DATA = USER_PLAYS[TRAIN_ITEMS+DEV_ITEMS:]

        # Crear ejemplos
        USR_TRAIN = np.empty([len(TRAIN_DATA)-1, 3], dtype=np.int32)
        USR_TRAIN[:,0] = u
        USR_TRAIN[:,1] = TRAIN_DATA[:len(TRAIN_DATA)-1]
        USR_TRAIN[:,2] = TRAIN_DATA[1:]

        USR_DEV = np.empty([len(DEV_DATA) - 1, 3], dtype=np.int32)
        USR_DEV[:, 0] = u
        USR_DEV[:, 1] = DEV_DATA[:len(DEV_DATA) - 1]
        USR_DEV[:, 2] = DEV_DATA[1:]
```

```
55
56          USR_TEST = np.empty([len(TEST_DATA) - 1, 3], dtype=np.int32)
57          USR_TEST[:, 0] = u
58          USR_TEST[:, 1] = TEST_DATA[:len(TEST_DATA) - 1]
59          USR_TEST[:, 2] = TEST_DATA[1:]
60
61
62          # nAadir al global
63          TRAIN = np.concatenate([TRAIN,USR_TRAIN])
64          DEV = np.concatenate([DEV,USR_DEV])
65          TEST = np.concatenate([TEST,USR_TEST])
66
67
68      #Mezclar
69      if(shuffle):
70          np.random.seed(seed)
71          np.random.shuffle(TRAIN)
72          np.random.shuffle(DEV)
73          np.random.shuffle(TEST)
74
75
76      return
            TRAIN[:,:2],np.matrix(TRAIN[:,-1]).T,DEV[:,:2],np.matrix(DEV[:,-1]).T,TEST[:,:2],np.matrix(TEST
77
78  def getBatchData(X, Y, batch_size, step, ITRS, RES_ITMS):
79
80      # Si se noaadi una oiteracin, introducir los datos restantes
81      if (step == ITRS - 1 and RES_ITMS > 0):
82          batch_inputs = X[(ITRS - 1) * batch_size:]
83          batch_context = Y[(ITRS - 1) * batch_size:]
84      # Si no, es un batch normal
85      else:
86          st = step * batch_size
87          nd = (step + 1) * batch_size
88
89          batch_inputs = X[st:nd]
90          batch_context = Y[st:nd]
91      return(batch_inputs,batch_context)
92
93  def trainNet(X,Y, DX,DY,TX,TY,config, user_size, train=True, save=True):
94
95      def getSlope(data):
96          r = linregress(range(len(data)), data)
97          return r.slope
98
99      batch_size = config['batch_size']
100     learning_rate = config['learning_rate']
101     epochs = config['epochs']
102
103     user_size = user_size
104     song_size = W2V.shape[0]
105
106     user_emb_size = 128
107     user_concat_size = user_emb_size * 2
108     song_emb_size = 64
109     hidden_size = 32
110
```

```
111     profile_size = user_emb_size + song_emb_size
112
113     num_sampled = 64  # Ejemplos negativos (Para el NSE)
114
115     model_name = 'model'
116     model_path = 'models/' + TEST_NAME + '/model'
117     complete_path = model_path + "/" + model_name
118
119     # Se almacena la loss de dev de las 5 ultimas epochs
120     slope_size = 100
121     train_hist = []
122     dev_hist = []
123
124     # Creacin del grafo de TF.
125     graph = tf.Graph()
126
127     with graph.as_default():
128
129         if(config["seed"]!=1):
130             tf.set_random_seed(config["seed"])
131
132         # Nmero global de iteraciones
133         global_step = tf.Variable(0, dtype=tf.int32, trainable=False, name='global_step')
134
135         # Datos de entrada
                ------------------------------------------------------------------------------
136         # Array del tamao del batch con las X
137         train_dataset = tf.placeholder(tf.int32, shape=[None, 2], name="train_inputs")
138
139         # Array del tamao del batch con las Y asociadas
140         train_labels = tf.placeholder(tf.float32, shape=[None, 1], name="train_labels")
141
142         # Embeddings
                ------------------------------------------------------------------------------
143
144         D2V_EMB_VRB = tf.Variable(D2V_EMB, trainable=False, name="doc_embeddings")
145
146         T0 = tf.Variable(tf.truncated_normal([user_concat_size, user_emb_size], mean=0.0,
                stddev=1.0 / math.sqrt(hidden_size)), name="T0")
147         B0 = tf.Variable(tf.zeros([user_emb_size]), name="B0")
148
149         W2V_EMB_VRB = tf.Variable(W2V, trainable=False, name="word_embeddings")
150
151         # Operaciones
                ------------------------------------------------------------------------------
152
153         # Embedding de documento
154         ed = tf.nn.embedding_lookup(D2V_EMB_VRB, train_dataset[:, 0])
155
156         #Transformar concat documento de 256 a 128
157         ed = tf.matmul(ed,T0)+B0
158
159         # Embedding de cancion
160         ec = tf.nn.embedding_lookup(W2V_EMB_VRB, train_dataset[:, 1])
161
162         # Clculo de LOSS y optimizador
                ------------------------------------------------------------------------------
```

```

        embed = []
        embed.append(ed)
        embed.append(ec)
        embed = tf.concat(embed, 1)

        # softmax weights, W and D vectors should be concatenated before applying softmax
        weights = tf.Variable(tf.truncated_normal([song_size, profile_size], stddev=1.0 / math.sqrt(song_size)),
                              name="weights")
        # softmax biases
        biases = tf.Variable(tf.zeros([song_size]), name="biases")

        # aClculo de LOSS y optimizador------------------------------------------------------------

        nce_loss = tf.nn.nce_loss(weights=weights,
                                  biases=biases,
                                  labels=train_labels,
                                  inputs=embed,
                                  num_sampled=num_sampled,
                                  num_classes=song_size,
                                  name="nce_loss")

        batch_loss = nce_loss
        nce_loss = tf.reduce_mean(nce_loss)

        logits = tf.matmul(embed, tf.transpose(weights))
        logits = tf.nn.bias_add(logits, biases)
        # logits = tf.nn.softmax(logits,axis=0)
        top_v,top_i = tf.nn.top_k(logits,k=song_size)

        # optimizer = tf.train.AdamOptimizer(learning_rate).minimize(nce_loss,global_step=global_step)
        train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(nce_loss, global_step=global_step)

        # Inicializar variables
        init = tf.global_variables_initializer()

        # Crear objeto encargado de almacenar la red
        saver = tf.train.Saver(max_to_keep=1)

    #config = tf.ConfigProto()
    #config.gpu_options.allow_growth = True
    #with tf.Session(graph=graph, config=config) as session:
    with tf.Session(graph=graph) as session:

        # We must initialize all variables before we use them.
        init.run()


        if(save):
            # Obtener el checkpoint
            ckpt = tf.train.get_checkpoint_state(model_path)

            # Si existe el checkpoint restaurar
            if ckpt and ckpt.model_checkpoint_path:
```

```
216             saver.restore(session, ckpt.model_checkpoint_path)
217
218         #Si se desea entrenar
219         if(train):
220             #-----------------------------------------------------------------------------------------------------------
221             # TRAIN
222             #-----------------------------------------------------------------------------------------------------------
223
224             ITRS= int(len(X)/batch_size)
225             RES_ITMS = len(X) - (ITRS*batch_size)
226             if(RES_ITMS>0):ITRS+=1 #Si no es @divisin exacta, se ñaade una @iteracin
227
228             for e in range(epochs):
229                 TRAIN_LOSS = 0.0
230
231                 print("Epoch "+str(e))
232
233                 for step in range(ITRS):
234
235                     batch_inputs, batch_context =
                            getBatchData(X,Y,batch_size,step,ITRS,RES_ITMS)
236
237                     feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
238
239                     _, loss_val, batch_lss, gs = session.run([train_step,
                            nce_loss,batch_loss,global_step], feed_dict=feed_dict)
240
241                     TRAIN_LOSS += np.sum(batch_lss)
242
243                 #print('Epoch '+str(e)+', last batch loss: '+str(loss_val))
244
245                 if (save): saver.save(session, complete_path, global_step=global_step)
246
247                 #-----------------------------------------------------------------------------------------------------------
248                 # DEV
249                 #-----------------------------------------------------------------------------------------------------------
250
251                 if(len(DX)>0):
252
253                     ITRS_TEST = int(len(DX) / batch_size)
254                     RES_ITMS_TEST = len(DX) - (ITRS_TEST * batch_size)
255                     TEST_LOSS = 0.0
256                     if (RES_ITMS_TEST > 0): ITRS_TEST += 1 # Si no es @divisin exacta, se
                            ñaade una @iteracin
257
258
259                     for step in range(ITRS_TEST):
260
261                         batch_inputs, batch_context =
                                getBatchData(DX,DY,batch_size,step,ITRS_TEST,RES_ITMS_TEST)
262
263                         feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
264                         loss_val,batch_lss = session.run([nce_loss,batch_loss],
                                feed_dict=feed_dict)
265
266                         TEST_LOSS+=np.sum(batch_lss)
267
```

```
268                 dev_hist.append(TEST_LOSS / (len(DX) * 1.0))
269                 train_hist.append(TRAIN_LOSS / (len(X) * 1.0))
270 
271             if (len(dev_hist) >= slope_size):
272                 # Calcular la pendiente
273                 print(getSlope(dev_hist[-slope_size:]))
274                 print(dev_hist[-slope_size:])
275 
276                 if (getSlope(dev_hist[-slope_size:]) > -1e-5 or e==epochs-1):
277 
278                     def printArray(data,data2):
279                         for i in range(len(data)):
280                             a=data[i]
281                             b=data2[i]
282                             print(str(a).replace(".",",")+"\t"+str(b).replace(".",","))
283 
284                     print("-" * 50)
285                     print(e)
286                     printArray(dev_hist,train_hist)
287                     return
288         else:
289             #
                ------------------------------------------------------------------------------
290             # TEST FINAL
291             #
                ------------------------------------------------------------------------------
292             batch_size = 300
293 
294             IN_TOP_5 = 0
295             IN_TOP_10 = 0
296             IN_TOP_20 = 0
297             IN_TOP_50 = 0
298             IN_TOP_100 = 0
299             IN_TOP_1000 = 0
300             real_positions = []
301 
302             items = 0
303 
304             ITRS_TEST = int(len(TX) / batch_size)
305             RES_ITMS_TEST = len(TX) - (ITRS_TEST * batch_size)
306             if (RES_ITMS_TEST > 0): ITRS_TEST += 1  # Si no es @divisin exacta, se ¤aade una
                    @iteracin
307 
308             for step in range(ITRS_TEST):
309                 #tss = time.time()
310 
311                 batch_inputs, batch_context = getBatchData(TX, TY, batch_size, step,
                        ITRS_TEST, RES_ITMS_TEST)
312                 top_values,top_indx = session.run([top_v,top_i], feed_dict={train_dataset:
                        batch_inputs, train_labels: batch_context})
313 
314                 for j in range(len(batch_context)):
315                     real_position = int(np.where(top_indx[j, :] == batch_context[j, 0])[0])
316                     real_positions.append(real_position)
317 
318                     IN_TOP_5 += 1 if real_position <= 5 else 0
319                     IN_TOP_10 += 1 if real_position <= 10 else 0
```

```
320                     IN_TOP_20 += 1 if real_position <= 20 else 0
321                     IN_TOP_50 += 1 if real_position <= 50 else 0
322                     IN_TOP_100 += 1 if real_position <= 100 else 0
323                     IN_TOP_1000 += 1 if real_position <= 1000 else 0
324
325                 items += len(batch_inputs)*1.0
326
327                 PTF = IN_TOP_5 / items
328                 PTT = IN_TOP_10 / items
329                 PTTW = IN_TOP_20 / items
330                 PTFT = IN_TOP_50 / items
331                 PTCIEN = IN_TOP_100 / items
332                 PTMIL = IN_TOP_1000 / items
333
334                 print(np.mean(real_positions), np.std(real_positions),
                        np.median(real_positions))
335                 print(str(PTF).replace(".", ",") + "\t" + str(PTT).replace(".", ",") + "\t"
336                       + str(PTTW).replace(".",",") + "\t" + str(PTFT).replace(".", ",") + "\t"
337                       + str(PTCIEN).replace(".", ",") + "\t" + str(PTMIL).replace(".",","))
338
339                 #print(time.time()-tss)
340
341             print(np.mean(real_positions), np.std(real_positions), np.median(real_positions))
342
343         #-------------------------------------------------------------------------------------------------------------
344
345 ##########################################################################################################################
346 # Llamadas
347 ##########################################################################################################################
348
349 TEST_NAME="test_052"
350
351 os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
352
353 #Se obtinenen cada uno de los embeddings aprendidos (incluidas las canciones)
354 D2V_DM_COMPLETE   = np.load('embeddings/d2v_dm/complete/EMB_MATRIX')
355 D2V_DBOW_COMPLETE = np.load('embeddings/d2v_dbow/complete/EMB_MATRIX')
356
357 #Se obtinenen cada uno de los embeddings aprendidos (incluidas las canciones)
358 D2V_DM_PRESENT  = np.load('embeddings/d2v_dm/present/EMB_MATRIX')
359 D2V_DBOW_PRESENT = np.load('embeddings/d2v_dbow/present/EMB_MATRIX')
360
361 W2V  = np.load('embeddings/w2v/EMB_MATRIX')
362
363 #Se concatenan los embeddings de los usuarios para generar el perfil
364 D2V_CONS = np.concatenate((D2V_DM_COMPLETE,D2V_DBOW_COMPLETE),axis=1) # El consolidado =
        Consolidado DM + Consolidado DBOW
365 D2V_PRES = np.concatenate((D2V_DM_PRESENT,D2V_DBOW_PRESENT),axis=1) # El presente = Presente
        DM + Presente DBOW
366
367 D2V_EMB = np.concatenate((D2V_CONS,D2V_PRES),axis=1) # El perfil completo CONS + PRES
368
369 #Se cargan los datos de train de las canciones para conocer los IDs reales
370 DATA = np.load("datos/Extend/ARRAY_ARRAYS_TRAIN.pkl")
371
372 #Obtener una lista con los ID de canciones que no aestn el la matriz de embeddings
373 ALL = set(list(range(W2V.shape[0])))
```

```
REAL = set(list(set(list(map(int, DATA)))))
NOT_EXISTS = list(ALL.difference(REAL))

#########################################################################################################
# TENSORFLOW
#########################################################################################################

#-------------------------------------------------------------------------------------------------------
# Grid Search + Early Stopping
#-------------------------------------------------------------------------------------------------------
'''
X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))
rates = [0.000001,1, 0.01, 0.0001]

for r in rates:
    config = {"batch_size": 1024, "learning_rate": r, "epochs": 1000, "seed": 100}

    print("Learning Rate: "+str(r))
    print("-" * 50)
    trainNet(X, Y, DX, DY, TX, TY, config, user_size, train=True, save=False)
    print("-"*50)

exit()
'''
#-------------------------------------------------------------------------------------------------------
# Train
#-------------------------------------------------------------------------------------------------------
'''
X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))

#Para esta fase, se une TRAIN y DEV
X = np.concatenate((X,DX),axis=0)
Y = np.concatenate((Y,DY),axis=0)

DX=[]
DY=[]

config = {"batch_size": 1024, "learning_rate": 0.01, "epochs":164, "seed":100}
trainNet(X, Y, DX, DY, TX, TY, config, user_size, train=True, save=True)

exit()
'''
#-------------------------------------------------------------------------------------------------------
# Test
#-------------------------------------------------------------------------------------------------------

X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))

config = {"batch_size": 1024, "learning_rate": 0.01, "epochs":164, "seed":100}
trainNet(X, Y, DX, DY, TX, TY, config, user_size,train=False, save=True)
```

4.13. TEST 13

```
# -*- coding: utf-8 -*-

import os.path
import os
import tensorflow as tf
import numpy as np
import random
import time
import math
import pandas as pd
import pickle
from collections import Counter
from scipy.stats import linregress
from tensorflow.python.framework import ops as framework_ops

########################################################################################################################
# @Mtodos
########################################################################################################################

def getData(REAL_SONG_IDS=[],data ="datos/Append/ARRAY_ARRAYS_REC.pkl",shuffle=True,
    seed=100, train=.7):

    DATA = np.load(data)

    #DATA = DATA[:5];print("ELIMINAR FILTRO")

    #User, cancion 1, cancion 2
    TRAIN = np.empty([0, 3], dtype=np.int32)
    DEV = np.empty([0, 3], dtype=np.int32)
    TEST = np.empty([0, 3], dtype=np.int32)

    for u in range(len(DATA)):

        # Eliminar canciones escuchadas que no tienen embedding
        USER_PLAYS = DATA[u]
        USER_PLAYS = pd.DataFrame(list(map(int, USER_PLAYS)))
        USER_PLAYS = USER_PLAYS.loc[USER_PLAYS[0].isin(REAL_SONG_IDS)][0].values

        #Separar en train dev test
        TRAIN_ITEMS = int(len(USER_PLAYS)*train)
        DEV_ITEMS = int((len(USER_PLAYS)-TRAIN_ITEMS)/2)

        TRAIN_DATA = USER_PLAYS[:TRAIN_ITEMS]
        DEV_DATA = USER_PLAYS[TRAIN_ITEMS:TRAIN_ITEMS+DEV_ITEMS]
        TEST_DATA = USER_PLAYS[TRAIN_ITEMS+DEV_ITEMS:]

        # Crear ejemplos
        USR_TRAIN = np.empty([len(TRAIN_DATA)-1, 3], dtype=np.int32)
        USR_TRAIN[:,0] = u
        USR_TRAIN[:,1] = TRAIN_DATA[:len(TRAIN_DATA)-1]
        USR_TRAIN[:,2] = TRAIN_DATA[1:]

        USR_DEV = np.empty([len(DEV_DATA) - 1, 3], dtype=np.int32)
        USR_DEV[:, 0] = u
        USR_DEV[:, 1] = DEV_DATA[:len(DEV_DATA) - 1]
```

```
        USR_DEV[:, 2] = DEV_DATA[1:]

        USR_TEST = np.empty([len(TEST_DATA) - 1, 3], dtype=np.int32)
        USR_TEST[:, 0] = u
        USR_TEST[:, 1] = TEST_DATA[:len(TEST_DATA) - 1]
        USR_TEST[:, 2] = TEST_DATA[1:]


        # nAadir al global
        TRAIN = np.concatenate([TRAIN,USR_TRAIN])
        DEV = np.concatenate([DEV,USR_DEV])
        TEST = np.concatenate([TEST,USR_TEST])


    #Mezclar
    if(shuffle):
        np.random.seed(seed)
        np.random.shuffle(TRAIN)
        np.random.shuffle(DEV)
        np.random.shuffle(TEST)


    return
        TRAIN[:,:2],np.matrix(TRAIN[:,-1]).T,DEV[:,:2],np.matrix(DEV[:,-1]).T,TEST[:,:2],np.matrix(TEST

def getBatchData(X, Y, batch_size, step, ITRS, RES_ITMS):

    # Si se no aadi una iteracin, introducir los datos restantes
    if (step == ITRS - 1 and RES_ITMS > 0):
        batch_inputs = X[(ITRS - 1) * batch_size:]
        batch_context = Y[(ITRS - 1) * batch_size:]
    # Si no, es un batch normal
    else:
        st = step * batch_size
        nd = (step + 1) * batch_size

        batch_inputs = X[st:nd]
        batch_context = Y[st:nd]
    return(batch_inputs,batch_context)

def trainNet(X,Y, DX,DY,TX,TY,config, user_size, train=True, save=True):

    def getSlope(data):
        r = linregress(range(len(data)), data)
        return r.slope

    batch_size = config['batch_size']
    learning_rate = config['learning_rate']
    epochs = config['epochs']

    user_size = user_size
    song_size = W2V.shape[0]

    user_emb_size = 128
    song_emb_size = 64
    hidden_size = 32

```

```
    profile_size = user_emb_size + song_emb_size

    num_sampled = 64  # Ejemplos negativos (Para el NSE)

    model_name = 'model'
    model_path = 'models/' + TEST_NAME + '/model'
    complete_path = model_path + "/" + model_name

    # Se almacena la loss de dev de las 5 ultimas epochs
    slope_size = 100
    train_hist = []
    dev_hist = []

    # Creacin del grafo de TF.
    graph = tf.Graph()

    with graph.as_default():

        if(config["seed"]!=1):
            tf.set_random_seed(config["seed"])

        # Nmero global de iteraciones
        global_step = tf.Variable(0, dtype=tf.int32, trainable=False, name='global_step')

        # Datos de entrada
        # ------------------------------------------------------------------------------------

        # Array del tamao del batch con las X
        train_dataset = tf.placeholder(tf.int32, shape=[None, 2], name="train_inputs")

        # Array del tamao del batch con las Y asociadas
        train_labels = tf.placeholder(tf.float32, shape=[None, 1], name="train_labels")

        # Embeddings
        # ------------------------------------------------------------------------------------

        O1 = tf.Variable(tf.truncated_normal([user_size, user_emb_size], mean=0.0, stddev=1.0
            / math.sqrt(user_emb_size)), name="O1")
        BO1 = tf.Variable(tf.zeros([user_emb_size]), name="BO1")

        O2 = tf.Variable(tf.truncated_normal([song_size, song_emb_size], mean=0.0, stddev=1.0
            / math.sqrt(song_emb_size)), name="O2")
        BO2 = tf.Variable(tf.zeros([song_emb_size]), name="BO2")

        # Operaciones
        # ------------------------------------------------------------------------------------

        # Embedding de documento
        ed = tf.nn.embedding_lookup(O1, train_dataset[:, 0])
        ed = ed + BO1

        # Embedding de cancion
        ec = tf.nn.embedding_lookup(O2, train_dataset[:, 1])
        ec = ec + BO2

        # Clculo de LOSS y optimizador
        # ------------------------------------------------------------------------------------
```

```
162         embed = []
163         embed.append(ed)
164         embed.append(ec)
165         embed = tf.concat(embed, 1)
166 
167         # softmax weights, W and D vectors should be concatenated before applying softmax
168         weights = tf.Variable(tf.truncated_normal([song_size, profile_size], stddev=1.0 /
                math.sqrt(song_size)),
169                               name="weights")
170         # softmax biases
171         biases = tf.Variable(tf.zeros([song_size]), name="biases")
172 
173         # aClculo de LOSS y optimizador—
                ------------------------------------------------------------------------------
174         nce_loss = tf.nn.nce_loss(weights=weights,
175                                   biases=biases,
176                                   labels=train_labels,
177                                   inputs=embed,
178                                   num_sampled=num_sampled,
179                                   num_classes=song_size,
180                                   name="nce_loss")
181 
182         batch_loss = nce_loss
183         nce_loss = tf.reduce_mean(nce_loss)
184 
185         logits = tf.matmul(embed, tf.transpose(weights))
186         logits = tf.nn.bias_add(logits, biases)
187 
188         top_v,top_i = tf.nn.top_k(logits,k=song_size)
189 
190         # logits = tf.nn.softmax(logits,axis=0)
191 
192         # optimizer =
                tf.train.AdamOptimizer(learning_rate).minimize(nce_loss,global_step=global_step)
193         train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(nce_loss,
                global_step=global_step)
194 
195         # Inicializar variables
196         init = tf.global_variables_initializer()
197 
198         # Crear objeto encargado de almacenar la red
199         saver = tf.train.Saver(max_to_keep=1)
200 
201     #config = tf.ConfigProto()
202     #config.gpu_options.allow_growth = True
203     #with tf.Session(graph=graph, config=config) as session:
204     with tf.Session(graph=graph) as session:
205 
206         # We must initialize all variables before we use them.
207         init.run()
208 
209 
210         if(save):
211             # Obtener el checkpoint
212             ckpt = tf.train.get_checkpoint_state(model_path)
213 
214             # Si existe el checkpoint restaurar
```

```
215             if ckpt and ckpt.model_checkpoint_path:
216                 saver.restore(session, ckpt.model_checkpoint_path)
217 
218         #Si se desea entrenar
219         if(train):
220             #-----------------------------------------------------------------------------------------------
221             # TRAIN
222             #-----------------------------------------------------------------------------------------------
223 
224             ITRS= int(len(X)/batch_size)
225             RES_ITMS = len(X) - (ITRS*batch_size)
226             if(RES_ITMS>0):ITRS+=1 #Si no es divisin exacta, se naade una iteracin
227 
228             for e in range(epochs):
229                 TRAIN_LOSS = 0.0
230 
231                 print("Epoch "+str(e))
232 
233                 for step in range(ITRS):
234 
235                     batch_inputs, batch_context =
                            getBatchData(X,Y,batch_size,step,ITRS,RES_ITMS)
236 
237                     feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
238 
239                     _, loss_val, batch_lss, gs = session.run([train_step,
                            nce_loss,batch_loss,global_step], feed_dict=feed_dict)
240 
241                     TRAIN_LOSS += np.sum(batch_lss)
242 
243                 #print('Epoch '+str(e)+', last batch loss: '+str(loss_val))
244 
245                 if (save): saver.save(session, complete_path, global_step=global_step)
246 
247                 #-----------------------------------------------------------------------------------------------
248                 # DEV
249                 #-----------------------------------------------------------------------------------------------
250 
251                 if(len(DX)>0):
252 
253                     ITRS_TEST = int(len(DX) / batch_size)
254                     RES_ITMS_TEST = len(DX) - (ITRS_TEST * batch_size)
255                     TEST_LOSS = 0.0
256                     if (RES_ITMS_TEST > 0): ITRS_TEST += 1 # Si no es divisin exacta, se
                            naade una iteracin
257 
258 
259                     for step in range(ITRS_TEST):
260 
261                         batch_inputs, batch_context =
                                getBatchData(DX,DY,batch_size,step,ITRS_TEST,RES_ITMS_TEST)
262 
263                         feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
264                         loss_val,batch_lss = session.run([nce_loss,batch_loss],
                                feed_dict=feed_dict)
265 
266                         TEST_LOSS+=np.sum(batch_lss)
```

```
267
268                 dev_hist.append(TEST_LOSS / (len(DX) * 1.0))
269                 train_hist.append(TRAIN_LOSS / (len(X) *  1.0))
270
271             if (len(dev_hist) >= slope_size):
272                 # Calcular la pendiente
273                 print(getSlope(dev_hist[-slope_size:]))
274                 print(dev_hist[-slope_size:])
275
276                 if (getSlope(dev_hist[-slope_size:]) > -1e-5 or e==epochs-1):
277
278                     def printArray(data,data2):
279                         for i in range(len(data)):
280                             a=data[i]
281                             b=data2[i]
282                             print(str(a).replace(".",",")+"\t"+str(b).replace(".",","))
283
284                     print("-" * 50)
285                     print(e)
286                     printArray(dev_hist,train_hist)
287                     return
288         else:
289             #
                ------------------------------------------------------------------------------
290             # TEST FINAL
291             #
                ------------------------------------------------------------------------------
292             batch_size = 300
293
294             IN_TOP_5 = 0
295             IN_TOP_10 = 0
296             IN_TOP_20 = 0
297             IN_TOP_50 = 0
298             IN_TOP_100 = 0
299             IN_TOP_1000 = 0
300             real_positions = []
301
302             items = 0
303
304             ITRS_TEST = int(len(TX) / batch_size)
305             RES_ITMS_TEST = len(TX) - (ITRS_TEST * batch_size)
306             if (RES_ITMS_TEST > 0): ITRS_TEST += 1  # Si no es divisin exacta, se aade una
                    iteracin
307
308             for step in range(ITRS_TEST):
309                 #tss = time.time()
310
311                 batch_inputs, batch_context = getBatchData(TX, TY, batch_size, step,
                        ITRS_TEST, RES_ITMS_TEST)
312                 top_values,top_indx = session.run([top_v,top_i], feed_dict={train_dataset:
                        batch_inputs, train_labels: batch_context})
313
314                 for j in range(len(batch_context)):
315                     real_position = int(np.where(top_indx[j, :] == batch_context[j, 0])[0])
316                     real_positions.append(real_position)
317
318                     IN_TOP_5 += 1 if real_position <= 5 else 0
```

```
319                     IN_TOP_10 += 1 if real_position <= 10 else 0
320                     IN_TOP_20 += 1 if real_position <= 20 else 0
321                     IN_TOP_50 += 1 if real_position <= 50 else 0
322                     IN_TOP_100 += 1 if real_position <= 100 else 0
323                     IN_TOP_1000 += 1 if real_position <= 1000 else 0
324
325                 items += len(batch_inputs)*1.0
326
327                 PTF = IN_TOP_5 / items
328                 PTT = IN_TOP_10 / items
329                 PTTW = IN_TOP_20 / items
330                 PTFT = IN_TOP_50 / items
331                 PTCIEN = IN_TOP_100 / items
332                 PTMIL = IN_TOP_1000 / items
333
334                 print(np.mean(real_positions), np.std(real_positions),
                        np.median(real_positions))
335                 print(str(PTF).replace(".", ",") + "\t" + str(PTT).replace(".", ",") + "\t"
336                       + str(PTTW).replace(".",",") + "\t" + str(PTFT).replace(".", ",") + "\t"
337                       + str(PTCIEN).replace(".", ",") + "\t" + str(PTMIL).replace(".",","))
338
339                 #print(time.time()-tss)
340
341
342             print(np.mean(real_positions), np.std(real_positions), np.median(real_positions))
343
344         #-------------------------------------------------------------------------------------------------------------
345
346 ##################################################################################################################
347 # Llamadas
348 ##################################################################################################################
349
350 TEST_NAME="test_012"
351
352 os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
353
354 #Solamente se carga para conocer el unmero de canciones
355 W2V  = np.load('embeddings/w2v/EMB_MATRIX')
356
357 #Se cargan los datos de train de las canciones para conocer los IDs reales
358 DATA = np.load("datos/Extend/ARRAY_ARRAYS_TRAIN.pkl")
359
360 #Obtener una lista con los ID de canciones que no aestn el la matriz de embeddings
361 ALL = set(list(range(W2V.shape[0])))
362 REAL = set(list(set(list(map(int, DATA)))))
363
364 ##################################################################################################################
365 # TENSORFLOW
366 ##################################################################################################################
367
368 #-------------------------------------------------------------------------------------------------------------
369 # Grid Search + Early Stopping
370 #-------------------------------------------------------------------------------------------------------------
371 '''
372 X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))
373 rates = [0.000001,1, 0.01, 0.0001]
374
```

```
for r in rates:
    config = {"batch_size": 1024, "learning_rate": r, "epochs": 1000, "seed": 100}

    print("Learning Rate: "+str(r))
    print("-" * 50)
    trainNet(X, Y, DX, DY, TX, TY, config, user_size, train=True, save=False)
    print("-"*50)

exit()
'''
#-------------------------------------------------------------------------------
# Train
#-------------------------------------------------------------------------------
'''
X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))

#Para esta fase, se une TRAIN y DEV
X = np.concatenate((X,DX),axis=0)
Y = np.concatenate((Y,DY),axis=0)

DX=[]
DY=[]

config = {"batch_size": 1024, "learning_rate": 0.01, "epochs":1000, "seed":100}
trainNet(X, Y, DX, DY, TX, TY, config, user_size, train=True, save=True)

exit()
'''
#-------------------------------------------------------------------------------
# Test
#-------------------------------------------------------------------------------

X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))

config = {"batch_size": 1024, "learning_rate": 0.01, "epochs":1000, "seed":100}
trainNet(X, Y, DX, DY, TX, TY, config, user_size,train=False, save=True)
```

4.14. TEST 14

```
# —*— coding: utf—8 —*—

import os.path
import os
import tensorflow as tf
import numpy as np
import random
import time
import math
import pandas as pd
import pickle
from collections import Counter
from scipy.stats import linregress

#########################################################################################################################
# eMtodos
#########################################################################################################################

def getData(REAL_SONG_IDS=[],data ="datos/Append/ARRAY_ARRAYS_REC.pkl",shuffle=True,
    seed=100, train=.7):

    DATA = np.load(data)

    #DATA = DATA[:20];print("ELIMINAR FILTRO")

    #User, cancion 1, cancion 2
    TRAIN = np.empty([0, 3], dtype=np.int32)
    DEV = np.empty([0, 3], dtype=np.int32)
    TEST = np.empty([0, 3], dtype=np.int32)

    for u in range(len(DATA)):

        # Eliminar canciones escuchadas que no tienen embedding
        USER_PLAYS = DATA[u]
        USER_PLAYS = pd.DataFrame(list(map(int, USER_PLAYS)))
        USER_PLAYS = USER_PLAYS.loc[USER_PLAYS[0].isin(REAL_SONG_IDS)][0].values

        #Separar en train dev test
        TRAIN_ITEMS = int(len(USER_PLAYS)*train)
        DEV_ITEMS = int((len(USER_PLAYS)—TRAIN_ITEMS)/2)

        TRAIN_DATA = USER_PLAYS[:TRAIN_ITEMS]
        DEV_DATA = USER_PLAYS[TRAIN_ITEMS:TRAIN_ITEMS+DEV_ITEMS]
        TEST_DATA = USER_PLAYS[TRAIN_ITEMS+DEV_ITEMS:]

        # Crear ejemplos
        USR_TRAIN = np.empty([len(TRAIN_DATA)—1, 3], dtype=np.int32)
        USR_TRAIN[:,0] = u
        USR_TRAIN[:,1] = TRAIN_DATA[:len(TRAIN_DATA)—1]
        USR_TRAIN[:,2] = TRAIN_DATA[1:]

        USR_DEV = np.empty([len(DEV_DATA) — 1, 3], dtype=np.int32)
        USR_DEV[:, 0] = u
        USR_DEV[:, 1] = DEV_DATA[:len(DEV_DATA) — 1]
        USR_DEV[:, 2] = DEV_DATA[1:]
```

```
55
56          USR_TEST = np.empty([len(TEST_DATA) - 1, 3], dtype=np.int32)
57          USR_TEST[:, 0] = u
58          USR_TEST[:, 1] = TEST_DATA[:len(TEST_DATA) - 1]
59          USR_TEST[:, 2] = TEST_DATA[1:]
60
61
62          # Aadir al global
63          TRAIN = np.concatenate([TRAIN,USR_TRAIN])
64          DEV = np.concatenate([DEV,USR_DEV])
65          TEST = np.concatenate([TEST,USR_TEST])
66
67
68      #Mezclar
69      if(shuffle):
70          np.random.seed(seed)
71          np.random.shuffle(TRAIN)
72          np.random.shuffle(DEV)
73          np.random.shuffle(TEST)
74
75
76      return
            TRAIN[:,:2],np.matrix(TRAIN[:,-1]).T,DEV[:,:2],np.matrix(DEV[:,-1]).T,TEST[:,:2],np.matrix(TEST
77
78  def getBatchData(X, Y, batch_size, step, ITRS, RES_ITMS):
79
80      # Si se aadi una iteracin, introducir los datos restantes
81      if (step == ITRS - 1 and RES_ITMS > 0):
82          batch_inputs = X[(ITRS - 1) * batch_size:]
83          batch_context = Y[(ITRS - 1) * batch_size:]
84      # Si no, es un batch normal
85      else:
86          st = step * batch_size
87          nd = (step + 1) * batch_size
88
89          batch_inputs = X[st:nd]
90          batch_context = Y[st:nd]
91      return(batch_inputs,batch_context)
92
93  def trainNet(X,Y, DX,DY,TX,TY,config, user_size, train=True, save=True):
94
95      def getSlope(data):
96          r = linregress(range(len(data)), data)
97          return r.slope
98
99      batch_size = config['batch_size']
100     learning_rate = config['learning_rate']
101     epochs = config['epochs']
102
103     user_size = user_size
104     song_size = W2V.shape[0]
105
106     user_emb_size = 128
107     song_emb_size = 64
108     hidden_size = 32
109
110     profile_size = user_emb_size + song_emb_size
```

```
111
112     num_sampled = 64  # Ejemplos negativos (Para el NSE)
113
114     model_name = 'model'
115     model_path = 'models/' + TEST_NAME + '/model'
116     complete_path = model_path + "/" + model_name
117
118     # Se almacena la loss de dev de las 5 ultimas epochs
119     slope_size = 100
120     train_hist = []
121     dev_hist = []
122
123     # Creacin del grafo de TF.
124     graph = tf.Graph()
125
126     with graph.as_default():
127
128         if(config["seed"]!=1):
129             tf.set_random_seed(config["seed"])
130
131         # Nmero global de iteraciones
132         global_step = tf.Variable(0, dtype=tf.int32, trainable=False, name='global_step')
133
134         # Datos de entrada
            ------------------------------------------------------------------------------------
135         # Array del tamao del batch con las X
136         train_dataset = tf.placeholder(tf.int32, shape=[None, 2], name="train_inputs")
137
138         # Array del tamao del batch con las Y asociadas
139         train_labels = tf.placeholder(tf.float32, shape=[None, 1], name="train_labels")
140
141         # Embeddings
            ------------------------------------------------------------------------------------
142
143         D2V_EMB_VRB = tf.Variable(D2V_EMB, trainable=False, name="doc_embeddings")
144
145         O2 = tf.Variable(tf.truncated_normal([song_size, song_emb_size], mean=0.0, stddev=1.0
                / math.sqrt(song_emb_size)), name="O2")
146         BO2 = tf.Variable(tf.zeros([song_emb_size]), name="BO2")
147
148         # Operaciones
            ------------------------------------------------------------------------------------
149
150         # Embedding de documento
151         ed = tf.nn.embedding_lookup(D2V_EMB_VRB, train_dataset[:, 0])
152
153         # Embedding de cancion
154         ec = tf.nn.embedding_lookup(O2, train_dataset[:, 1])
155         ec = ec + BO2
156
157         # Clculo de LOSS y optimizador
            ------------------------------------------------------------------------------------
158
159         embed = []
160         embed.append(ed)
161         embed.append(ec)
162         embed = tf.concat(embed, 1)
```

```
163
164         # softmax weights, W and D vectors should be concatenated before applying softmax
165         weights = tf.Variable(tf.truncated_normal([song_size, profile_size], stddev=1.0 /
                math.sqrt(song_size)),
166                               name="weights")
167         # softmax biases
168         biases = tf.Variable(tf.zeros([song_size]), name="biases")
169
170         # aClculo de LOSS y optimizador—
            ------------------------------------------------------------------------------------
171         nce_loss = tf.nn.nce_loss(weights=weights,
172                                   biases=biases,
173                                   labels=train_labels,
174                                   inputs=embed,
175                                   num_sampled=num_sampled,
176                                   num_classes=song_size,
177                                   name="nce_loss")
178
179         batch_loss = nce_loss
180         nce_loss = tf.reduce_mean(nce_loss)
181
182         logits = tf.matmul(embed, tf.transpose(weights))
183         logits = tf.nn.bias_add(logits, biases)
184         # logits = tf.nn.softmax(logits,axis=0)
185         top_v,top_i = tf.nn.top_k(logits,k=song_size)
186
187         # optimizer =
                tf.train.AdamOptimizer(learning_rate).minimize(nce_loss,global_step=global_step)
188         train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(nce_loss,
                global_step=global_step)
189
190         # Inicializar variables
191         init = tf.global_variables_initializer()
192
193         # Crear objeto encargado de almacenar la red
194         saver = tf.train.Saver(max_to_keep=1)
195
196     config = tf.ConfigProto()
197     config.gpu_options.allow_growth = True
198     with tf.Session(graph=graph, config=config) as session:
199     #with tf.Session(graph=graph) as session:
200
201         # We must initialize all variables before we use them.
202         init.run()
203
204
205         if(save):
206             # Obtener el checkpoint
207             ckpt = tf.train.get_checkpoint_state(model_path)
208
209             # Si existe el checkpoint restaurar
210             if ckpt and ckpt.model_checkpoint_path:
211                 saver.restore(session, ckpt.model_checkpoint_path)
212
213         #Si se desea entrenar
214         if(train):
215             #------------------------------------------------------------------------------------
```

```
216             # TRAIN
217             #-----------------------------------------------------------------------------
218
219             ITRS= int(len(X)/batch_size)
220             RES_ITMS = len(X) - (ITRS*batch_size)
221             if(RES_ITMS>0):ITRS+=1 #Si no es divisin exacta, se aade una iteracin
222
223             for e in range(epochs):
224                 TRAIN_LOSS = 0.0
225
226                 print("Epoch "+str(e))
227
228                 for step in range(ITRS):
229
230                     batch_inputs, batch_context =
                            getBatchData(X,Y,batch_size,step,ITRS,RES_ITMS)
231
232                     feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
233
234                     _, loss_val, batch_lss, gs = session.run([train_step,
                            nce_loss,batch_loss,global_step], feed_dict=feed_dict)
235
236                     TRAIN_LOSS += np.sum(batch_lss)
237
238                 #print('Epoch '+str(e)+', last batch loss: '+str(loss_val))
239
240                 if (save): saver.save(session, complete_path, global_step=global_step)
241
242                 #-----------------------------------------------------------------------------
243                 # DEV
244                 #-----------------------------------------------------------------------------
245
246                 if(len(DX)>0):
247
248                     ITRS_TEST = int(len(DX) / batch_size)
249                     RES_ITMS_TEST = len(DX) - (ITRS_TEST * batch_size)
250                     TEST_LOSS = 0.0
251                     if (RES_ITMS_TEST > 0): ITRS_TEST += 1 # Si no es divisin exacta, se
                            aade una iteracin
252
253
254                     for step in range(ITRS_TEST):
255
256                         batch_inputs, batch_context =
                                getBatchData(DX,DY,batch_size,step,ITRS_TEST,RES_ITMS_TEST)
257
258                         feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
259                         loss_val,batch_lss = session.run([nce_loss,batch_loss],
                                feed_dict=feed_dict)
260
261                         TEST_LOSS+=np.sum(batch_lss)
262
263                     dev_hist.append(TEST_LOSS / (len(DX) * 1.0))
264                     train_hist.append(TRAIN_LOSS / (len(X) *1.0))
265
266                 if (len(dev_hist) >= slope_size):
267                     # Calcular la pendiente
```

```
268                 print(getSlope(dev_hist[-slope_size:]))
269                 print(dev_hist[-slope_size:])
270 
271                 if (getSlope(dev_hist[-slope_size:]) > -1e-5 or e==epochs-1):
272 
273                     def printArray(data,data2):
274                         for i in range(len(data)):
275                             a=data[i]
276                             b=data2[i]
277                             print(str(a).replace(".",",")+"\t"+str(b).replace(".",","))
278 
279                     print("-" * 50)
280                     print(e)
281                     printArray(dev_hist,train_hist)
282                     return
283         else:
284             #
                ------------------------------------------------------------------------------
285             # TEST FINAL
286             #
                ------------------------------------------------------------------------------
287             batch_size = 300
288 
289             IN_TOP_5 = 0
290             IN_TOP_10 = 0
291             IN_TOP_20 = 0
292             IN_TOP_50 = 0
293             IN_TOP_100 = 0
294             IN_TOP_1000 = 0
295             real_positions = []
296 
297             items = 0
298 
299             ITRS_TEST = int(len(TX) / batch_size)
300             RES_ITMS_TEST = len(TX) - (ITRS_TEST * batch_size)
301             if (RES_ITMS_TEST > 0): ITRS_TEST += 1  # Si no es @divisin exacta, se ñaade una
                    @iteracin
302 
303             for step in range(ITRS_TEST):
304                 #tss = time.time()
305 
306                 batch_inputs, batch_context = getBatchData(TX, TY, batch_size, step,
                        ITRS_TEST, RES_ITMS_TEST)
307                 top_values,top_indx = session.run([top_v,top_i], feed_dict={train_dataset:
                        batch_inputs, train_labels: batch_context})
308 
309                 for j in range(len(batch_context)):
310                     real_position = int(np.where(top_indx[j, :] == batch_context[j, 0])[0])
311                     real_positions.append(real_position)
312 
313                     IN_TOP_5 += 1 if real_position <= 5 else 0
314                     IN_TOP_10 += 1 if real_position <= 10 else 0
315                     IN_TOP_20 += 1 if real_position <= 20 else 0
316                     IN_TOP_50 += 1 if real_position <= 50 else 0
317                     IN_TOP_100 += 1 if real_position <= 100 else 0
318                     IN_TOP_1000 += 1 if real_position <= 1000 else 0
319 
```

```
320                 items += len(batch_inputs)*1.0
321
322                 PTF = IN_TOP_5 / items
323                 PTT = IN_TOP_10 / items
324                 PTTW = IN_TOP_20 / items
325                 PTFT = IN_TOP_50 / items
326                 PTCIEN = IN_TOP_100 / items
327                 PTMIL = IN_TOP_1000 / items
328
329                 print(np.mean(real_positions), np.std(real_positions),
                        np.median(real_positions))
330                 print(str(PTF).replace(".", ",") + "\t" + str(PTT).replace(".", ",") + "\t"
331                       + str(PTTW).replace(".",",") + "\t" + str(PTFT).replace(".", ",") + "\t"
332                       + str(PTCIEN).replace(".", ",") + "\t" + str(PTMIL).replace(".",","))
333
334                 #print(time.time()-tss)
335
336             print(np.mean(real_positions), np.std(real_positions), np.median(real_positions))
337
338         #-------------------------------------------------------------------------------------------------------------
339
340 ###############################################################################################################
341 # Llamadas
342 ###############################################################################################################
343
344 TEST_NAME="test_062"
345
346 os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
347
348 #Se obtinenen cada uno de los embeddings aprendidos (incluidas las canciones)
349 D2V_DM_COMPLETE   = np.load('embeddings/d2v_dm/complete/EMB_MATRIX')
350 D2V_DBOW_COMPLETE = np.load('embeddings/d2v_dbow/complete/EMB_MATRIX')
351 W2V  = np.load('embeddings/w2v/EMB_MATRIX')
352
353 #Se concatenan los embeddings de los usuarios para generar el perfil
354 D2V_EMB = np.concatenate((D2V_DM_COMPLETE,D2V_DBOW_COMPLETE),axis=1) # El consolidado =
        Consolidado DM + Consolidado DBOW
355
356 #Se cargan los datos de train de las canciones para conocer los IDs reales
357 DATA = np.load("datos/Extend/ARRAY_ARRAYS_TRAIN.pkl")
358
359 #Obtener una lista con los ID de canciones que no aestn el la matriz de embeddings
360 ALL = set(list(range(W2V.shape[0])))
361 REAL = set(list(set(list(map(int, DATA)))))
362 NOT_EXISTS = list(ALL.difference(REAL))
363
364 ###############################################################################################################
365 # TENSORFLOW
366 ###############################################################################################################
367
368 #-------------------------------------------------------------------------------------------------------------
369 # Grid Search + Early Stopping
370 #-------------------------------------------------------------------------------------------------------------
371 '''
372 X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))
373 rates = [0.000001,1, 0.01, 0.0001]
374
```

```
for r in rates:
    config = {"batch_size": 1024, "learning_rate": r, "epochs": 1000, "seed": 100}

    print("Learning Rate: "+str(r))
    print("-" * 50)
    trainNet(X, Y, DX, DY, TX, TY, config, user_size, train=True, save=False)
    print("-"*50)

exit()
'''
#-------------------------------------------------------------------------------
# Train
#-------------------------------------------------------------------------------
'''
X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))

#Para esta fase, se une TRAIN y DEV
X = np.concatenate((X,DX),axis=0)
Y = np.concatenate((Y,DY),axis=0)

DX=[]
DY=[]

config = {"batch_size": 1024, "learning_rate": 0.01, "epochs":624, "seed":100}
trainNet(X, Y, DX, DY, TX, TY, config, user_size, train=True, save=True)

exit()
'''
#-------------------------------------------------------------------------------
# Test
#-------------------------------------------------------------------------------

X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))

config = {"batch_size": 1024, "learning_rate": 0.01, "epochs":624, "seed":100}
trainNet(X, Y, DX, DY, TX, TY, config, user_size,train=False, save=True)
```

4.15. TEST 15

```
# —*— coding: utf—8 —*—

import os.path
import os
import tensorflow as tf
import numpy as np
import random
import time
import math
import pandas as pd
import pickle
from collections import Counter
from scipy.stats import linregress

#########################################################################################################
# eMtodos
#########################################################################################################

def getData(REAL_SONG_IDS=[],data ="datos/Append/ARRAY_ARRAYS_REC.pkl",shuffle=True,
    seed=100, train=.7):

    DATA = np.load(data)

    #DATA = DATA[:20];print("ELIMINAR FILTRO")

    #User, cancion 1, cancion 2
    TRAIN = np.empty([0, 3], dtype=np.int32)
    DEV = np.empty([0, 3], dtype=np.int32)
    TEST = np.empty([0, 3], dtype=np.int32)

    for u in range(len(DATA)):

        # Eliminar canciones escuchadas que no tienen embedding
        USER_PLAYS = DATA[u]
        USER_PLAYS = pd.DataFrame(list(map(int, USER_PLAYS)))
        USER_PLAYS = USER_PLAYS.loc[USER_PLAYS[0].isin(REAL_SONG_IDS)][0].values

        #Separar en train dev test
        TRAIN_ITEMS = int(len(USER_PLAYS)*train)
        DEV_ITEMS = int((len(USER_PLAYS)—TRAIN_ITEMS)/2)

        TRAIN_DATA = USER_PLAYS[:TRAIN_ITEMS]
        DEV_DATA = USER_PLAYS[TRAIN_ITEMS:TRAIN_ITEMS+DEV_ITEMS]
        TEST_DATA = USER_PLAYS[TRAIN_ITEMS+DEV_ITEMS:]

        # Crear ejemplos
        USR_TRAIN = np.empty([len(TRAIN_DATA)—1, 3], dtype=np.int32)
        USR_TRAIN[:,0] = u
        USR_TRAIN[:,1] = TRAIN_DATA[:len(TRAIN_DATA)—1]
        USR_TRAIN[:,2] = TRAIN_DATA[1:]

        USR_DEV = np.empty([len(DEV_DATA) — 1, 3], dtype=np.int32)
        USR_DEV[:, 0] = u
        USR_DEV[:, 1] = DEV_DATA[:len(DEV_DATA) — 1]
        USR_DEV[:, 2] = DEV_DATA[1:]
```

```
55
56          USR_TEST = np.empty([len(TEST_DATA) - 1, 3], dtype=np.int32)
57          USR_TEST[:, 0] = u
58          USR_TEST[:, 1] = TEST_DATA[:len(TEST_DATA) - 1]
59          USR_TEST[:, 2] = TEST_DATA[1:]
60
61
62          # nAadir al global
63          TRAIN = np.concatenate([TRAIN,USR_TRAIN])
64          DEV = np.concatenate([DEV,USR_DEV])
65          TEST = np.concatenate([TEST,USR_TEST])
66
67
68      #Mezclar
69      if(shuffle):
70          np.random.seed(seed)
71          np.random.shuffle(TRAIN)
72          np.random.shuffle(DEV)
73          np.random.shuffle(TEST)
74
75
76      return
            TRAIN[:,:2],np.matrix(TRAIN[:,-1]).T,DEV[:,:2],np.matrix(DEV[:,-1]).T,TEST[:,:2],np.matrix(TEST
77
78  def getBatchData(X, Y, batch_size, step, ITRS, RES_ITMS):
79
80      # Si se noaadi una oiteracin, introducir los datos restantes
81      if (step == ITRS - 1 and RES_ITMS > 0):
82          batch_inputs = X[(ITRS - 1) * batch_size:]
83          batch_context = Y[(ITRS - 1) * batch_size:]
84      # Si no, es un batch normal
85      else:
86          st = step * batch_size
87          nd = (step + 1) * batch_size
88
89          batch_inputs = X[st:nd]
90          batch_context = Y[st:nd]
91      return(batch_inputs,batch_context)
92
93  def trainNet(X,Y, DX,DY,TX,TY,config, user_size, train=True, save=True):
94
95      def getSlope(data):
96          r = linregress(range(len(data)), data)
97          return r.slope
98
99      batch_size = config['batch_size']
100     learning_rate = config['learning_rate']
101     epochs = config['epochs']
102
103     user_size = user_size
104     song_size = W2V.shape[0]
105
106     user_emb_size = 128
107     song_emb_size = 64
108     hidden_size = 32
109
110     profile_size = user_emb_size + song_emb_size
```

```
111
112     num_sampled = 64  # Ejemplos negativos (Para el NSE)
113
114     model_name = 'model'
115     model_path = 'models/' + TEST_NAME + '/model'
116     complete_path = model_path + "/" + model_name
117
118     # Se almacena la loss de dev de las 5 ultimas epochs
119     slope_size = 100
120     train_hist = []
121     dev_hist = []
122
123     # Creacin del grafo de TF.
124     graph = tf.Graph()
125
126     with graph.as_default():
127
128         if(config["seed"]!=1):
129             tf.set_random_seed(config["seed"])
130
131         # Nmero global de iteraciones
132         global_step = tf.Variable(0, dtype=tf.int32, trainable=False, name='global_step')
133
134         # Datos de entrada
            ------------------------------------------------------------------------------------
135         # Array del tamao del batch con las X
136         train_dataset = tf.placeholder(tf.int32, shape=[None, 2], name="train_inputs")
137
138         # Array del tamao del batch con las Y asociadas
139         train_labels = tf.placeholder(tf.float32, shape=[None, 1], name="train_labels")
140
141         # Embeddings
            ------------------------------------------------------------------------------------
142
143         D2V_EMB_VRB = tf.Variable(D2V_EMB, trainable=False, name="doc_embeddings")
144
145         O2 = tf.Variable(tf.truncated_normal([song_size, song_emb_size], mean=0.0, stddev=1.0
                / math.sqrt(song_emb_size)), name="O2")
146         BO2 = tf.Variable(tf.zeros([song_emb_size]), name="BO2")
147
148         # Operaciones
            ------------------------------------------------------------------------------------
149
150         # Embedding de documento
151         ed = tf.nn.embedding_lookup(D2V_EMB_VRB, train_dataset[:, 0])
152
153         # Embedding de cancion
154         ec = tf.nn.embedding_lookup(O2, train_dataset[:, 1])
155         ec = ec + BO2
156
157         # Clculo de LOSS y optimizador
            ------------------------------------------------------------------------------------
158
159         embed = []
160         embed.append(ed)
161         embed.append(ec)
162         embed = tf.concat(embed, 1)
```

```
163
164         # softmax weights, W and D vectors should be concatenated before applying softmax
165         weights = tf.Variable(tf.truncated_normal([song_size, profile_size], stddev=1.0 /
                math.sqrt(song_size)),
166                               name="weights")
167         # softmax biases
168         biases = tf.Variable(tf.zeros([song_size]), name="biases")
169
170         # aClculo de LOSS y optimizador—
                ------------------------------------------------------------------------------
171         nce_loss = tf.nn.nce_loss(weights=weights,
172                                   biases=biases,
173                                   labels=train_labels,
174                                   inputs=embed,
175                                   num_sampled=num_sampled,
176                                   num_classes=song_size,
177                                   name="nce_loss")
178
179         batch_loss = nce_loss
180         nce_loss = tf.reduce_mean(nce_loss)
181
182         logits = tf.matmul(embed, tf.transpose(weights))
183         logits = tf.nn.bias_add(logits, biases)
184         # logits = tf.nn.softmax(logits,axis=0)
185         top_v,top_i = tf.nn.top_k(logits,k=song_size)
186
187         # optimizer =
                tf.train.AdamOptimizer(learning_rate).minimize(nce_loss,global_step=global_step)
188         train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(nce_loss,
                global_step=global_step)
189
190         # Inicializar variables
191         init = tf.global_variables_initializer()
192
193         # Crear objeto encargado de almacenar la red
194         saver = tf.train.Saver(max_to_keep=1)
195
196     config = tf.ConfigProto()
197     config.gpu_options.allow_growth = True
198     with tf.Session(graph=graph, config=config) as session:
199     #with tf.Session(graph=graph) as session:
200
201         # We must initialize all variables before we use them.
202         init.run()
203
204
205         if(save):
206             # Obtener el checkpoint
207             ckpt = tf.train.get_checkpoint_state(model_path)
208
209             # Si existe el checkpoint restaurar
210             if ckpt and ckpt.model_checkpoint_path:
211                 saver.restore(session, ckpt.model_checkpoint_path)
212
213         #Si se desea entrenar
214         if(train):
215             #-------------------------------------------------------------------------------
```

```
216             # TRAIN
217             #--------------------------------------------------------------------------------
218
219             ITRS= int(len(X)/batch_size)
220             RES_ITMS = len(X) - (ITRS*batch_size)
221             if(RES_ITMS>0):ITRS+=1 #Si no es divisin exacta, se aade una iteracin
222
223             for e in range(epochs):
224                 TRAIN_LOSS = 0.0
225
226                 print("Epoch "+str(e))
227
228                 for step in range(ITRS):
229
230                     batch_inputs, batch_context =
                            getBatchData(X,Y,batch_size,step,ITRS,RES_ITMS)
231
232                     feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
233
234                     _, loss_val, batch_lss, gs = session.run([train_step,
                            nce_loss,batch_loss,global_step], feed_dict=feed_dict)
235
236                     TRAIN_LOSS += np.sum(batch_lss)
237
238                 #print('Epoch '+str(e)+', last batch loss: '+str(loss_val))
239
240                 if (save): saver.save(session, complete_path, global_step=global_step)
241
242                 #--------------------------------------------------------------------------------
243                 # DEV
244                 #--------------------------------------------------------------------------------
245
246                 if(len(DX)>0):
247
248                     ITRS_TEST = int(len(DX) / batch_size)
249                     RES_ITMS_TEST = len(DX) - (ITRS_TEST * batch_size)
250                     TEST_LOSS = 0.0
251                     if (RES_ITMS_TEST > 0): ITRS_TEST += 1 # Si no es divisin exacta, se
                            aade una iteracin
252
253
254                     for step in range(ITRS_TEST):
255
256                         batch_inputs, batch_context =
                                getBatchData(DX,DY,batch_size,step,ITRS_TEST,RES_ITMS_TEST)
257
258                         feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
259                         loss_val,batch_lss = session.run([nce_loss,batch_loss],
                                feed_dict=feed_dict)
260
261                         TEST_LOSS+=np.sum(batch_lss)
262
263                     dev_hist.append(TEST_LOSS / (len(DX) * 1.0))
264                     train_hist.append(TRAIN_LOSS / (len(X) * 1.0))
265
266                 if (len(dev_hist) >= slope_size):
267                     # Calcular la pendiente
```

```
                print(getSlope(dev_hist[-slope_size:]))
                print(dev_hist[-slope_size:])

                if (getSlope(dev_hist[-slope_size:]) > -1e-5 or e==epochs-1):

                    def printArray(data,data2):
                        for i in range(len(data)):
                            a=data[i]
                            b=data2[i]
                            print(str(a).replace(".",",")+"\t"+str(b).replace(".",","))

                    print("-" * 50)
                    print(e)
                    printArray(dev_hist,train_hist)
                    return
    else:
        #
        # TEST FINAL
        #
        batch_size = 300

        IN_TOP_5 = 0
        IN_TOP_10 = 0
        IN_TOP_20 = 0
        IN_TOP_50 = 0
        IN_TOP_100 = 0
        IN_TOP_1000 = 0
        real_positions = []

        items = 0

        ITRS_TEST = int(len(TX) / batch_size)
        RES_ITMS_TEST = len(TX) - (ITRS_TEST * batch_size)
        if (RES_ITMS_TEST > 0): ITRS_TEST += 1  # Si no es @divisin exacta, se naade una
            @iteracin

        for step in range(ITRS_TEST):
            #tss = time.time()

            batch_inputs, batch_context = getBatchData(TX, TY, batch_size, step,
                ITRS_TEST, RES_ITMS_TEST)
            top_values,top_indx = session.run([top_v,top_i], feed_dict={train_dataset:
                batch_inputs, train_labels: batch_context})

            for j in range(len(batch_context)):
                real_position = int(np.where(top_indx[j, :] == batch_context[j, 0])[0])
                real_positions.append(real_position)

                IN_TOP_5 += 1 if real_position <= 5 else 0
                IN_TOP_10 += 1 if real_position <= 10 else 0
                IN_TOP_20 += 1 if real_position <= 20 else 0
                IN_TOP_50 += 1 if real_position <= 50 else 0
                IN_TOP_100 += 1 if real_position <= 100 else 0
                IN_TOP_1000 += 1 if real_position <= 1000 else 0

```

```
320                 items += len(batch_inputs)*1.0
321
322                 PTF = IN_TOP_5 / items
323                 PTT = IN_TOP_10 / items
324                 PTTW = IN_TOP_20 / items
325                 PTFT = IN_TOP_50 / items
326                 PTCIEN = IN_TOP_100 / items
327                 PTMIL = IN_TOP_1000 / items
328
329                 print(np.mean(real_positions), np.std(real_positions),
                        np.median(real_positions))
330                 print(str(PTF).replace(".", ",") + "\t" + str(PTT).replace(".", ",") + "\t"
331                       + str(PTTW).replace(".",",") + "\t" + str(PTFT).replace(".", ",") + "\t"
332                       + str(PTCIEN).replace(".", ",") + "\t" + str(PTMIL).replace(".",","))
333
334                 #print(time.time()-tss)
335
336             print(np.mean(real_positions), np.std(real_positions), np.median(real_positions))
337
338         #-----------------------------------------------------------------------------------------------------------
339
340 ##############################################################################################################
341 # Llamadas
342 ##############################################################################################################
343
344 TEST_NAME="test_072"
345
346 os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
347
348 #Se obtinenen cada uno de los embeddings aprendidos (incluidas las canciones)
349 D2V_DM_PRESENT   = np.load('embeddings/d2v_dm/present/EMB_MATRIX')
350 D2V_DBOW_PRESENT  = np.load('embeddings/d2v_dbow/present/EMB_MATRIX')
351 W2V  = np.load('embeddings/w2v/EMB_MATRIX')
352
353 #Se concatenan los embeddings de los usuarios para generar el perfil
354 D2V_EMB = np.concatenate((D2V_DM_PRESENT,D2V_DBOW_PRESENT),axis=1) # El reciente = Reciente
        DM + Reciente DBOW
355
356 #Se cargan los datos de train de las canciones para conocer los IDs reales
357 DATA = np.load("datos/Extend/ARRAY_ARRAYS_TRAIN.pkl")
358
359 #Obtener una lista con los ID de canciones que no aestn el la matriz de embeddings
360 ALL = set(list(range(W2V.shape[0])))
361 REAL = set(list(set(list(map(int, DATA)))))
362 NOT_EXISTS = list(ALL.difference(REAL))
363
364 ##############################################################################################################
365 # TENSORFLOW
366 ##############################################################################################################
367
368 #-----------------------------------------------------------------------------------------------------------
369 # Grid Search + Early Stopping
370 #-----------------------------------------------------------------------------------------------------------
371 '''
372 X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))
373 rates = [0.000001,1, 0.01, 0.0001]
374
```

```
for r in rates:
    config = {"batch_size": 1024, "learning_rate": r, "epochs": 1000, "seed": 100}

    print("Learning Rate: "+str(r))
    print("-" * 50)
    trainNet(X, Y, DX, DY, TX, TY, config, user_size, train=True, save=False)
    print("-"*50)

exit()
'''
#-------------------------------------------------------------------------------
# Train
#-------------------------------------------------------------------------------
'''
X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))

#Para esta fase, se une TRAIN y DEV
X = np.concatenate((X,DX),axis=0)
Y = np.concatenate((Y,DY),axis=0)

DX=[]
DY=[]

config = {"batch_size": 1024, "learning_rate": 0.01, "epochs":1000, "seed":100}
trainNet(X, Y, DX, DY, TX, TY, config, user_size, train=True, save=True)

exit()
'''
#-------------------------------------------------------------------------------
# Test
#-------------------------------------------------------------------------------

X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))

config = {"batch_size": 1024, "learning_rate": 0.01, "epochs":1000, "seed":100}
trainNet(X, Y, DX, DY, TX, TY, config, user_size,train=False, save=True)
```

4.16. TEST 16

```
# —*— coding: utf—8 —*—

import os.path
import os
import tensorflow as tf
import numpy as np
import random
import time
import math
import pandas as pd
import pickle
from collections import Counter
from scipy.stats import linregress

#########################################################################################################################
# eMtodos
#########################################################################################################################

def getData(REAL_SONG_IDS=[],data ="datos/Append/ARRAY_ARRAYS_REC.pkl",shuffle=True,
    seed=100, train=.7):

    DATA = np.load(data)

    #DATA = DATA[:20];print("ELIMINAR FILTRO")

    #User, cancion 1, cancion 2
    TRAIN = np.empty([0, 3], dtype=np.int32)
    DEV = np.empty([0, 3], dtype=np.int32)
    TEST = np.empty([0, 3], dtype=np.int32)

    for u in range(len(DATA)):

        # Eliminar canciones escuchadas que no tienen embedding
        USER_PLAYS = DATA[u]
        USER_PLAYS = pd.DataFrame(list(map(int, USER_PLAYS)))
        USER_PLAYS = USER_PLAYS.loc[USER_PLAYS[0].isin(REAL_SONG_IDS)][0].values

        #Separar en train dev test
        TRAIN_ITEMS = int(len(USER_PLAYS)*train)
        DEV_ITEMS = int((len(USER_PLAYS)—TRAIN_ITEMS)/2)

        TRAIN_DATA = USER_PLAYS[:TRAIN_ITEMS]
        DEV_DATA = USER_PLAYS[TRAIN_ITEMS:TRAIN_ITEMS+DEV_ITEMS]
        TEST_DATA = USER_PLAYS[TRAIN_ITEMS+DEV_ITEMS:]

        # Crear ejemplos
        USR_TRAIN = np.empty([len(TRAIN_DATA)—1, 3], dtype=np.int32)
        USR_TRAIN[:,0] = u
        USR_TRAIN[:,1] = TRAIN_DATA[:len(TRAIN_DATA)—1]
        USR_TRAIN[:,2] = TRAIN_DATA[1:]

        USR_DEV = np.empty([len(DEV_DATA) — 1, 3], dtype=np.int32)
        USR_DEV[:, 0] = u
        USR_DEV[:, 1] = DEV_DATA[:len(DEV_DATA) — 1]
        USR_DEV[:, 2] = DEV_DATA[1:]
```

```
55
56          USR_TEST = np.empty([len(TEST_DATA) - 1, 3], dtype=np.int32)
57          USR_TEST[:, 0] = u
58          USR_TEST[:, 1] = TEST_DATA[:len(TEST_DATA) - 1]
59          USR_TEST[:, 2] = TEST_DATA[1:]
60
61
62          # nAadir al global
63          TRAIN = np.concatenate([TRAIN,USR_TRAIN])
64          DEV = np.concatenate([DEV,USR_DEV])
65          TEST = np.concatenate([TEST,USR_TEST])
66
67
68      #Mezclar
69      if(shuffle):
70          np.random.seed(seed)
71          np.random.shuffle(TRAIN)
72          np.random.shuffle(DEV)
73          np.random.shuffle(TEST)
74
75
76      return
            TRAIN[:,:2],np.matrix(TRAIN[:,-1]).T,DEV[:,:2],np.matrix(DEV[:,-1]).T,TEST[:,:2],np.matrix(TEST
77
78  def getBatchData(X, Y, batch_size, step, ITRS, RES_ITMS):
79
80      # Si se noaadi una oiteracin, introducir los datos restantes
81      if (step == ITRS - 1 and RES_ITMS > 0):
82          batch_inputs = X[(ITRS - 1) * batch_size:]
83          batch_context = Y[(ITRS - 1) * batch_size:]
84      # Si no, es un batch normal
85      else:
86          st = step * batch_size
87          nd = (step + 1) * batch_size
88
89          batch_inputs = X[st:nd]
90          batch_context = Y[st:nd]
91      return(batch_inputs,batch_context)
92
93  def trainNet(X,Y, DX,DY,TX,TY,config, user_size, train=True, save=True):
94
95      def getSlope(data):
96          r = linregress(range(len(data)), data)
97          return r.slope
98
99      batch_size = config['batch_size']
100     learning_rate = config['learning_rate']
101     epochs = config['epochs']
102
103     user_size = user_size
104     song_size = W2V.shape[0]
105
106     user_emb_size = 128
107     user_concat_size = user_emb_size * 2
108     song_emb_size = 64
109     hidden_size = 32
110
```

```
111     profile_size = user_emb_size + song_emb_size
112
113     num_sampled = 64  # Ejemplos negativos (Para el NSE)
114
115     model_name = 'model'
116     model_path = 'models/' + TEST_NAME + '/model'
117     complete_path = model_path + "/" + model_name
118
119     # Se almacena la loss de dev de las 5 ultimas epochs
120     slope_size = 100
121     train_hist = []
122     dev_hist = []
123
124     # Creacin del grafo de TF.
125     graph = tf.Graph()
126
127     with graph.as_default():
128
129         if(config["seed"]!=1):
130             tf.set_random_seed(config["seed"])
131
132         # Nmero global de iteraciones
133         global_step = tf.Variable(0, dtype=tf.int32, trainable=False, name='global_step')
134
135         # Datos de entrada
            ------------------------------------------------------------------------------------
136         # Array del tamao del batch con las X
137         train_dataset = tf.placeholder(tf.int32, shape=[None, 2], name="train_inputs")
138
139         # Array del tamao del batch con las Y asociadas
140         train_labels = tf.placeholder(tf.float32, shape=[None, 1], name="train_labels")
141
142         # Embeddings
            ------------------------------------------------------------------------------------
143
144         D2V_EMB_VRB = tf.Variable(D2V_EMB, trainable=False, name="doc_embeddings")
145
146         T0 = tf.Variable(tf.truncated_normal([user_concat_size, user_emb_size], mean=0.0,
                stddev=1.0 / math.sqrt(hidden_size)), name="T0")
147         B0 = tf.Variable(tf.zeros([user_emb_size]), name="B0")
148
149         O2 = tf.Variable(tf.truncated_normal([song_size, song_emb_size], mean=0.0, stddev=1.0
                / math.sqrt(song_emb_size)), name="O2")
150         BO2 = tf.Variable(tf.zeros([song_emb_size]), name="BO2")
151
152         # Operaciones
            ------------------------------------------------------------------------------------
153
154         # Embedding de documento
155         ed = tf.nn.embedding_lookup(D2V_EMB_VRB, train_dataset[:, 0])
156
157         #Transformar concat documento de 256 a 128
158         ed = tf.matmul(ed,T0)+B0
159
160         # Embedding de cancion
161         ec = tf.nn.embedding_lookup(O2, train_dataset[:, 1])
162         ec = ec + BO2
```

```
    # aClculo de LOSS y optimizador—
        ————————————————————————————————————————————————————————————————————————————————————————————————

    embed = []
    embed.append(ed)
    embed.append(ec)
    embed = tf.concat(embed, 1)

    # softmax weights, W and D vectors should be concatenated before applying softmax
    weights = tf.Variable(tf.truncated_normal([song_size, profile_size], stddev=1.0 /
        math.sqrt(song_size)),
                          name="weights")
    # softmax biases
    biases = tf.Variable(tf.zeros([song_size]), name="biases")

    # aClculo de LOSS y optimizador—
        ————————————————————————————————————————————————————————————————————————————————————————————————
    nce_loss = tf.nn.nce_loss(weights=weights,
                              biases=biases,
                              labels=train_labels,
                              inputs=embed,
                              num_sampled=num_sampled,
                              num_classes=song_size,
                              name="nce_loss")

    batch_loss = nce_loss
    nce_loss = tf.reduce_mean(nce_loss)

    logits = tf.matmul(embed, tf.transpose(weights))
    logits = tf.nn.bias_add(logits, biases)
    # logits = tf.nn.softmax(logits,axis=0)
    top_v,top_i = tf.nn.top_k(logits,k=song_size)

    # optimizer =
        tf.train.AdamOptimizer(learning_rate).minimize(nce_loss,global_step=global_step)
    train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(nce_loss,
        global_step=global_step)

    # Inicializar variables
    init = tf.global_variables_initializer()

    # Crear objeto encargado de almacenar la red
    saver = tf.train.Saver(max_to_keep=1)

config = tf.ConfigProto()
config.gpu_options.allow_growth = True
with tf.Session(graph=graph, config=config) as session:
#with tf.Session(graph=graph) as session:

    # We must initialize all variables before we use them.
    init.run()


    if(save):
        # Obtener el checkpoint
        ckpt = tf.train.get_checkpoint_state(model_path)
```

```
215
216             # Si existe el checkpoint restaurar
217             if ckpt and ckpt.model_checkpoint_path:
218                 saver.restore(session, ckpt.model_checkpoint_path)
219
220         #Si se desea entrenar
221         if(train):
222             #-----------------------------------------------------------------------------------------------------
223             # TRAIN
224             #-----------------------------------------------------------------------------------------------------
225
226             ITRS= int(len(X)/batch_size)
227             RES_ITMS = len(X) - (ITRS*batch_size)
228             if(RES_ITMS>0):ITRS+=1 #Si no es divisin exacta, se naade una iteracin
229
230             for e in range(epochs):
231                 TRAIN_LOSS = 0.0
232
233                 print("Epoch "+str(e))
234
235                 for step in range(ITRS):
236
237                     batch_inputs, batch_context =
                            getBatchData(X,Y,batch_size,step,ITRS,RES_ITMS)
238
239                     feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
240
241                     _, loss_val, batch_lss, gs = session.run([train_step,
                            nce_loss,batch_loss,global_step], feed_dict=feed_dict)
242
243                     TRAIN_LOSS += np.sum(batch_lss)
244
245                 #print('Epoch '+str(e)+', last batch loss: '+str(loss_val))
246
247                 if (save): saver.save(session, complete_path, global_step=global_step)
248
249                 #-----------------------------------------------------------------------------------------------------
250                 # DEV
251                 #-----------------------------------------------------------------------------------------------------
252
253                 if(len(DX)>0):
254
255                     ITRS_TEST = int(len(DX) / batch_size)
256                     RES_ITMS_TEST = len(DX) - (ITRS_TEST * batch_size)
257                     TEST_LOSS = 0.0
258                     if (RES_ITMS_TEST > 0): ITRS_TEST += 1 # Si no es divisin exacta, se
                            naade una iteracin
259
260
261                     for step in range(ITRS_TEST):
262
263                         batch_inputs, batch_context =
                                getBatchData(DX,DY,batch_size,step,ITRS_TEST,RES_ITMS_TEST)
264
265                         feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
266                         loss_val,batch_lss = session.run([nce_loss,batch_loss],
                                feed_dict=feed_dict)
```

```
267
268                     TEST_LOSS+=np.sum(batch_lss)
269
270                 dev_hist.append(TEST_LOSS / (len(DX) * 1.0))
271                 train_hist.append(TRAIN_LOSS / (len(X) * 1.0))
272
273             if (len(dev_hist) >= slope_size):
274                 # Calcular la pendiente
275                 print(getSlope(dev_hist[-slope_size:]))
276                 print(dev_hist[-slope_size:])
277
278                 if (getSlope(dev_hist[-slope_size:]) > -1e-5 or e==epochs-1):
279
280                     def printArray(data,data2):
281                         for i in range(len(data)):
282                             a=data[i]
283                             b=data2[i]
284                             print(str(a).replace(".",",")+"\t"+str(b).replace(".",","))
285
286                     print("-" * 50)
287                     print(e)
288                     printArray(dev_hist,train_hist)
289                     return
290         else:
291             #
                -----------------------------------------------------------------------------
292             # TEST FINAL
293             #
                -----------------------------------------------------------------------------
294             batch_size = 300
295
296             IN_TOP_5 = 0
297             IN_TOP_10 = 0
298             IN_TOP_20 = 0
299             IN_TOP_50 = 0
300             IN_TOP_100 = 0
301             IN_TOP_1000 = 0
302             real_positions = []
303
304             items = 0
305
306             ITRS_TEST = int(len(TX) / batch_size)
307             RES_ITMS_TEST = len(TX) - (ITRS_TEST * batch_size)
308             if (RES_ITMS_TEST > 0): ITRS_TEST += 1  # Si no es @divisin exacta, se ¤aade una
                @iteracin
309
310             for step in range(ITRS_TEST):
311                 #tss = time.time()
312
313                 batch_inputs, batch_context = getBatchData(TX, TY, batch_size, step,
                    ITRS_TEST, RES_ITMS_TEST)
314                 top_values,top_indx = session.run([top_v,top_i], feed_dict={train_dataset:
                    batch_inputs, train_labels: batch_context})
315
316                 for j in range(len(batch_context)):
317                     real_position = int(np.where(top_indx[j, :] == batch_context[j, 0])[0])
318                     real_positions.append(real_position)
```

```
319
320                     IN_TOP_5 += 1 if real_position <= 5 else 0
321                     IN_TOP_10 += 1 if real_position <= 10 else 0
322                     IN_TOP_20 += 1 if real_position <= 20 else 0
323                     IN_TOP_50 += 1 if real_position <= 50 else 0
324                     IN_TOP_100 += 1 if real_position <= 100 else 0
325                     IN_TOP_1000 += 1 if real_position <= 1000 else 0
326
327                 items += len(batch_inputs)*1.0
328
329                 PTF = IN_TOP_5 / items
330                 PTT = IN_TOP_10 / items
331                 PTTW = IN_TOP_20 / items
332                 PTFT = IN_TOP_50 / items
333                 PTCIEN = IN_TOP_100 / items
334                 PTMIL = IN_TOP_1000 / items
335
336                 print(np.mean(real_positions), np.std(real_positions),
                        np.median(real_positions))
337                 print(str(PTF).replace(".", ",") + "\t" + str(PTT).replace(".", ",") + "\t"
338                       + str(PTTW).replace(".",",") + "\t" + str(PTFT).replace(".", ",") + "\t"
339                       + str(PTCIEN).replace(".", ",") + "\t" + str(PTMIL).replace(".",","))
340
341                 #print(time.time()-tss)
342
343             print(np.mean(real_positions), np.std(real_positions), np.median(real_positions))
344
345         #-----------------------------------------------------------------------------------------------------------
346
347 ##############################################################################################################
348 # Llamadas
349 ##############################################################################################################
350
351 TEST_NAME="test_082"
352
353 os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
354
355 #Se obtinenen cada uno de los embeddings aprendidos (incluidas las canciones)
356 D2V_DM_COMPLETE   = np.load('embeddings/d2v_dm/complete/EMB_MATRIX')
357 D2V_DBOW_COMPLETE = np.load('embeddings/d2v_dbow/complete/EMB_MATRIX')
358
359 #Se obtinenen cada uno de los embeddings aprendidos (incluidas las canciones)
360 D2V_DM_PRESENT  = np.load('embeddings/d2v_dm/present/EMB_MATRIX')
361 D2V_DBOW_PRESENT = np.load('embeddings/d2v_dbow/present/EMB_MATRIX')
362
363 W2V  = np.load('embeddings/w2v/EMB_MATRIX')
364
365 #Se concatenan los embeddings de los usuarios para generar el perfil
366 D2V_CONS = np.concatenate((D2V_DM_COMPLETE,D2V_DBOW_COMPLETE),axis=1) # El consolidado =
        Consolidado DM + Consolidado DBOW
367 D2V_PRES = np.concatenate((D2V_DM_PRESENT,D2V_DBOW_PRESENT),axis=1) # El presente = Presente
        DM + Presente DBOW
368
369 D2V_EMB = np.concatenate((D2V_CONS,D2V_PRES),axis=1) # El perfil completo CONS + PRES
370
371 #Se cargan los datos de train de las canciones para conocer los IDs reales
372 DATA = np.load("datos/Extend/ARRAY_ARRAYS_TRAIN.pkl")
```

```
373
374 #Obtener una lista con los ID de canciones que no aestn el la matriz de embeddings
375 ALL = set(list(range(W2V.shape[0])))
376 REAL = set(list(set(list(map(int, DATA)))))
377 NOT_EXISTS = list(ALL.difference(REAL))
378
379 ##########################################################################################################
380 # TENSORFLOW
381 ##########################################################################################################
382
383 #---------------------------------------------------------------------------------------------------------
384 # Grid Search + Early Stopping
385 #---------------------------------------------------------------------------------------------------------
386 '''
387 X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))
388 rates = [0.000001,1, 0.01, 0.0001]
389
390 for r in rates:
391     config = {"batch_size": 1024, "learning_rate": r, "epochs": 1000, "seed": 100}
392
393     print("Learning Rate: "+str(r))
394     print("-" * 50)
395     trainNet(X, Y, DX, DY, TX, TY, config, user_size, train=True, save=False)
396     print("-"*50)
397
398 exit()
399 '''
400 #---------------------------------------------------------------------------------------------------------
401 # Train
402 #---------------------------------------------------------------------------------------------------------
403 '''
404 X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))
405
406 #Para esta fase, se une TRAIN y DEV
407 X = np.concatenate((X,DX),axis=0)
408 Y = np.concatenate((Y,DY),axis=0)
409
410 DX=[]
411 DY=[]
412
413 config = {"batch_size": 1024, "learning_rate": 0.0001, "epochs":1000, "seed":100}
414 trainNet(X, Y, DX, DY, TX, TY, config, user_size, train=True, save=True)
415
416 exit()
417 '''
418 #---------------------------------------------------------------------------------------------------------
419 # Test
420 #---------------------------------------------------------------------------------------------------------
421
422 X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))
423
424 config = {"batch_size": 1024, "learning_rate": 0.0001, "epochs":1000, "seed":100}
425 trainNet(X, Y, DX, DY, TX, TY, config, user_size,train=False, save=True)
```

4.17. TEST 17

```
# -*- coding: utf-8 -*-

import os.path
import os
import tensorflow as tf
import numpy as np
import random
import time
import math
import pandas as pd
import pickle
from collections import Counter
from scipy.stats import linregress
from tensorflow.python.framework import ops as framework_ops

########################################################################################################################
# @Mtodos
########################################################################################################################

def getData(REAL_SONG_IDS=[],data ="datos/Append/ARRAY_ARRAYS_REC.pkl",shuffle=True,
    seed=100, train=.7):

    DATA = np.load(data)

    #DATA = DATA[:5];print("ELIMINAR FILTRO")

    #User, cancion 1, cancion 2, cancion 3
    TRAIN = np.empty([0, 4], dtype=np.int32)
    DEV = np.empty([0, 4], dtype=np.int32)
    TEST = np.empty([0, 4], dtype=np.int32)

    for u in range(len(DATA)):

        # Eliminar canciones escuchadas que no tienen embedding
        USER_PLAYS = DATA[u]
        USER_PLAYS = pd.DataFrame(list(map(int, USER_PLAYS)))
        USER_PLAYS = USER_PLAYS.loc[USER_PLAYS[0].isin(REAL_SONG_IDS)][0].values

        #Separar en train dev test
        TRAIN_ITEMS = int(len(USER_PLAYS)*train)
        DEV_ITEMS = int((len(USER_PLAYS)-TRAIN_ITEMS)/2)

        TRAIN_DATA = USER_PLAYS[:TRAIN_ITEMS]
        DEV_DATA = USER_PLAYS[TRAIN_ITEMS:TRAIN_ITEMS+DEV_ITEMS]
        TEST_DATA = USER_PLAYS[TRAIN_ITEMS+DEV_ITEMS:]

        # Crear ejemplos
        USR_TRAIN = np.empty([len(TRAIN_DATA)-2, 4], dtype=np.int32)
        USR_TRAIN[:,0] = u
        USR_TRAIN[:,1] = TRAIN_DATA[:len(TRAIN_DATA)-2]
        USR_TRAIN[:,2] = TRAIN_DATA[1:-1]
        USR_TRAIN[:,3] = TRAIN_DATA[2:]

        USR_DEV = np.empty([len(DEV_DATA) -2, 4], dtype=np.int32)
        USR_DEV[:, 0] = u
```

```
55          USR_DEV[:, 1] = DEV_DATA[:len(DEV_DATA)-2]
56          USR_DEV[:, 2] = DEV_DATA[1:-1]
57          USR_DEV[:, 3] = DEV_DATA[2:]
58
59          USR_TEST = np.empty([len(TEST_DATA)-2, 4], dtype=np.int32)
60          USR_TEST[:, 0] = u
61          USR_TEST[:, 1] = TEST_DATA[:len(TEST_DATA) -2]
62          USR_TEST[:, 2] = TEST_DATA[1:-1]
63          USR_TEST[:, 3] = TEST_DATA[2:]
64
65          # nAadir al global
66          TRAIN = np.concatenate([TRAIN,USR_TRAIN])
67          DEV = np.concatenate([DEV,USR_DEV])
68          TEST = np.concatenate([TEST,USR_TEST])
69
70      #Mezclar
71      if(shuffle):
72          np.random.seed(seed)
73          np.random.shuffle(TRAIN)
74          np.random.shuffle(DEV)
75          np.random.shuffle(TEST)
76
77      return
            TRAIN[:,:3],np.matrix(TRAIN[:,-1]).T,DEV[:,:3],np.matrix(DEV[:,-1]).T,TEST[:,:3],np.matrix(TEST
78
79  def getBatchData(X, Y, batch_size, step, ITRS, RES_ITMS):
80
81      # Si se no aadi una iteracin, introducir los datos restantes
82      if (step == ITRS - 1 and RES_ITMS > 0):
83          batch_inputs = X[(ITRS - 1) * batch_size:]
84          batch_context = Y[(ITRS - 1) * batch_size:]
85      # Si no, es un batch normal
86      else:
87          st = step * batch_size
88          nd = (step + 1) * batch_size
89
90          batch_inputs = X[st:nd]
91          batch_context = Y[st:nd]
92      return(batch_inputs,batch_context)
93
94  def trainNet(X,Y, DX,DY,TX,TY,config, user_size, train=True, save=True):
95
96      def getSlope(data):
97          r = linregress(range(len(data)), data)
98          return r.slope
99
100     batch_size = config['batch_size']
101     learning_rate = config['learning_rate']
102     epochs = config['epochs']
103
104     user_size = user_size
105     song_size = W2V.shape[0]
106
107     user_emb_size = 128
108     song_emb_size = 64
109     hidden_size = 32
110
```

```
111     profile_size = user_emb_size + song_emb_size + song_emb_size
112
113     num_sampled = 64  # Ejemplos negativos (Para el NSE)
114
115     model_name = 'model'
116     model_path = 'models/' + TEST_NAME + '/model'
117     complete_path = model_path + "/" + model_name
118
119     # Se almacena la loss de dev de las 5 ultimas epochs
120     slope_size = 100
121     train_hist = []
122     dev_hist = []
123
124     # Creacin del grafo de TF.
125     graph = tf.Graph()
126
127     with graph.as_default():
128
129         if(config["seed"]!=1):
130             tf.set_random_seed(config["seed"])
131
132         # Nmero global de iteraciones
133         global_step = tf.Variable(0, dtype=tf.int32, trainable=False, name='global_step')
134
135         # Datos de entrada
            ------------------------------------------------------------------------------------
136         # Array del tamao del batch con las X
137         train_dataset = tf.placeholder(tf.int32, shape=[None, 3], name="train_inputs")
138
139         # Array del tamao del batch con las Y asociadas
140         train_labels = tf.placeholder(tf.float32, shape=[None, 1], name="train_labels")
141
142         # Embeddings
            ------------------------------------------------------------------------------------
143
144         O1 = tf.Variable(tf.truncated_normal([user_size, user_emb_size], mean=0.0, stddev=1.0
                / math.sqrt(user_emb_size)), name="O1")
145         BO1 = tf.Variable(tf.zeros([user_emb_size]), name="BO1")
146
147         W2V_EMB_VRB = tf.Variable(W2V, trainable=False, name="word_embeddings")
148
149         # Operaciones
            ------------------------------------------------------------------------------------
150
151         # Embedding de documento
152         ed = tf.nn.embedding_lookup(O1, train_dataset[:, 0])
153         ed = ed + BO1
154
155         # Embedding de cancion 1
156         ec1 = tf.nn.embedding_lookup(W2V_EMB_VRB, train_dataset[:, 1])
157
158         # Embedding de cancion 2
159         ec2 = tf.nn.embedding_lookup(W2V_EMB_VRB, train_dataset[:, 2])
160
161         # Clculo de LOSS y optimizador
            ------------------------------------------------------------------------------------
162
```

```
163         embed = []
164         embed.append(ed)
165         embed.append(ec1)
166         embed.append(ec2)
167         embed = tf.concat(embed, 1)
168 
169         # softmax weights, W and D vectors should be concatenated before applying softmax
170         weights = tf.Variable(tf.truncated_normal([song_size, profile_size], stddev=1.0 /
                math.sqrt(song_size)), name="weights")
171         # softmax biases
172         biases = tf.Variable(tf.zeros([song_size]), name="biases")
173 
174         # aClculo de LOSS y optimizador—
                ------------------------------------------------------------------------------
175         nce_loss = tf.nn.nce_loss(weights=weights,
176                                   biases=biases,
177                                   labels=train_labels,
178                                   inputs=embed,
179                                   num_sampled=num_sampled,
180                                   num_classes=song_size,
181                                   name="nce_loss")
182 
183         batch_loss = nce_loss
184         nce_loss = tf.reduce_mean(nce_loss)
185 
186         logits = tf.matmul(embed, tf.transpose(weights))
187         logits = tf.nn.bias_add(logits, biases)
188 
189         top_v,top_i = tf.nn.top_k(logits,k=song_size)
190 
191         # logits = tf.nn.softmax(logits,axis=0)
192 
193         # optimizer =
                tf.train.AdamOptimizer(learning_rate).minimize(nce_loss,global_step=global_step)
194         train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(nce_loss,
                global_step=global_step)
195 
196         # Inicializar variables
197         init = tf.global_variables_initializer()
198 
199         # Crear objeto encargado de almacenar la red
200         saver = tf.train.Saver(max_to_keep=1)
201 
202     config = tf.ConfigProto()
203     config.gpu_options.allow_growth = True
204     with tf.Session(graph=graph, config=config) as session:
205     #with tf.Session(graph=graph) as session:
206 
207         # We must initialize all variables before we use them.
208         init.run()
209 
210 
211         if(save):
212             # Obtener el checkpoint
213             ckpt = tf.train.get_checkpoint_state(model_path)
214 
215             # Si existe el checkpoint restaurar
```

```
216             if ckpt and ckpt.model_checkpoint_path:
217                 saver.restore(session, ckpt.model_checkpoint_path)
218 
219         #Si se desea entrenar
220         if(train):
221             #-----------------------------------------------------------------------------------------------------------
222             # TRAIN
223             #-----------------------------------------------------------------------------------------------------------
224 
225             ITRS= int(len(X)/batch_size)
226             RES_ITMS = len(X) - (ITRS*batch_size)
227             if(RES_ITMS>0):ITRS+=1 #Si no es divisin exacta, se naade una iteracin
228 
229             for e in range(epochs):
230                 TRAIN_LOSS = 0.0
231 
232                 print("Epoch "+str(e))
233 
234                 for step in range(ITRS):
235 
236                     batch_inputs, batch_context =
                            getBatchData(X,Y,batch_size,step,ITRS,RES_ITMS)
237 
238                     feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
239 
240                     _, loss_val, batch_lss, gs = session.run([train_step,
                            nce_loss,batch_loss,global_step], feed_dict=feed_dict)
241 
242                     TRAIN_LOSS += np.sum(batch_lss)
243 
244                 #print('Epoch '+str(e)+', last batch loss: '+str(loss_val))
245 
246                 if (save): saver.save(session, complete_path, global_step=global_step)
247 
248                 #-----------------------------------------------------------------------------------------------------------
249                 # DEV
250                 #-----------------------------------------------------------------------------------------------------------
251 
252                 if(len(DX)>0):
253 
254                     ITRS_TEST = int(len(DX) / batch_size)
255                     RES_ITMS_TEST = len(DX) - (ITRS_TEST * batch_size)
256                     TEST_LOSS = 0.0
257                     if (RES_ITMS_TEST > 0): ITRS_TEST += 1 # Si no es divisin exacta, se
                            naade una iteracin
258 
259 
260                     for step in range(ITRS_TEST):
261 
262                         batch_inputs, batch_context =
                                getBatchData(DX,DY,batch_size,step,ITRS_TEST,RES_ITMS_TEST)
263 
264                         feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
265                         loss_val,batch_lss = session.run([nce_loss,batch_loss],
                                feed_dict=feed_dict)
266 
267                         TEST_LOSS+=np.sum(batch_lss)
```

```
268
269                 dev_hist.append(TEST_LOSS / (len(DX) * 1.0))
270                 train_hist.append(TRAIN_LOSS / (len(X) *  1.0))
271
272             if (len(dev_hist) >= slope_size):
273                 # Calcular la pendiente
274                 print(getSlope(dev_hist[-slope_size:]))
275                 print(dev_hist[-slope_size:])
276
277                 if (getSlope(dev_hist[-slope_size:]) > -1e-5 or e==epochs-1):
278
279                     def printArray(data,data2):
280                         for i in range(len(data)):
281                             a=data[i]
282                             b=data2[i]
283                             print(str(a).replace(".",",")+"\t"+str(b).replace(".",","))
284
285                     print("-" * 50)
286                     print(e)
287                     printArray(dev_hist,train_hist)
288                     return
289         else:
290             #
                -----------------------------------------------------------------------------
291             # TEST FINAL
292             #
                -----------------------------------------------------------------------------
293             batch_size = 300
294
295             IN_TOP_5 = 0
296             IN_TOP_10 = 0
297             IN_TOP_20 = 0
298             IN_TOP_50 = 0
299             IN_TOP_100 = 0
300             IN_TOP_1000 = 0
301             real_positions = []
302
303             items = 0
304
305             ITRS_TEST = int(len(TX) / batch_size)
306             RES_ITMS_TEST = len(TX) - (ITRS_TEST * batch_size)
307             if (RES_ITMS_TEST > 0): ITRS_TEST += 1  # Si no es divisin exacta, se aade una
                iteracin
308
309             for step in range(ITRS_TEST):
310                 #tss = time.time()
311
312                 batch_inputs, batch_context = getBatchData(TX, TY, batch_size, step,
                        ITRS_TEST, RES_ITMS_TEST)
313                 top_values,top_indx = session.run([top_v,top_i], feed_dict={train_dataset:
                        batch_inputs, train_labels: batch_context})
314
315                 for j in range(len(batch_context)):
316                     real_position = int(np.where(top_indx[j, :] == batch_context[j, 0])[0])
317                     real_positions.append(real_position)
318
319                     IN_TOP_5 += 1 if real_position <= 5 else 0
```

```
320                     IN_TOP_10 += 1 if real_position <= 10 else 0
321                     IN_TOP_20 += 1 if real_position <= 20 else 0
322                     IN_TOP_50 += 1 if real_position <= 50 else 0
323                     IN_TOP_100 += 1 if real_position <= 100 else 0
324                     IN_TOP_1000 += 1 if real_position <= 1000 else 0
325
326                 items += len(batch_inputs)*1.0
327
328                 PTF = IN_TOP_5 / items
329                 PTT = IN_TOP_10 / items
330                 PTTW = IN_TOP_20 / items
331                 PTFT = IN_TOP_50 / items
332                 PTCIEN = IN_TOP_100 / items
333                 PTMIL = IN_TOP_1000 / items
334
335                 print(np.mean(real_positions), np.std(real_positions),
                        np.median(real_positions))
336                 print(str(PTF).replace(".", ",") + "\t" + str(PTT).replace(".", ",") + "\t"
337                       + str(PTTW).replace(".",",") + "\t" + str(PTFT).replace(".", ",") + "\t"
338                       + str(PTCIEN).replace(".", ",") + "\t" + str(PTMIL).replace(".",","))
339
340                 #print(time.time()-tss)
341
342
343             print(np.mean(real_positions), np.std(real_positions), np.median(real_positions))
344
345         #-------------------------------------------------------------------------------------------------------------
346
347 ##################################################################################################################
348 # Llamadas
349 ##################################################################################################################
350
351 TEST_NAME="test_023"
352
353 os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
354
355 #Solamente se carga para conocer el unmero de canciones
356 W2V  = np.load('embeddings/w2v/EMB_MATRIX')
357
358 #Se cargan los datos de train de las canciones para conocer los IDs reales
359 DATA = np.load("datos/Extend/ARRAY_ARRAYS_TRAIN.pkl")
360
361 #Obtener una lista con los ID de canciones que no aestn el la matriz de embeddings
362 ALL = set(list(range(W2V.shape[0])))
363 REAL = set(list(set(list(map(int, DATA)))))
364
365 ##################################################################################################################
366 # TENSORFLOW
367 ##################################################################################################################
368
369 #-------------------------------------------------------------------------------------------------------------
370 # Grid Search + Early Stopping
371 #-------------------------------------------------------------------------------------------------------------
372 '''
373 X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))
374 rates = [0.000001,1, 0.01, 0.0001]
375
```

```
376
377 for r in rates:
378     config = {"batch_size": 1024, "learning_rate": r, "epochs": 1000, "seed": 100}
379
380     print("Learning Rate: "+str(r))
381     print("-" * 50)
382     trainNet(X, Y, DX, DY, TX, TY, config, user_size, train=True, save=False)
383     print("-"*50)
384
385 exit()
386 '''
387 #-----------------------------------------------------------------------------------------------------------------
388 # Train
389 #-----------------------------------------------------------------------------------------------------------------
390 '''
391 X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))
392
393 #Para esta fase, se une TRAIN y DEV
394 X = np.concatenate((X,DX),axis=0)
395 Y = np.concatenate((Y,DY),axis=0)
396
397 DX=[]
398 DY=[]
399
400 config = {"batch_size": 1024, "learning_rate": 0.01, "epochs":1000, "seed":100}
401 trainNet(X, Y, DX, DY, TX, TY, config, user_size, train=True, save=True)
402
403 exit()
404 '''
405 #-----------------------------------------------------------------------------------------------------------------
406 # Test
407 #-----------------------------------------------------------------------------------------------------------------
408
409 X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))
410
411 config = {"batch_size": 1024, "learning_rate": 0.01, "epochs":1000, "seed":100}
412 trainNet(X, Y, DX, DY, TX, TY, config, user_size,train=False, save=True)
```

4.18. TEST 18

```
# -*- coding: utf-8 -*-

import os.path
import os
import tensorflow as tf
import numpy as np
import random
import time
import math
import pandas as pd
import pickle
from collections import Counter
from scipy.stats import linregress
from tensorflow.python.framework import ops as framework_ops

########################################################################################################################
# @Mtodos
########################################################################################################################

def getData(REAL_SONG_IDS=[],data ="datos/Append/ARRAY_ARRAYS_REC.pkl",shuffle=True, seed=100, train=.7):

    DATA = np.load(data)

    #DATA = DATA[:5];print("ELIMINAR FILTRO")

    #User, cancion 1, cancion 2, cancion 3
    TRAIN = np.empty([0, 4], dtype=np.int32)
    DEV = np.empty([0, 4], dtype=np.int32)
    TEST = np.empty([0, 4], dtype=np.int32)

    for u in range(len(DATA)):

        # Eliminar canciones escuchadas que no tienen embedding
        USER_PLAYS = DATA[u]
        USER_PLAYS = pd.DataFrame(list(map(int, USER_PLAYS)))
        USER_PLAYS = USER_PLAYS.loc[USER_PLAYS[0].isin(REAL_SONG_IDS)][0].values

        #Separar en train dev test
        TRAIN_ITEMS = int(len(USER_PLAYS)*train)
        DEV_ITEMS = int((len(USER_PLAYS)-TRAIN_ITEMS)/2)

        TRAIN_DATA = USER_PLAYS[:TRAIN_ITEMS]
        DEV_DATA = USER_PLAYS[TRAIN_ITEMS:TRAIN_ITEMS+DEV_ITEMS]
        TEST_DATA = USER_PLAYS[TRAIN_ITEMS+DEV_ITEMS:]

        # Crear ejemplos
        USR_TRAIN = np.empty([len(TRAIN_DATA)-2, 4], dtype=np.int32)
        USR_TRAIN[:,0] = u
        USR_TRAIN[:,1] = TRAIN_DATA[:len(TRAIN_DATA)-2]
        USR_TRAIN[:,2] = TRAIN_DATA[1:-1]
        USR_TRAIN[:,3] = TRAIN_DATA[2:]

        USR_DEV = np.empty([len(DEV_DATA) -2, 4], dtype=np.int32)
        USR_DEV[:, 0] = u
```

```
        USR_DEV[:, 1] = DEV_DATA[:len(DEV_DATA)-2]
        USR_DEV[:, 2] = DEV_DATA[1:-1]
        USR_DEV[:, 3] = DEV_DATA[2:]

        USR_TEST = np.empty([len(TEST_DATA)-2, 4], dtype=np.int32)
        USR_TEST[:, 0] = u
        USR_TEST[:, 1] = TEST_DATA[:len(TEST_DATA) -2]
        USR_TEST[:, 2] = TEST_DATA[1:-1]
        USR_TEST[:, 3] = TEST_DATA[2:]

        # nAadir al global
        TRAIN = np.concatenate([TRAIN,USR_TRAIN])
        DEV = np.concatenate([DEV,USR_DEV])
        TEST = np.concatenate([TEST,USR_TEST])

    #Mezclar
    if(shuffle):
        np.random.seed(seed)
        np.random.shuffle(TRAIN)
        np.random.shuffle(DEV)
        np.random.shuffle(TEST)

    return
        TRAIN[:,:3],np.matrix(TRAIN[:,-1]).T,DEV[:,:3],np.matrix(DEV[:,-1]).T,TEST[:,:3],np.matrix(TEST

def getBatchData(X, Y, batch_size, step, ITRS, RES_ITMS):

    # Si se no aadi una iteracin, introducir los datos restantes
    if (step == ITRS - 1 and RES_ITMS > 0):
        batch_inputs = X[(ITRS - 1) * batch_size:]
        batch_context = Y[(ITRS - 1) * batch_size:]
    # Si no, es un batch normal
    else:
        st = step * batch_size
        nd = (step + 1) * batch_size

        batch_inputs = X[st:nd]
        batch_context = Y[st:nd]
    return(batch_inputs,batch_context)

def trainNet(X,Y, DX,DY,TX,TY,config, user_size, train=True, save=True):

    def getSlope(data):
        r = linregress(range(len(data)), data)
        return r.slope

    batch_size = config['batch_size']
    learning_rate = config['learning_rate']
    epochs = config['epochs']

    user_size = user_size
    song_size = W2V.shape[0]

    user_emb_size = 128
    song_emb_size = 64
    hidden_size = 32

```

```
111     profile_size = user_emb_size + song_emb_size + song_emb_size
112
113     num_sampled = 64  # Ejemplos negativos (Para el NSE)
114
115     model_name = 'model'
116     model_path = 'models/' + TEST_NAME + '/model'
117     complete_path = model_path + "/" + model_name
118
119     # Se almacena la loss de dev de las 5 ultimas epochs
120     slope_size = 100
121     train_hist = []
122     dev_hist = []
123
124     # Creacin del grafo de TF.
125     graph = tf.Graph()
126
127     with graph.as_default():
128
129         if(config["seed"]!=1):
130             tf.set_random_seed(config["seed"])
131
132         # Nmero global de iteraciones
133         global_step = tf.Variable(0, dtype=tf.int32, trainable=False, name='global_step')
134
135         # Datos de entrada
            ------------------------------------------------------------------------------------------
136         # Array del tamao del batch con las X
137         train_dataset = tf.placeholder(tf.int32, shape=[None, 3], name="train_inputs")
138
139         # Array del tamao del batch con las Y asociadas
140         train_labels = tf.placeholder(tf.float32, shape=[None, 1], name="train_labels")
141
142         # Embeddings
            ------------------------------------------------------------------------------------------
143
144         D2V_EMB_VRB = tf.Variable(D2V_EMB, trainable=False, name="doc_embeddings")
145
146         W2V_EMB_VRB = tf.Variable(W2V, trainable=False, name="word_embeddings")
147
148         # Operaciones
            ------------------------------------------------------------------------------------------
149
150         # Embedding de documento
151         ed = tf.nn.embedding_lookup(D2V_EMB_VRB, train_dataset[:, 0])
152
153         # Embedding de cancion 1
154         ec1 = tf.nn.embedding_lookup(W2V_EMB_VRB, train_dataset[:, 1])
155
156         # Embedding de cancion 2
157         ec2 = tf.nn.embedding_lookup(W2V_EMB_VRB, train_dataset[:, 2])
158
159         # Clculo de LOSS y optimizador
            ------------------------------------------------------------------------------------------
160
161         embed = []
162         embed.append(ed)
163         embed.append(ec1)
```

```
164         embed.append(ec2)
165         embed = tf.concat(embed, 1)
166 
167         # softmax weights, W and D vectors should be concatenated before applying softmax
168         weights = tf.Variable(tf.truncated_normal([song_size, profile_size], stddev=1.0 /
                math.sqrt(song_size)), name="weights")
169         # softmax biases
170         biases = tf.Variable(tf.zeros([song_size]), name="biases")
171 
172         # aClculo de LOSS y optimizador—
                ------------------------------------------------------------------------------
173         nce_loss = tf.nn.nce_loss(weights=weights,
174                                   biases=biases,
175                                   labels=train_labels,
176                                   inputs=embed,
177                                   num_sampled=num_sampled,
178                                   num_classes=song_size,
179                                   name="nce_loss")
180 
181         batch_loss = nce_loss
182         nce_loss = tf.reduce_mean(nce_loss)
183 
184         logits = tf.matmul(embed, tf.transpose(weights))
185         logits = tf.nn.bias_add(logits, biases)
186 
187         top_v,top_i = tf.nn.top_k(logits,k=song_size)
188 
189         # logits = tf.nn.softmax(logits,axis=0)
190 
191         # optimizer =
                tf.train.AdamOptimizer(learning_rate).minimize(nce_loss,global_step=global_step)
192         train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(nce_loss,
                global_step=global_step)
193 
194         # Inicializar variables
195         init = tf.global_variables_initializer()
196 
197         # Crear objeto encargado de almacenar la red
198         saver = tf.train.Saver(max_to_keep=1)
199 
200     config = tf.ConfigProto()
201     config.gpu_options.allow_growth = True
202     with tf.Session(graph=graph, config=config) as session:
203     #with tf.Session(graph=graph) as session:
204 
205         # We must initialize all variables before we use them.
206         init.run()
207 
208 
209         if(save):
210             # Obtener el checkpoint
211             ckpt = tf.train.get_checkpoint_state(model_path)
212 
213             # Si existe el checkpoint restaurar
214             if ckpt and ckpt.model_checkpoint_path:
215                 saver.restore(session, ckpt.model_checkpoint_path)
216 
```

```
217         #Si se desea entrenar
218         if(train):
219             #-----------------------------------------------------------------------------------------------
220             # TRAIN
221             #-----------------------------------------------------------------------------------------------
222 
223             ITRS= int(len(X)/batch_size)
224             RES_ITMS = len(X) - (ITRS*batch_size)
225             if(RES_ITMS>0):ITRS+=1 #Si no es divisin exacta, se aade una iteracin
226 
227             for e in range(epochs):
228                 TRAIN_LOSS = 0.0
229 
230                 print("Epoch "+str(e))
231 
232                 for step in range(ITRS):
233 
234                     batch_inputs, batch_context =
                            getBatchData(X,Y,batch_size,step,ITRS,RES_ITMS)
235 
236                     feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
237 
238                     _, loss_val, batch_lss, gs = session.run([train_step,
                            nce_loss,batch_loss,global_step], feed_dict=feed_dict)
239 
240                     TRAIN_LOSS += np.sum(batch_lss)
241 
242                 #print('Epoch '+str(e)+', last batch loss: '+str(loss_val))
243 
244                 if (save): saver.save(session, complete_path, global_step=global_step)
245 
246                 #-----------------------------------------------------------------------------------------------
247                 # DEV
248                 #-----------------------------------------------------------------------------------------------
249 
250                 if(len(DX)>0):
251 
252                     ITRS_TEST = int(len(DX) / batch_size)
253                     RES_ITMS_TEST = len(DX) - (ITRS_TEST * batch_size)
254                     TEST_LOSS = 0.0
255                     if (RES_ITMS_TEST > 0): ITRS_TEST += 1 # Si no es divisin exacta, se
                            aade una iteracin
256 
257 
258                     for step in range(ITRS_TEST):
259 
260                         batch_inputs, batch_context =
                                getBatchData(DX,DY,batch_size,step,ITRS_TEST,RES_ITMS_TEST)
261 
262                         feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
263                         loss_val,batch_lss = session.run([nce_loss,batch_loss],
                                feed_dict=feed_dict)
264 
265                         TEST_LOSS+=np.sum(batch_lss)
266 
267                     dev_hist.append(TEST_LOSS / (len(DX) * 1.0))
268                     train_hist.append(TRAIN_LOSS / (len(X) *  1.0))
```

```
269
270             if (len(dev_hist) >= slope_size):
271                 # Calcular la pendiente
272                 print(getSlope(dev_hist[-slope_size:]))
273                 print(dev_hist[-slope_size:])
274
275                 if (getSlope(dev_hist[-slope_size:]) > -1e-5 or e==epochs-1):
276
277                     def printArray(data,data2):
278                         for i in range(len(data)):
279                             a=data[i]
280                             b=data2[i]
281                             print(str(a).replace(".",",")+"\t"+str(b).replace(".",","))
282
283                     print("-" * 50)
284                     print(e)
285                     printArray(dev_hist,train_hist)
286                     return
287         else:
288             #
                ------------------------------------------------------------------------------
289             # TEST FINAL
290             #
                ------------------------------------------------------------------------------
291             batch_size = 300
292
293             IN_TOP_5 = 0
294             IN_TOP_10 = 0
295             IN_TOP_20 = 0
296             IN_TOP_50 = 0
297             IN_TOP_100 = 0
298             IN_TOP_1000 = 0
299             real_positions = []
300
301             items = 0
302
303             ITRS_TEST = int(len(TX) / batch_size)
304             RES_ITMS_TEST = len(TX) - (ITRS_TEST * batch_size)
305             if (RES_ITMS_TEST > 0): ITRS_TEST += 1  # Si no es divisin exacta, se aade una
                    iteracin
306
307             for step in range(ITRS_TEST):
308                 #tss = time.time()
309
310                 batch_inputs, batch_context = getBatchData(TX, TY, batch_size, step,
                        ITRS_TEST, RES_ITMS_TEST)
311                 top_values,top_indx = session.run([top_v,top_i], feed_dict={train_dataset:
                        batch_inputs, train_labels: batch_context})
312
313                 for j in range(len(batch_context)):
314                     real_position = int(np.where(top_indx[j, :] == batch_context[j, 0])[0])
315                     real_positions.append(real_position)
316
317                     IN_TOP_5 += 1 if real_position <= 5 else 0
318                     IN_TOP_10 += 1 if real_position <= 10 else 0
319                     IN_TOP_20 += 1 if real_position <= 20 else 0
320                     IN_TOP_50 += 1 if real_position <= 50 else 0
```

```
321                     IN_TOP_100 += 1 if real_position <= 100 else 0
322                     IN_TOP_1000 += 1 if real_position <= 1000 else 0
323 
324                 items += len(batch_inputs)*1.0
325 
326                 PTF = IN_TOP_5 / items
327                 PTT = IN_TOP_10 / items
328                 PTTW = IN_TOP_20 / items
329                 PTFT = IN_TOP_50 / items
330                 PTCIEN = IN_TOP_100 / items
331                 PTMIL = IN_TOP_1000 / items
332 
333                 print(np.mean(real_positions), np.std(real_positions),
                        np.median(real_positions))
334                 print(str(PTF).replace(".", ",") + "\t" + str(PTT).replace(".", ",") + "\t"
335                       + str(PTTW).replace(".",",") + "\t" + str(PTFT).replace(".", ",") + "\t"
336                       + str(PTCIEN).replace(".", ",") + "\t" + str(PTMIL).replace(".",","))
337 
338                 #print(time.time()-tss)
339 
340 
341             print(np.mean(real_positions), np.std(real_positions), np.median(real_positions))
342 
343         #-------------------------------------------------------------------------------------------------------------
344 
345 ##########################################################################################################
346 # Llamadas
347 ##########################################################################################################
348 
349 TEST_NAME="test_033"
350 
351 os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
352 
353 #Se obtinenen cada uno de los embeddings aprendidos (incluidas las canciones)
354 D2V_DM_COMPLETE   = np.load('embeddings/d2v_dm/complete/EMB_MATRIX')
355 D2V_DBOW_COMPLETE = np.load('embeddings/d2v_dbow/complete/EMB_MATRIX')
356 W2V  = np.load('embeddings/w2v/EMB_MATRIX')
357 
358 #Se concatenan los embeddings de los usuarios para generar el perfil
359 D2V_EMB = np.concatenate((D2V_DM_COMPLETE,D2V_DBOW_COMPLETE),axis=1) # El consolidado =
      Consolidado DM + Consolidado DBOW
360 
361 #Se cargan los datos de train de las canciones para conocer los IDs reales
362 DATA = np.load("datos/Extend/ARRAY_ARRAYS_TRAIN.pkl")
363 
364 #Obtener una lista con los ID de canciones que no aestn el la matriz de embeddings
365 ALL = set(list(range(W2V.shape[0])))
366 REAL = set(list(set(list(map(int, DATA)))))
367 
368 ##########################################################################################################
369 # TENSORFLOW
370 ##########################################################################################################
371 
372 #-------------------------------------------------------------------------------------------------------------
373 # Grid Search + Early Stopping
374 #-------------------------------------------------------------------------------------------------------------
375 '''
```

```
X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))
rates = [0.000001,1, 0.01, 0.0001]

for r in rates:
    config = {"batch_size": 1024, "learning_rate": r, "epochs": 1000, "seed": 100}

    print("Learning Rate: "+str(r))
    print("-" * 50)
    trainNet(X, Y, DX, DY, TX, TY, config, user_size, train=True, save=False)
    print("-"*50)

exit()
'''
#-------------------------------------------------------------------------------
# Train
#-------------------------------------------------------------------------------
'''
X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))

#Para esta fase, se une TRAIN y DEV
X = np.concatenate((X,DX),axis=0)
Y = np.concatenate((Y,DY),axis=0)

DX=[]
DY=[]

config = {"batch_size": 1024, "learning_rate": 0.01, "epochs":1000, "seed":100}
trainNet(X, Y, DX, DY, TX, TY, config, user_size, train=True, save=True)

exit()
'''
#-------------------------------------------------------------------------------
# Test
#-------------------------------------------------------------------------------

X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))

config = {"batch_size": 1024, "learning_rate": 0.01, "epochs":1000, "seed":100}
trainNet(X, Y, DX, DY, TX, TY, config, user_size,train=False, save=True)
```

4.19. TEST 19

```
# -*- coding: utf-8 -*-

import os.path
import os
import tensorflow as tf
import numpy as np
import random
import time
import math
import pandas as pd
import pickle
from collections import Counter
from scipy.stats import linregress
from tensorflow.python.framework import ops as framework_ops

########################################################################################################################
# @Mtodos
########################################################################################################################

def getData(REAL_SONG_IDS=[],data ="datos/Append/ARRAY_ARRAYS_REC.pkl",shuffle=True,
    seed=100, train=.7):

    DATA = np.load(data)

    #DATA = DATA[:5];print("ELIMINAR FILTRO")

    #User, cancion 1, cancion 2, cancion 3
    TRAIN = np.empty([0, 4], dtype=np.int32)
    DEV = np.empty([0, 4], dtype=np.int32)
    TEST = np.empty([0, 4], dtype=np.int32)

    for u in range(len(DATA)):

        # Eliminar canciones escuchadas que no tienen embedding
        USER_PLAYS = DATA[u]
        USER_PLAYS = pd.DataFrame(list(map(int, USER_PLAYS)))
        USER_PLAYS = USER_PLAYS.loc[USER_PLAYS[0].isin(REAL_SONG_IDS)][0].values

        #Separar en train dev test
        TRAIN_ITEMS = int(len(USER_PLAYS)*train)
        DEV_ITEMS = int((len(USER_PLAYS)-TRAIN_ITEMS)/2)

        TRAIN_DATA = USER_PLAYS[:TRAIN_ITEMS]
        DEV_DATA = USER_PLAYS[TRAIN_ITEMS:TRAIN_ITEMS+DEV_ITEMS]
        TEST_DATA = USER_PLAYS[TRAIN_ITEMS+DEV_ITEMS:]

        # Crear ejemplos
        USR_TRAIN = np.empty([len(TRAIN_DATA)-2, 4], dtype=np.int32)
        USR_TRAIN[:,0] = u
        USR_TRAIN[:,1] = TRAIN_DATA[:len(TRAIN_DATA)-2]
        USR_TRAIN[:,2] = TRAIN_DATA[1:-1]
        USR_TRAIN[:,3] = TRAIN_DATA[2:]

        USR_DEV = np.empty([len(DEV_DATA) -2, 4], dtype=np.int32)
        USR_DEV[:, 0] = u
```

```
        USR_DEV[:, 1] = DEV_DATA[:len(DEV_DATA)-2]
        USR_DEV[:, 2] = DEV_DATA[1:-1]
        USR_DEV[:, 3] = DEV_DATA[2:]

        USR_TEST = np.empty([len(TEST_DATA)-2, 4], dtype=np.int32)
        USR_TEST[:, 0] = u
        USR_TEST[:, 1] = TEST_DATA[:len(TEST_DATA) -2]
        USR_TEST[:, 2] = TEST_DATA[1:-1]
        USR_TEST[:, 3] = TEST_DATA[2:]

        # nAadir al global
        TRAIN = np.concatenate([TRAIN,USR_TRAIN])
        DEV = np.concatenate([DEV,USR_DEV])
        TEST = np.concatenate([TEST,USR_TEST])

    #Mezclar
    if(shuffle):
        np.random.seed(seed)
        np.random.shuffle(TRAIN)
        np.random.shuffle(DEV)
        np.random.shuffle(TEST)

    return
        TRAIN[:,:3],np.matrix(TRAIN[:,-1]).T,DEV[:,:3],np.matrix(DEV[:,-1]).T,TEST[:,:3],np.matrix(TEST

def getBatchData(X, Y, batch_size, step, ITRS, RES_ITMS):

    # Si se no aadi una iteracin, introducir los datos restantes
    if (step == ITRS - 1 and RES_ITMS > 0):
        batch_inputs = X[(ITRS - 1) * batch_size:]
        batch_context = Y[(ITRS - 1) * batch_size:]
    # Si no, es un batch normal
    else:
        st = step * batch_size
        nd = (step + 1) * batch_size

        batch_inputs = X[st:nd]
        batch_context = Y[st:nd]
    return(batch_inputs,batch_context)

def trainNet(X,Y, DX,DY,TX,TY,config, user_size, train=True, save=True):

    def getSlope(data):
        r = linregress(range(len(data)), data)
        return r.slope

    batch_size = config['batch_size']
    learning_rate = config['learning_rate']
    epochs = config['epochs']

    user_size = user_size
    song_size = W2V.shape[0]

    user_emb_size = 128
    song_emb_size = 64
    hidden_size = 32

```

```
111     profile_size = user_emb_size + song_emb_size + song_emb_size
112
113     num_sampled = 64  # Ejemplos negativos (Para el NSE)
114
115     model_name = 'model'
116     model_path = 'models/' + TEST_NAME + '/model'
117     complete_path = model_path + "/" + model_name
118
119     # Se almacena la loss de dev de las 5 ultimas epochs
120     slope_size = 100
121     train_hist = []
122     dev_hist = []
123
124     # Creacin del grafo de TF.
125     graph = tf.Graph()
126
127     with graph.as_default():
128
129         if(config["seed"]!=1):
130             tf.set_random_seed(config["seed"])
131
132         # Nmero global de iteraciones
133         global_step = tf.Variable(0, dtype=tf.int32, trainable=False, name='global_step')
134
135         # Datos de entrada
            ------------------------------------------------------------------------------------------
136         # Array del tamao del batch con las X
137         train_dataset = tf.placeholder(tf.int32, shape=[None, 3], name="train_inputs")
138
139         # Array del tamao del batch con las Y asociadas
140         train_labels = tf.placeholder(tf.float32, shape=[None, 1], name="train_labels")
141
142         # Embeddings
            ------------------------------------------------------------------------------------------
143
144         D2V_EMB_VRB = tf.Variable(D2V_EMB, trainable=False, name="doc_embeddings")
145
146         W2V_EMB_VRB = tf.Variable(W2V, trainable=False, name="word_embeddings")
147
148         # Operaciones
            ------------------------------------------------------------------------------------------
149
150         # Embedding de documento
151         ed = tf.nn.embedding_lookup(D2V_EMB_VRB, train_dataset[:, 0])
152
153         # Embedding de cancion 1
154         ec1 = tf.nn.embedding_lookup(W2V_EMB_VRB, train_dataset[:, 1])
155
156         # Embedding de cancion 2
157         ec2 = tf.nn.embedding_lookup(W2V_EMB_VRB, train_dataset[:, 2])
158
159         # Clculo de LOSS y optimizador
            ------------------------------------------------------------------------------------------
160
161         embed = []
162         embed.append(ed)
163         embed.append(ec1)
```

```
164         embed.append(ec2)
165         embed = tf.concat(embed, 1)
166 
167         # softmax weights, W and D vectors should be concatenated before applying softmax
168         weights = tf.Variable(tf.truncated_normal([song_size, profile_size], stddev=1.0 /
                math.sqrt(song_size)), name="weights")
169         # softmax biases
170         biases = tf.Variable(tf.zeros([song_size]), name="biases")
171 
172         # aClculo de LOSS y optimizador—
                ------------------------------------------------------------------------------
173         nce_loss = tf.nn.nce_loss(weights=weights,
174                                   biases=biases,
175                                   labels=train_labels,
176                                   inputs=embed,
177                                   num_sampled=num_sampled,
178                                   num_classes=song_size,
179                                   name="nce_loss")
180 
181         batch_loss = nce_loss
182         nce_loss = tf.reduce_mean(nce_loss)
183 
184         logits = tf.matmul(embed, tf.transpose(weights))
185         logits = tf.nn.bias_add(logits, biases)
186 
187         top_v,top_i = tf.nn.top_k(logits,k=song_size)
188 
189         # logits = tf.nn.softmax(logits,axis=0)
190 
191         # optimizer =
                tf.train.AdamOptimizer(learning_rate).minimize(nce_loss,global_step=global_step)
192         train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(nce_loss,
                global_step=global_step)
193 
194         # Inicializar variables
195         init = tf.global_variables_initializer()
196 
197         # Crear objeto encargado de almacenar la red
198         saver = tf.train.Saver(max_to_keep=1)
199 
200     config = tf.ConfigProto()
201     config.gpu_options.allow_growth = True
202     with tf.Session(graph=graph, config=config) as session:
203     #with tf.Session(graph=graph) as session:
204 
205         # We must initialize all variables before we use them.
206         init.run()
207 
208 
209         if(save):
210             # Obtener el checkpoint
211             ckpt = tf.train.get_checkpoint_state(model_path)
212 
213             # Si existe el checkpoint restaurar
214             if ckpt and ckpt.model_checkpoint_path:
215                 saver.restore(session, ckpt.model_checkpoint_path)
216 
```

```
217         #Si se desea entrenar
218         if(train):
219             #-------------------------------------------------------------------------------
220             # TRAIN
221             #-------------------------------------------------------------------------------
222 
223             ITRS= int(len(X)/batch_size)
224             RES_ITMS = len(X) - (ITRS*batch_size)
225             if(RES_ITMS>0):ITRS+=1 #Si no es divisin exacta, se aade una iteracin
226 
227             for e in range(epochs):
228                 TRAIN_LOSS = 0.0
229 
230                 print("Epoch "+str(e))
231 
232                 for step in range(ITRS):
233 
234                     batch_inputs, batch_context =
                            getBatchData(X,Y,batch_size,step,ITRS,RES_ITMS)
235 
236                     feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
237 
238                     _, loss_val, batch_lss, gs = session.run([train_step,
                            nce_loss,batch_loss,global_step], feed_dict=feed_dict)
239 
240                     TRAIN_LOSS += np.sum(batch_lss)
241 
242                 #print('Epoch '+str(e)+', last batch loss: '+str(loss_val))
243 
244                 if (save): saver.save(session, complete_path, global_step=global_step)
245 
246                 #-------------------------------------------------------------------------------
247                 # DEV
248                 #-------------------------------------------------------------------------------
249 
250                 if(len(DX)>0):
251 
252                     ITRS_TEST = int(len(DX) / batch_size)
253                     RES_ITMS_TEST = len(DX) - (ITRS_TEST * batch_size)
254                     TEST_LOSS = 0.0
255                     if (RES_ITMS_TEST > 0): ITRS_TEST += 1 # Si no es divisin exacta, se
                            aade una iteracin
256 
257 
258                     for step in range(ITRS_TEST):
259 
260                         batch_inputs, batch_context =
                                getBatchData(DX,DY,batch_size,step,ITRS_TEST,RES_ITMS_TEST)
261 
262                         feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
263                         loss_val,batch_lss = session.run([nce_loss,batch_loss],
                                feed_dict=feed_dict)
264 
265                         TEST_LOSS+=np.sum(batch_lss)
266 
267                     dev_hist.append(TEST_LOSS / (len(DX) * 1.0))
268                     train_hist.append(TRAIN_LOSS / (len(X) *  1.0))
```

```
269
270             if (len(dev_hist) >= slope_size):
271                 # Calcular la pendiente
272                 print(getSlope(dev_hist[-slope_size:]))
273                 print(dev_hist[-slope_size:])
274
275                 if (getSlope(dev_hist[-slope_size:]) > -1e-5 or e==epochs-1):
276
277                     def printArray(data,data2):
278                         for i in range(len(data)):
279                             a=data[i]
280                             b=data2[i]
281                             print(str(a).replace(".",",")+"\t"+str(b).replace(".",","))
282
283                     print("-" * 50)
284                     print(e)
285                     printArray(dev_hist,train_hist)
286                     return
287         else:
288             #
                ------------------------------------------------------------------------------
289             # TEST FINAL
290             #
                ------------------------------------------------------------------------------
291             batch_size = 300
292
293             IN_TOP_5 = 0
294             IN_TOP_10 = 0
295             IN_TOP_20 = 0
296             IN_TOP_50 = 0
297             IN_TOP_100 = 0
298             IN_TOP_1000 = 0
299             real_positions = []
300
301             items = 0
302
303             ITRS_TEST = int(len(TX) / batch_size)
304             RES_ITMS_TEST = len(TX) - (ITRS_TEST * batch_size)
305             if (RES_ITMS_TEST > 0): ITRS_TEST += 1  # Si no es divisin exacta, se aade una
                    iteracin
306
307             for step in range(ITRS_TEST):
308                 #tss = time.time()
309
310                 batch_inputs, batch_context = getBatchData(TX, TY, batch_size, step,
                        ITRS_TEST, RES_ITMS_TEST)
311                 top_values,top_indx = session.run([top_v,top_i], feed_dict={train_dataset:
                        batch_inputs, train_labels: batch_context})
312
313                 for j in range(len(batch_context)):
314                     real_position = int(np.where(top_indx[j, :] == batch_context[j, 0])[0])
315                     real_positions.append(real_position)
316
317                     IN_TOP_5 += 1 if real_position <= 5 else 0
318                     IN_TOP_10 += 1 if real_position <= 10 else 0
319                     IN_TOP_20 += 1 if real_position <= 20 else 0
320                     IN_TOP_50 += 1 if real_position <= 50 else 0
```

```
321                     IN_TOP_100 += 1 if real_position <= 100 else 0
322                     IN_TOP_1000 += 1 if real_position <= 1000 else 0
323 
324                 items += len(batch_inputs)*1.0
325 
326                 PTF = IN_TOP_5 / items
327                 PTT = IN_TOP_10 / items
328                 PTTW = IN_TOP_20 / items
329                 PTFT = IN_TOP_50 / items
330                 PTCIEN = IN_TOP_100 / items
331                 PTMIL = IN_TOP_1000 / items
332 
333                 print(np.mean(real_positions), np.std(real_positions),
                        np.median(real_positions))
334                 print(str(PTF).replace(".", ",") + "\t" + str(PTT).replace(".", ",") + "\t"
335                       + str(PTTW).replace(".",",") + "\t" + str(PTFT).replace(".", ",") + "\t"
336                       + str(PTCIEN).replace(".", ",") + "\t" + str(PTMIL).replace(".",","))
337 
338                 #print(time.time()-tss)
339 
340 
341             print(np.mean(real_positions), np.std(real_positions), np.median(real_positions))
342 
343         #-------------------------------------------------------------------------------------------------------------
344 
345 ##########################################################################################################
346 # Llamadas
347 ##########################################################################################################
348 
349 TEST_NAME="test_043"
350 
351 os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
352 
353 #Se obtinenen cada uno de los embeddings aprendidos (incluidas las canciones)
354 D2V_DM_PRESENT   = np.load('embeddings/d2v_dm/present/EMB_MATRIX')
355 D2V_DBOW_PRESENT  = np.load('embeddings/d2v_dbow/present/EMB_MATRIX')
356 W2V  = np.load('embeddings/w2v/EMB_MATRIX')
357 
358 #Se concatenan los embeddings de los usuarios para generar el perfil
359 D2V_EMB = np.concatenate((D2V_DM_PRESENT,D2V_DBOW_PRESENT),axis=1) # El reciente = Reciente
    DM + Reciente DBOW
360 
361 #Se cargan los datos de train de las canciones para conocer los IDs reales
362 DATA = np.load("datos/Extend/ARRAY_ARRAYS_TRAIN.pkl")
363 
364 #Obtener una lista con los ID de canciones que no aestn el la matriz de embeddings
365 ALL = set(list(range(W2V.shape[0])))
366 REAL = set(list(set(list(map(int, DATA)))))
367 
368 ##########################################################################################################
369 # TENSORFLOW
370 ##########################################################################################################
371 
372 #-------------------------------------------------------------------------------------------------------------
373 # Grid Search + Early Stopping
374 #-------------------------------------------------------------------------------------------------------------
375 '''
```

```
X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))
rates = [0.000001,1, 0.01, 0.0001]

for r in rates:
    config = {"batch_size": 1024, "learning_rate": r, "epochs": 1000, "seed": 100}

    print("Learning Rate: "+str(r))
    print("-" * 50)
    trainNet(X, Y, DX, DY, TX, TY, config, user_size, train=True, save=False)
    print("-"*50)

exit()
'''
#-------------------------------------------------------------------------------
# Train
#-------------------------------------------------------------------------------
'''
X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))

#Para esta fase, se une TRAIN y DEV
X = np.concatenate((X,DX),axis=0)
Y = np.concatenate((Y,DY),axis=0)

DX=[]
DY=[]

config = {"batch_size": 1024, "learning_rate": 0.01, "epochs":1000, "seed":100}
trainNet(X, Y, DX, DY, TX, TY, config, user_size, train=True, save=True)

exit()
'''
#-------------------------------------------------------------------------------
# Test
#-------------------------------------------------------------------------------

X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))

config = {"batch_size": 1024, "learning_rate": 0.01, "epochs":1000, "seed":100}
trainNet(X, Y, DX, DY, TX, TY, config, user_size,train=False, save=True)
```

4.20. TEST 20

```
# -*- coding: utf-8 -*-

import os.path
import os
import tensorflow as tf
import numpy as np
import random
import time
import math
import pandas as pd
import pickle
from collections import Counter
from scipy.stats import linregress
from tensorflow.python.framework import ops as framework_ops

########################################################################################################################
# @Mtodos
########################################################################################################################

def getData(REAL_SONG_IDS=[],data ="datos/Append/ARRAY_ARRAYS_REC.pkl",shuffle=True,
    seed=100, train=.7):

    DATA = np.load(data)

    #DATA = DATA[:5];print("ELIMINAR FILTRO")

    #User, cancion 1, cancion 2, cancion 3
    TRAIN = np.empty([0, 4], dtype=np.int32)
    DEV = np.empty([0, 4], dtype=np.int32)
    TEST = np.empty([0, 4], dtype=np.int32)

    for u in range(len(DATA)):

        # Eliminar canciones escuchadas que no tienen embedding
        USER_PLAYS = DATA[u]
        USER_PLAYS = pd.DataFrame(list(map(int, USER_PLAYS)))
        USER_PLAYS = USER_PLAYS.loc[USER_PLAYS[0].isin(REAL_SONG_IDS)][0].values

        #Separar en train dev test
        TRAIN_ITEMS = int(len(USER_PLAYS)*train)
        DEV_ITEMS = int((len(USER_PLAYS)-TRAIN_ITEMS)/2)

        TRAIN_DATA = USER_PLAYS[:TRAIN_ITEMS]
        DEV_DATA = USER_PLAYS[TRAIN_ITEMS:TRAIN_ITEMS+DEV_ITEMS]
        TEST_DATA = USER_PLAYS[TRAIN_ITEMS+DEV_ITEMS:]

        # Crear ejemplos
        USR_TRAIN = np.empty([len(TRAIN_DATA)-2, 4], dtype=np.int32)
        USR_TRAIN[:,0] = u
        USR_TRAIN[:,1] = TRAIN_DATA[:len(TRAIN_DATA)-2]
        USR_TRAIN[:,2] = TRAIN_DATA[1:-1]
        USR_TRAIN[:,3] = TRAIN_DATA[2:]

        USR_DEV = np.empty([len(DEV_DATA) -2, 4], dtype=np.int32)
        USR_DEV[:, 0] = u
```

```
        USR_DEV[:, 1] = DEV_DATA[:len(DEV_DATA)-2]
        USR_DEV[:, 2] = DEV_DATA[1:-1]
        USR_DEV[:, 3] = DEV_DATA[2:]

        USR_TEST = np.empty([len(TEST_DATA)-2, 4], dtype=np.int32)
        USR_TEST[:, 0] = u
        USR_TEST[:, 1] = TEST_DATA[:len(TEST_DATA) -2]
        USR_TEST[:, 2] = TEST_DATA[1:-1]
        USR_TEST[:, 3] = TEST_DATA[2:]

        # nAadir al global
        TRAIN = np.concatenate([TRAIN,USR_TRAIN])
        DEV = np.concatenate([DEV,USR_DEV])
        TEST = np.concatenate([TEST,USR_TEST])

    #Mezclar
    if(shuffle):
        np.random.seed(seed)
        np.random.shuffle(TRAIN)
        np.random.shuffle(DEV)
        np.random.shuffle(TEST)

    return
        TRAIN[:,:3],np.matrix(TRAIN[:,-1]).T,DEV[:,:3],np.matrix(DEV[:,-1]).T,TEST[:,:3],np.matrix(TEST

def getBatchData(X, Y, batch_size, step, ITRS, RES_ITMS):

    # Si se no aadi una iteracin, introducir los datos restantes
    if (step == ITRS - 1 and RES_ITMS > 0):
        batch_inputs = X[(ITRS - 1) * batch_size:]
        batch_context = Y[(ITRS - 1) * batch_size:]
    # Si no, es un batch normal
    else:
        st = step * batch_size
        nd = (step + 1) * batch_size

        batch_inputs = X[st:nd]
        batch_context = Y[st:nd]
    return(batch_inputs,batch_context)

def trainNet(X,Y, DX,DY,TX,TY,config, user_size, train=True, save=True):

    def getSlope(data):
        r = linregress(range(len(data)), data)
        return r.slope

    batch_size = config['batch_size']
    learning_rate = config['learning_rate']
    epochs = config['epochs']

    user_size = user_size
    song_size = W2V.shape[0]

    user_emb_size = 128
    song_emb_size = 64
    hidden_size = 32

```

```
111     profile_size = user_emb_size + song_emb_size + song_emb_size
112     user_concat_size = user_emb_size * 2
113 
114     num_sampled = 64  # Ejemplos negativos (Para el NSE)
115 
116     model_name = 'model'
117     model_path = 'models/' + TEST_NAME + '/model'
118     complete_path = model_path + "/" + model_name
119 
120     # Se almacena la loss de dev de las 5 ultimas epochs
121     slope_size = 100
122     train_hist = []
123     dev_hist = []
124 
125     # Creacin del grafo de TF.
126     graph = tf.Graph()
127 
128     with graph.as_default():
129 
130         if(config["seed"]!=1):
131             tf.set_random_seed(config["seed"])
132 
133         # Nmero global de iteraciones
134         global_step = tf.Variable(0, dtype=tf.int32, trainable=False, name='global_step')
135 
136         # Datos de entrada
            ------------------------------------------------------------------------------------
137         # Array del tamao del batch con las X
138         train_dataset = tf.placeholder(tf.int32, shape=[None, 3], name="train_inputs")
139 
140         # Array del tamao del batch con las Y asociadas
141         train_labels = tf.placeholder(tf.float32, shape=[None, 1], name="train_labels")
142 
143         # Embeddings
            ------------------------------------------------------------------------------------
144 
145         D2V_EMB_VRB = tf.Variable(D2V_EMB, trainable=False, name="doc_embeddings")
146 
147         T0 = tf.Variable(tf.truncated_normal([user_concat_size, user_emb_size], mean=0.0,
                stddev=1.0 / math.sqrt(hidden_size)), name="T0")
148         B0 = tf.Variable(tf.zeros([user_emb_size]), name="B0")
149 
150         W2V_EMB_VRB = tf.Variable(W2V, trainable=False, name="word_embeddings")
151 
152         # Operaciones
            ------------------------------------------------------------------------------------
153 
154         # Embedding de documento
155         ed = tf.nn.embedding_lookup(D2V_EMB_VRB, train_dataset[:, 0])
156 
157         #Transformar concat documento de 256 a 128
158         ed = tf.matmul(ed,T0)+B0
159 
160         # Embedding de cancion 1
161         ec1 = tf.nn.embedding_lookup(W2V_EMB_VRB, train_dataset[:, 1])
162 
163         # Embedding de cancion 2
```

```
        ec2 = tf.nn.embedding_lookup(W2V_EMB_VRB, train_dataset[:, 2])

        # aClculo de LOSS y optimizador—————————————————————————————————————————————

        embed = []
        embed.append(ed)
        embed.append(ec1)
        embed.append(ec2)
        embed = tf.concat(embed, 1)

        # softmax weights, W and D vectors should be concatenated before applying softmax
        weights = tf.Variable(tf.truncated_normal([song_size, profile_size], stddev=1.0 / math.sqrt(song_size)), name="weights")
        # softmax biases
        biases = tf.Variable(tf.zeros([song_size]), name="biases")

        # aClculo de LOSS y optimizador—————————————————————————————————————————————
        nce_loss = tf.nn.nce_loss(weights=weights,
                                  biases=biases,
                                  labels=train_labels,
                                  inputs=embed,
                                  num_sampled=num_sampled,
                                  num_classes=song_size,
                                  name="nce_loss")

        batch_loss = nce_loss
        nce_loss = tf.reduce_mean(nce_loss)

        logits = tf.matmul(embed, tf.transpose(weights))
        logits = tf.nn.bias_add(logits, biases)

        top_v,top_i = tf.nn.top_k(logits,k=song_size)

        # logits = tf.nn.softmax(logits,axis=0)

        # optimizer = tf.train.AdamOptimizer(learning_rate).minimize(nce_loss,global_step=global_step)
        train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(nce_loss, global_step=global_step)

        # Inicializar variables
        init = tf.global_variables_initializer()

        # Crear objeto encargado de almacenar la red
        saver = tf.train.Saver(max_to_keep=1)

    config = tf.ConfigProto()
    config.gpu_options.allow_growth = True
    with tf.Session(graph=graph, config=config) as session:
    #with tf.Session(graph=graph) as session:

        # We must initialize all variables before we use them.
        init.run()
```

```
216         if(save):
217             # Obtener el checkpoint
218             ckpt = tf.train.get_checkpoint_state(model_path)
219 
220             # Si existe el checkpoint restaurar
221             if ckpt and ckpt.model_checkpoint_path:
222                 saver.restore(session, ckpt.model_checkpoint_path)
223 
224         #Si se desea entrenar
225         if(train):
226             #-----------------------------------------------------------------------------------------------
227             # TRAIN
228             #-----------------------------------------------------------------------------------------------
229 
230             ITRS= int(len(X)/batch_size)
231             RES_ITMS = len(X) - (ITRS*batch_size)
232             if(RES_ITMS>0):ITRS+=1 #Si no es divisin exacta, se aade una iteracin
233 
234             for e in range(epochs):
235                 TRAIN_LOSS = 0.0
236 
237                 print("Epoch "+str(e))
238 
239                 for step in range(ITRS):
240 
241                     batch_inputs, batch_context =
                            getBatchData(X,Y,batch_size,step,ITRS,RES_ITMS)
242 
243                     feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
244 
245                     _, loss_val, batch_lss, gs = session.run([train_step,
                            nce_loss,batch_loss,global_step], feed_dict=feed_dict)
246 
247                     TRAIN_LOSS += np.sum(batch_lss)
248 
249                 #print('Epoch '+str(e)+', last batch loss: '+str(loss_val))
250 
251                 if (save): saver.save(session, complete_path, global_step=global_step)
252 
253                 #-----------------------------------------------------------------------------------------------
254                 # DEV
255                 #-----------------------------------------------------------------------------------------------
256 
257                 if(len(DX)>0):
258 
259                     ITRS_TEST = int(len(DX) / batch_size)
260                     RES_ITMS_TEST = len(DX) - (ITRS_TEST * batch_size)
261                     TEST_LOSS = 0.0
262                     if (RES_ITMS_TEST > 0): ITRS_TEST += 1 # Si no es divisin exacta, se
                            aade una iteracin
263 
264 
265                     for step in range(ITRS_TEST):
266 
267                         batch_inputs, batch_context =
                                getBatchData(DX,DY,batch_size,step,ITRS_TEST,RES_ITMS_TEST)
268 
```

```
                    feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
                    loss_val,batch_lss = session.run([nce_loss,batch_loss],
                        feed_dict=feed_dict)

                    TEST_LOSS+=np.sum(batch_lss)

                dev_hist.append(TEST_LOSS / (len(DX) * 1.0))
                train_hist.append(TRAIN_LOSS / (len(X) *  1.0))

            if (len(dev_hist) >= slope_size):
                # Calcular la pendiente
                print(getSlope(dev_hist[-slope_size:]))
                print(dev_hist[-slope_size:])

                if (getSlope(dev_hist[-slope_size:]) > -1e-5 or e==epochs-1):

                    def printArray(data,data2):
                        for i in range(len(data)):
                            a=data[i]
                            b=data2[i]
                            print(str(a).replace(".",",")+"\t"+str(b).replace(".",","))

                    print("-" * 50)
                    print(e)
                    printArray(dev_hist,train_hist)
                    return
    else:
        #
        # TEST FINAL
        #
        batch_size = 300

        IN_TOP_5 = 0
        IN_TOP_10 = 0
        IN_TOP_20 = 0
        IN_TOP_50 = 0
        IN_TOP_100 = 0
        IN_TOP_1000 = 0
        real_positions = []

        items = 0

        ITRS_TEST = int(len(TX) / batch_size)
        RES_ITMS_TEST = len(TX) - (ITRS_TEST * batch_size)
        if (RES_ITMS_TEST > 0): ITRS_TEST += 1  # Si no es divisin exacta, se aade una
            iteracin

        for step in range(ITRS_TEST):
            #tss = time.time()

            batch_inputs, batch_context = getBatchData(TX, TY, batch_size, step,
                ITRS_TEST, RES_ITMS_TEST)
            top_values,top_indx = session.run([top_v,top_i], feed_dict={train_dataset:
                batch_inputs, train_labels: batch_context})
```

```
320                 for j in range(len(batch_context)):
321                     real_position = int(np.where(top_indx[j, :] == batch_context[j, 0])[0])
322                     real_positions.append(real_position)
323 
324                     IN_TOP_5 += 1 if real_position <= 5 else 0
325                     IN_TOP_10 += 1 if real_position <= 10 else 0
326                     IN_TOP_20 += 1 if real_position <= 20 else 0
327                     IN_TOP_50 += 1 if real_position <= 50 else 0
328                     IN_TOP_100 += 1 if real_position <= 100 else 0
329                     IN_TOP_1000 += 1 if real_position <= 1000 else 0
330 
331                 items += len(batch_inputs)*1.0
332 
333                 PTF = IN_TOP_5 / items
334                 PTT = IN_TOP_10 / items
335                 PTTW = IN_TOP_20 / items
336                 PTFT = IN_TOP_50 / items
337                 PTCIEN = IN_TOP_100 / items
338                 PTMIL = IN_TOP_1000 / items
339 
340                 print(np.mean(real_positions), np.std(real_positions),
                        np.median(real_positions))
341                 print(str(PTF).replace(".", ",") + "\t" + str(PTT).replace(".", ",") + "\t"
342                       + str(PTTW).replace(".",",") + "\t" + str(PTFT).replace(".", ",") + "\t"
343                       + str(PTCIEN).replace(".", ",") + "\t" + str(PTMIL).replace(".",","))
344 
345                 #print(time.time()-tss)
346 
347 
348             print(np.mean(real_positions), np.std(real_positions), np.median(real_positions))
349 
350         #-----------------------------------------------------------------------------------------------------------
351 
352 ##################################################################################################################
353 # Llamadas
354 ##################################################################################################################
355 
356 TEST_NAME="test_053"
357 
358 os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
359 
360 #Se obtinenen cada uno de los embeddings aprendidos (incluidas las canciones)
361 D2V_DM_COMPLETE   = np.load('embeddings/d2v_dm/complete/EMB_MATRIX')
362 D2V_DBOW_COMPLETE = np.load('embeddings/d2v_dbow/complete/EMB_MATRIX')
363 
364 #Se obtinenen cada uno de los embeddings aprendidos (incluidas las canciones)
365 D2V_DM_PRESENT  = np.load('embeddings/d2v_dm/present/EMB_MATRIX')
366 D2V_DBOW_PRESENT = np.load('embeddings/d2v_dbow/present/EMB_MATRIX')
367 
368 W2V  = np.load('embeddings/w2v/EMB_MATRIX')
369 
370 #Se concatenan los embeddings de los usuarios para generar el perfil
371 D2V_CONS = np.concatenate((D2V_DM_COMPLETE,D2V_DBOW_COMPLETE),axis=1) # El consolidado =
        Consolidado DM + Consolidado DBOW
372 D2V_PRES = np.concatenate((D2V_DM_PRESENT,D2V_DBOW_PRESENT),axis=1) # El presente = Presente
        DM + Presente DBOW
373 
```

```
D2V_EMB = np.concatenate((D2V_CONS,D2V_PRES),axis=1) # El perfil completo CONS + PRES

#Se cargan los datos de train de las canciones para conocer los IDs reales
DATA = np.load("datos/Extend/ARRAY_ARRAYS_TRAIN.pkl")

#Obtener una lista con los ID de canciones que no aestn el la matriz de embeddings
ALL = set(list(range(W2V.shape[0])))
REAL = set(list(set(list(map(int, DATA)))))

#############################################################################################################
# TENSORFLOW
#############################################################################################################

#-----------------------------------------------------------------------------------------------------------
# Grid Search + Early Stopping
#-----------------------------------------------------------------------------------------------------------
'''
X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))
rates = [0.000001,1, 0.01, 0.0001]

for r in rates:
    config = {"batch_size": 1024, "learning_rate": r, "epochs": 1000, "seed": 100}

    print("Learning Rate: "+str(r))
    print("-" * 50)
    trainNet(X, Y, DX, DY, TX, TY, config, user_size, train=True, save=False)
    print("-"*50)

exit()
'''
#-----------------------------------------------------------------------------------------------------------
# Train
#-----------------------------------------------------------------------------------------------------------
'''
X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))

#Para esta fase, se une TRAIN y DEV
X = np.concatenate((X,DX),axis=0)
Y = np.concatenate((Y,DY),axis=0)

DX=[]
DY=[]

config = {"batch_size": 1024, "learning_rate": 0.01, "epochs":167, "seed":100}
trainNet(X, Y, DX, DY, TX, TY, config, user_size, train=True, save=True)

exit()
'''
#-----------------------------------------------------------------------------------------------------------
# Test
#-----------------------------------------------------------------------------------------------------------

X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))

config = {"batch_size": 1024, "learning_rate": 0.01, "epochs":167, "seed":100}
trainNet(X, Y, DX, DY, TX, TY, config, user_size,train=False, save=True)
```

4.21. TEST 21

```
# -*- coding: utf-8 -*-

import os.path
import os
import tensorflow as tf
import numpy as np
import random
import time
import math
import pandas as pd
import pickle
from collections import Counter
from scipy.stats import linregress
from tensorflow.python.framework import ops as framework_ops

########################################################################################################################
# @Mtodos
########################################################################################################################

def getData(REAL_SONG_IDS=[],data ="datos/Append/ARRAY_ARRAYS_REC.pkl",shuffle=True,
    seed=100, train=.7):

    DATA = np.load(data)

    #DATA = DATA[:5];print("ELIMINAR FILTRO")

    #User, cancion 1, cancion 2, cancion 3
    TRAIN = np.empty([0, 4], dtype=np.int32)
    DEV = np.empty([0, 4], dtype=np.int32)
    TEST = np.empty([0, 4], dtype=np.int32)

    for u in range(len(DATA)):

        # Eliminar canciones escuchadas que no tienen embedding
        USER_PLAYS = DATA[u]
        USER_PLAYS = pd.DataFrame(list(map(int, USER_PLAYS)))
        USER_PLAYS = USER_PLAYS.loc[USER_PLAYS[0].isin(REAL_SONG_IDS)][0].values

        #Separar en train dev test
        TRAIN_ITEMS = int(len(USER_PLAYS)*train)
        DEV_ITEMS = int((len(USER_PLAYS)-TRAIN_ITEMS)/2)

        TRAIN_DATA = USER_PLAYS[:TRAIN_ITEMS]
        DEV_DATA = USER_PLAYS[TRAIN_ITEMS:TRAIN_ITEMS+DEV_ITEMS]
        TEST_DATA = USER_PLAYS[TRAIN_ITEMS+DEV_ITEMS:]

        # Crear ejemplos
        USR_TRAIN = np.empty([len(TRAIN_DATA)-2, 4], dtype=np.int32)
        USR_TRAIN[:,0] = u
        USR_TRAIN[:,1] = TRAIN_DATA[:len(TRAIN_DATA)-2]
        USR_TRAIN[:,2] = TRAIN_DATA[1:-1]
        USR_TRAIN[:,3] = TRAIN_DATA[2:]

        USR_DEV = np.empty([len(DEV_DATA) -2, 4], dtype=np.int32)
        USR_DEV[:, 0] = u
```

```
        USR_DEV[:, 1] = DEV_DATA[:len(DEV_DATA)-2]
        USR_DEV[:, 2] = DEV_DATA[1:-1]
        USR_DEV[:, 3] = DEV_DATA[2:]

        USR_TEST = np.empty([len(TEST_DATA)-2, 4], dtype=np.int32)
        USR_TEST[:, 0] = u
        USR_TEST[:, 1] = TEST_DATA[:len(TEST_DATA) -2]
        USR_TEST[:, 2] = TEST_DATA[1:-1]
        USR_TEST[:, 3] = TEST_DATA[2:]

        # nAadir al global
        TRAIN = np.concatenate([TRAIN,USR_TRAIN])
        DEV = np.concatenate([DEV,USR_DEV])
        TEST = np.concatenate([TEST,USR_TEST])

    #Mezclar
    if(shuffle):
        np.random.seed(seed)
        np.random.shuffle(TRAIN)
        np.random.shuffle(DEV)
        np.random.shuffle(TEST)

    return
        TRAIN[:,:3],np.matrix(TRAIN[:,-1]).T,DEV[:,:3],np.matrix(DEV[:,-1]).T,TEST[:,:3],np.matrix(TEST

def getBatchData(X, Y, batch_size, step, ITRS, RES_ITMS):

    # Si se no aadi una iteracin, introducir los datos restantes
    if (step == ITRS - 1 and RES_ITMS > 0):
        batch_inputs = X[(ITRS - 1) * batch_size:]
        batch_context = Y[(ITRS - 1) * batch_size:]
    # Si no, es un batch normal
    else:
        st = step * batch_size
        nd = (step + 1) * batch_size

        batch_inputs = X[st:nd]
        batch_context = Y[st:nd]
    return(batch_inputs,batch_context)

def trainNet(X,Y, DX,DY,TX,TY,config, user_size, train=True, save=True):

    def getSlope(data):
        r = linregress(range(len(data)), data)
        return r.slope

    batch_size = config['batch_size']
    learning_rate = config['learning_rate']
    epochs = config['epochs']

    user_size = user_size
    song_size = W2V.shape[0]

    user_emb_size = 128
    song_emb_size = 64
    hidden_size = 32

```

```
111     profile_size = user_emb_size + song_emb_size + song_emb_size
112
113     num_sampled = 64  # Ejemplos negativos (Para el NSE)
114
115     model_name = 'model'
116     model_path = 'models/' + TEST_NAME + '/model'
117     complete_path = model_path + "/" + model_name
118
119     # Se almacena la loss de dev de las 5 ultimas epochs
120     slope_size = 100
121     train_hist = []
122     dev_hist = []
123
124     # Creacion del grafo de TF.
125     graph = tf.Graph()
126
127     with graph.as_default():
128
129         if(config["seed"]!=1):
130             tf.set_random_seed(config["seed"])
131
132         # Numero global de iteraciones
133         global_step = tf.Variable(0, dtype=tf.int32, trainable=False, name='global_step')
134
135         # Datos de entrada
            # -------------------------------------------------------------------------------
136         # Array del tamao del batch con las X
137         train_dataset = tf.placeholder(tf.int32, shape=[None, 3], name="train_inputs")
138
139         # Array del tamao del batch con las Y asociadas
140         train_labels = tf.placeholder(tf.float32, shape=[None, 1], name="train_labels")
141
142         # Embeddings
            # -------------------------------------------------------------------------------
143
144         O1 = tf.Variable(tf.truncated_normal([user_size, user_emb_size], mean=0.0, stddev=1.0
                / math.sqrt(user_emb_size)), name="O1")
145         BO1 = tf.Variable(tf.zeros([user_emb_size]), name="BO1")
146
147         O2 = tf.Variable(tf.truncated_normal([song_size, song_emb_size], mean=0.0, stddev=1.0
                / math.sqrt(song_emb_size)), name="O2")
148         BO2 = tf.Variable(tf.zeros([song_emb_size]), name="BO2")
149
150         # Operaciones
            # -------------------------------------------------------------------------------
151
152         # Embedding de documento
153         ed = tf.nn.embedding_lookup(O1, train_dataset[:, 0])
154         ed = ed + BO1
155
156         # Embedding de cancion 1
157         ec1 = tf.nn.embedding_lookup(O2, train_dataset[:, 1])
158         ec1 = ec1 + BO2
159
160         # Embedding de cancion 2
161         ec2 = tf.nn.embedding_lookup(O2, train_dataset[:, 2])
162         ec2 = ec2 + BO2
```

```
        # aClculo de LOSS y optimizador—
            ---------------------------------------------------------------------------------------------------------

        embed = []
        embed.append(ed)
        embed.append(ec1)
        embed.append(ec2)
        embed = tf.concat(embed, 1)

        # softmax weights, W and D vectors should be concatenated before applying softmax
        weights = tf.Variable(tf.truncated_normal([song_size, profile_size], stddev=1.0 /
            math.sqrt(song_size)), name="weights")
        # softmax biases
        biases = tf.Variable(tf.zeros([song_size]), name="biases")

        # aClculo de LOSS y optimizador—
            ---------------------------------------------------------------------------------------------------------
        nce_loss = tf.nn.nce_loss(weights=weights,
                                  biases=biases,
                                  labels=train_labels,
                                  inputs=embed,
                                  num_sampled=num_sampled,
                                  num_classes=song_size,
                                  name="nce_loss")

        batch_loss = nce_loss
        nce_loss = tf.reduce_mean(nce_loss)

        logits = tf.matmul(embed, tf.transpose(weights))
        logits = tf.nn.bias_add(logits, biases)

        top_v,top_i = tf.nn.top_k(logits,k=song_size)

        # logits = tf.nn.softmax(logits,axis=0)

        # optimizer =
            tf.train.AdamOptimizer(learning_rate).minimize(nce_loss,global_step=global_step)
        train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(nce_loss,
            global_step=global_step)

        # Inicializar variables
        init = tf.global_variables_initializer()

        # Crear objeto encargado de almacenar la red
        saver = tf.train.Saver(max_to_keep=1)

    config = tf.ConfigProto()
    config.gpu_options.allow_growth = True
    with tf.Session(graph=graph, config=config) as session:
    #with tf.Session(graph=graph) as session:

        # We must initialize all variables before we use them.
        init.run()


        if(save):
```

```
215             # Obtener el checkpoint
216             ckpt = tf.train.get_checkpoint_state(model_path)
217 
218             # Si existe el checkpoint restaurar
219             if ckpt and ckpt.model_checkpoint_path:
220                 saver.restore(session, ckpt.model_checkpoint_path)
221 
222         #Si se desea entrenar
223         if(train):
224             #-----------------------------------------------------------------------------------------------
225             # TRAIN
226             #-----------------------------------------------------------------------------------------------
227 
228             ITRS= int(len(X)/batch_size)
229             RES_ITMS = len(X) - (ITRS*batch_size)
230             if(RES_ITMS>0):ITRS+=1 #Si no es divisin exacta, se aade una iteracin
231 
232             for e in range(epochs):
233                 TRAIN_LOSS = 0.0
234 
235                 print("Epoch "+str(e))
236 
237                 for step in range(ITRS):
238 
239                     batch_inputs, batch_context =
                            getBatchData(X,Y,batch_size,step,ITRS,RES_ITMS)
240 
241                     feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
242 
243                     _, loss_val, batch_lss, gs = session.run([train_step,
                            nce_loss,batch_loss,global_step], feed_dict=feed_dict)
244 
245                     TRAIN_LOSS += np.sum(batch_lss)
246 
247                 #print('Epoch '+str(e)+', last batch loss: '+str(loss_val))
248 
249                 if (save): saver.save(session, complete_path, global_step=global_step)
250 
251                 #-----------------------------------------------------------------------------------------------
252                 # DEV
253                 #-----------------------------------------------------------------------------------------------
254 
255                 if(len(DX)>0):
256 
257                     ITRS_TEST = int(len(DX) / batch_size)
258                     RES_ITMS_TEST = len(DX) - (ITRS_TEST * batch_size)
259                     TEST_LOSS = 0.0
260                     if (RES_ITMS_TEST > 0): ITRS_TEST += 1 # Si no es divisin exacta, se
                            aade una iteracin
261 
262 
263                     for step in range(ITRS_TEST):
264 
265                         batch_inputs, batch_context =
                                getBatchData(DX,DY,batch_size,step,ITRS_TEST,RES_ITMS_TEST)
266 
267                         feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
```

```
268                     loss_val,batch_lss = session.run([nce_loss,batch_loss],
                            feed_dict=feed_dict)
269
270                     TEST_LOSS+=np.sum(batch_lss)
271
272                 dev_hist.append(TEST_LOSS / (len(DX) * 1.0))
273                 train_hist.append(TRAIN_LOSS / (len(X) *  1.0))
274
275             if (len(dev_hist) >= slope_size):
276                 # Calcular la pendiente
277                 print(getSlope(dev_hist[-slope_size:]))
278                 print(dev_hist[-slope_size:])
279
280                 if (getSlope(dev_hist[-slope_size:]) > -1e-5 or e==epochs-1):
281
282                     def printArray(data,data2):
283                         for i in range(len(data)):
284                             a=data[i]
285                             b=data2[i]
286                             print(str(a).replace(".",",")+"\t"+str(b).replace(".",","))
287
288                     print("-" * 50)
289                     print(e)
290                     printArray(dev_hist,train_hist)
291                     return
292         else:
293             #
                -------------------------------------------------------------------------------------------------------------------
294             # TEST FINAL
295             #
                -------------------------------------------------------------------------------------------------------------------
296             batch_size = 300
297
298             IN_TOP_5 = 0
299             IN_TOP_10 = 0
300             IN_TOP_20 = 0
301             IN_TOP_50 = 0
302             IN_TOP_100 = 0
303             IN_TOP_1000 = 0
304             real_positions = []
305
306             items = 0
307
308             ITRS_TEST = int(len(TX) / batch_size)
309             RES_ITMS_TEST = len(TX) - (ITRS_TEST * batch_size)
310             if (RES_ITMS_TEST > 0): ITRS_TEST += 1  # Si no es @divisin exacta, se ¤aade una
                    @iteracin
311
312             for step in range(ITRS_TEST):
313                 #tss = time.time()
314
315                 batch_inputs, batch_context = getBatchData(TX, TY, batch_size, step,
                        ITRS_TEST, RES_ITMS_TEST)
316                 top_values,top_indx = session.run([top_v,top_i], feed_dict={train_dataset:
                        batch_inputs, train_labels: batch_context})
317
318                 for j in range(len(batch_context)):
```

```
                    real_position = int(np.where(top_indx[j, :] == batch_context[j, 0])[0])
                    real_positions.append(real_position)

                    IN_TOP_5 += 1 if real_position <= 5 else 0
                    IN_TOP_10 += 1 if real_position <= 10 else 0
                    IN_TOP_20 += 1 if real_position <= 20 else 0
                    IN_TOP_50 += 1 if real_position <= 50 else 0
                    IN_TOP_100 += 1 if real_position <= 100 else 0
                    IN_TOP_1000 += 1 if real_position <= 1000 else 0

                items += len(batch_inputs)*1.0

                PTF = IN_TOP_5 / items
                PTT = IN_TOP_10 / items
                PTTW = IN_TOP_20 / items
                PTFT = IN_TOP_50 / items
                PTCIEN = IN_TOP_100 / items
                PTMIL = IN_TOP_1000 / items

                print(np.mean(real_positions), np.std(real_positions),
                    np.median(real_positions))
                print(str(PTF).replace(".", ",") + "\t" + str(PTT).replace(".", ",") + "\t"
                      + str(PTTW).replace(".",",") + "\t" + str(PTFT).replace(".", ",") + "\t"
                      + str(PTCIEN).replace(".", ",") + "\t" + str(PTMIL).replace(".",","))

                #print(time.time()-tss)


            print(np.mean(real_positions), np.std(real_positions), np.median(real_positions))

        #-------------------------------------------------------------------------------------------------------------------

##########################################################################################################################
# Llamadas
##########################################################################################################################

TEST_NAME="test_013"

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'

#Solamente se carga para conocer el ú nmero de canciones
W2V  = np.load('embeddings/w2v/EMB_MATRIX')

#Se cargan los datos de train de las canciones para conocer los IDs reales
DATA = np.load("datos/Extend/ARRAY_ARRAYS_TRAIN.pkl")

#Obtener una lista con los ID de canciones que no a estn el la matriz de embeddings
ALL = set(list(range(W2V.shape[0])))
REAL = set(list(set(list(map(int, DATA)))))

##########################################################################################################################
# TENSORFLOW
##########################################################################################################################

#-------------------------------------------------------------------------------------------------------------------
# Grid Search + Early Stopping
#-------------------------------------------------------------------------------------------------------------------
```

```
'''
X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))
rates = [0.000001,1, 0.01, 0.0001]


for r in rates:
    config = {"batch_size": 1024, "learning_rate": r, "epochs": 1000, "seed": 100}

    print("Learning Rate: "+str(r))
    print("-" * 50)
    trainNet(X, Y, DX, DY, TX, TY, config, user_size, train=True, save=False)
    print("-"*50)

exit()
'''
#-----------------------------------------------------------------------------------------------------------------------
# Train
#-----------------------------------------------------------------------------------------------------------------------
'''
X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))

#Para esta fase, se une TRAIN y DEV
X = np.concatenate((X,DX),axis=0)
Y = np.concatenate((Y,DY),axis=0)

DX=[]
DY=[]

config = {"batch_size": 1024, "learning_rate": 0.01, "epochs":1000, "seed":100}
trainNet(X, Y, DX, DY, TX, TY, config, user_size, train=True, save=True)

exit()
'''
#-----------------------------------------------------------------------------------------------------------------------
# Test
#-----------------------------------------------------------------------------------------------------------------------

X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))

config = {"batch_size": 1024, "learning_rate": 0.01, "epochs":1000, "seed":100}
trainNet(X, Y, DX, DY, TX, TY, config, user_size,train=False, save=True)
```

4.22. TEST 22

```
# -*- coding: utf-8 -*-

import os.path
import os
import tensorflow as tf
import numpy as np
import random
import time
import math
import pandas as pd
import pickle
from collections import Counter
from scipy.stats import linregress
from tensorflow.python.framework import ops as framework_ops

########################################################################################################################
# @Mtodos
########################################################################################################################

def getData(REAL_SONG_IDS=[],data ="datos/Append/ARRAY_ARRAYS_REC.pkl",shuffle=True, seed=100, train=.7):

    DATA = np.load(data)

    #DATA = DATA[:5];print("ELIMINAR FILTRO")

    #User, cancion 1, cancion 2, cancion 3
    TRAIN = np.empty([0, 4], dtype=np.int32)
    DEV = np.empty([0, 4], dtype=np.int32)
    TEST = np.empty([0, 4], dtype=np.int32)

    for u in range(len(DATA)):

        # Eliminar canciones escuchadas que no tienen embedding
        USER_PLAYS = DATA[u]
        USER_PLAYS = pd.DataFrame(list(map(int, USER_PLAYS)))
        USER_PLAYS = USER_PLAYS.loc[USER_PLAYS[0].isin(REAL_SONG_IDS)][0].values

        #Separar en train dev test
        TRAIN_ITEMS = int(len(USER_PLAYS)*train)
        DEV_ITEMS = int((len(USER_PLAYS)-TRAIN_ITEMS)/2)

        TRAIN_DATA = USER_PLAYS[:TRAIN_ITEMS]
        DEV_DATA = USER_PLAYS[TRAIN_ITEMS:TRAIN_ITEMS+DEV_ITEMS]
        TEST_DATA = USER_PLAYS[TRAIN_ITEMS+DEV_ITEMS:]

        # Crear ejemplos
        USR_TRAIN = np.empty([len(TRAIN_DATA)-2, 4], dtype=np.int32)
        USR_TRAIN[:,0] = u
        USR_TRAIN[:,1] = TRAIN_DATA[:len(TRAIN_DATA)-2]
        USR_TRAIN[:,2] = TRAIN_DATA[1:-1]
        USR_TRAIN[:,3] = TRAIN_DATA[2:]

        USR_DEV = np.empty([len(DEV_DATA) -2, 4], dtype=np.int32)
        USR_DEV[:, 0] = u
```

```
55          USR_DEV[:, 1] = DEV_DATA[:len(DEV_DATA)-2]
56          USR_DEV[:, 2] = DEV_DATA[1:-1]
57          USR_DEV[:, 3] = DEV_DATA[2:]
58
59          USR_TEST = np.empty([len(TEST_DATA)-2, 4], dtype=np.int32)
60          USR_TEST[:, 0] = u
61          USR_TEST[:, 1] = TEST_DATA[:len(TEST_DATA) -2]
62          USR_TEST[:, 2] = TEST_DATA[1:-1]
63          USR_TEST[:, 3] = TEST_DATA[2:]
64
65          # nAadir al global
66          TRAIN = np.concatenate([TRAIN,USR_TRAIN])
67          DEV = np.concatenate([DEV,USR_DEV])
68          TEST = np.concatenate([TEST,USR_TEST])
69
70      #Mezclar
71      if(shuffle):
72          np.random.seed(seed)
73          np.random.shuffle(TRAIN)
74          np.random.shuffle(DEV)
75          np.random.shuffle(TEST)
76
77      return
            TRAIN[:,:3],np.matrix(TRAIN[:,-1]).T,DEV[:,:3],np.matrix(DEV[:,-1]).T,TEST[:,:3],np.matrix(TEST
78
79  def getBatchData(X, Y, batch_size, step, ITRS, RES_ITMS):
80
81      # Si se no aadi una iteracin, introducir los datos restantes
82      if (step == ITRS - 1 and RES_ITMS > 0):
83          batch_inputs = X[(ITRS - 1) * batch_size:]
84          batch_context = Y[(ITRS - 1) * batch_size:]
85      # Si no, es un batch normal
86      else:
87          st = step * batch_size
88          nd = (step + 1) * batch_size
89
90          batch_inputs = X[st:nd]
91          batch_context = Y[st:nd]
92      return(batch_inputs,batch_context)
93
94  def trainNet(X,Y, DX,DY,TX,TY,config, user_size, train=True, save=True):
95
96      def getSlope(data):
97          r = linregress(range(len(data)), data)
98          return r.slope
99
100     batch_size = config['batch_size']
101     learning_rate = config['learning_rate']
102     epochs = config['epochs']
103
104     user_size = user_size
105     song_size = W2V.shape[0]
106
107     user_emb_size = 128
108     song_emb_size = 64
109     hidden_size = 32
110
```

```
111     profile_size = user_emb_size + song_emb_size + song_emb_size
112
113     num_sampled = 64  # Ejemplos negativos (Para el NSE)
114
115     model_name = 'model'
116     model_path = 'models/' + TEST_NAME + '/model'
117     complete_path = model_path + "/" + model_name
118
119     # Se almacena la loss de dev de las 5 ultimas epochs
120     slope_size = 100
121     train_hist = []
122     dev_hist = []
123
124     # Creacin del grafo de TF.
125     graph = tf.Graph()
126
127     with graph.as_default():
128
129         if(config["seed"]!=1):
130             tf.set_random_seed(config["seed"])
131
132         # Nmero global de iteraciones
133         global_step = tf.Variable(0, dtype=tf.int32, trainable=False, name='global_step')
134
135         # Datos de entrada
            ------------------------------------------------------------------------------
136         # Array del tamao del batch con las X
137         train_dataset = tf.placeholder(tf.int32, shape=[None, 3], name="train_inputs")
138
139         # Array del tamao del batch con las Y asociadas
140         train_labels = tf.placeholder(tf.float32, shape=[None, 1], name="train_labels")
141
142         # Embeddings
            ------------------------------------------------------------------------------
143
144         D2V_EMB_VRB = tf.Variable(D2V_EMB, trainable=False, name="doc_embeddings")
145
146         O2 = tf.Variable(tf.truncated_normal([song_size, song_emb_size], mean=0.0, stddev=1.0
                / math.sqrt(song_emb_size)), name="O2")
147         BO2 = tf.Variable(tf.zeros([song_emb_size]), name="BO2")
148
149         # Operaciones
            ------------------------------------------------------------------------------
150
151         # Embedding de documento
152         ed = tf.nn.embedding_lookup(D2V_EMB_VRB, train_dataset[:, 0])
153
154         # Embedding de cancion 1
155         ec1 = tf.nn.embedding_lookup(O2, train_dataset[:, 1])
156         ec1 = ec1 + BO2
157
158         # Embedding de cancion 2
159         ec2 = tf.nn.embedding_lookup(O2, train_dataset[:, 2])
160         ec2 = ec2 + BO2
161
162         # Clculo de LOSS y optimizador
            ------------------------------------------------------------------------------
```

```
        embed = []
        embed.append(ed)
        embed.append(ec1)
        embed.append(ec2)
        embed = tf.concat(embed, 1)

        # softmax weights, W and D vectors should be concatenated before applying softmax
        weights = tf.Variable(tf.truncated_normal([song_size, profile_size], stddev=1.0 /
            math.sqrt(song_size)), name="weights")
        # softmax biases
        biases = tf.Variable(tf.zeros([song_size]), name="biases")

        # aClculo de LOSS y optimizador—
            ------------------------------------------------------------------------------------------
        nce_loss = tf.nn.nce_loss(weights=weights,
                                  biases=biases,
                                  labels=train_labels,
                                  inputs=embed,
                                  num_sampled=num_sampled,
                                  num_classes=song_size,
                                  name="nce_loss")

        batch_loss = nce_loss
        nce_loss = tf.reduce_mean(nce_loss)

        logits = tf.matmul(embed, tf.transpose(weights))
        logits = tf.nn.bias_add(logits, biases)

        top_v,top_i = tf.nn.top_k(logits,k=song_size)

        # logits = tf.nn.softmax(logits,axis=0)

        # optimizer =
            tf.train.AdamOptimizer(learning_rate).minimize(nce_loss,global_step=global_step)
        train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(nce_loss,
            global_step=global_step)

        # Inicializar variables
        init = tf.global_variables_initializer()

        # Crear objeto encargado de almacenar la red
        saver = tf.train.Saver(max_to_keep=1)

    config = tf.ConfigProto()
    config.gpu_options.allow_growth = True
    with tf.Session(graph=graph, config=config) as session:
    #with tf.Session(graph=graph) as session:

        # We must initialize all variables before we use them.
        init.run()


        if(save):
            # Obtener el checkpoint
            ckpt = tf.train.get_checkpoint_state(model_path)

```

```
216             # Si existe el checkpoint restaurar
217             if ckpt and ckpt.model_checkpoint_path:
218                 saver.restore(session, ckpt.model_checkpoint_path)
219 
220         #Si se desea entrenar
221         if(train):
222             #-----------------------------------------------------------------------------------------------------
223             # TRAIN
224             #-----------------------------------------------------------------------------------------------------
225 
226             ITRS= int(len(X)/batch_size)
227             RES_ITMS = len(X) - (ITRS*batch_size)
228             if(RES_ITMS>0):ITRS+=1 #Si no es ødivisin exacta, se ñaade una øiteracin
229 
230             for e in range(epochs):
231                 TRAIN_LOSS = 0.0
232 
233                 print("Epoch "+str(e))
234 
235                 for step in range(ITRS):
236 
237                     batch_inputs, batch_context =
                            getBatchData(X,Y,batch_size,step,ITRS,RES_ITMS)
238 
239                     feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
240 
241                     _, loss_val, batch_lss, gs = session.run([train_step,
                            nce_loss,batch_loss,global_step], feed_dict=feed_dict)
242 
243                     TRAIN_LOSS += np.sum(batch_lss)
244 
245                 #print('Epoch '+str(e)+', last batch loss: '+str(loss_val))
246 
247                 if (save): saver.save(session, complete_path, global_step=global_step)
248 
249                 #-----------------------------------------------------------------------------------------------------
250                 # DEV
251                 #-----------------------------------------------------------------------------------------------------
252 
253                 if(len(DX)>0):
254 
255                     ITRS_TEST = int(len(DX) / batch_size)
256                     RES_ITMS_TEST = len(DX) - (ITRS_TEST * batch_size)
257                     TEST_LOSS = 0.0
258                     if (RES_ITMS_TEST > 0): ITRS_TEST += 1 # Si no es ødivisin exacta, se
                            ñaade una øiteracin
259 
260 
261                     for step in range(ITRS_TEST):
262 
263                         batch_inputs, batch_context =
                                getBatchData(DX,DY,batch_size,step,ITRS_TEST,RES_ITMS_TEST)
264 
265                         feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
266                         loss_val,batch_lss = session.run([nce_loss,batch_loss],
                                feed_dict=feed_dict)
267 
```

```
                    TEST_LOSS+=np.sum(batch_lss)

                dev_hist.append(TEST_LOSS / (len(DX) * 1.0))
                train_hist.append(TRAIN_LOSS / (len(X) *  1.0))

            if (len(dev_hist) >= slope_size):
                # Calcular la pendiente
                print(getSlope(dev_hist[-slope_size:]))
                print(dev_hist[-slope_size:])

                if (getSlope(dev_hist[-slope_size:]) > -1e-5 or e==epochs-1):

                    def printArray(data,data2):
                        for i in range(len(data)):
                            a=data[i]
                            b=data2[i]
                            print(str(a).replace(".",",")+"\t"+str(b).replace(".",","))

                    print("-" * 50)
                    print(e)
                    printArray(dev_hist,train_hist)
                    return
    else:
        #
        # TEST FINAL
        #
        batch_size = 300

        IN_TOP_5 = 0
        IN_TOP_10 = 0
        IN_TOP_20 = 0
        IN_TOP_50 = 0
        IN_TOP_100 = 0
        IN_TOP_1000 = 0
        real_positions = []

        items = 0

        ITRS_TEST = int(len(TX) / batch_size)
        RES_ITMS_TEST = len(TX) - (ITRS_TEST * batch_size)
        if (RES_ITMS_TEST > 0): ITRS_TEST += 1  # Si no es divisin exacta, se naade una
            iteracin

        for step in range(ITRS_TEST):
            #tss = time.time()

            batch_inputs, batch_context = getBatchData(TX, TY, batch_size, step,
                ITRS_TEST, RES_ITMS_TEST)
            top_values,top_indx = session.run([top_v,top_i], feed_dict={train_dataset:
                batch_inputs, train_labels: batch_context})

            for j in range(len(batch_context)):
                real_position = int(np.where(top_indx[j, :] == batch_context[j, 0])[0])
                real_positions.append(real_position)

```

```
320                     IN_TOP_5 += 1 if real_position <= 5 else 0
321                     IN_TOP_10 += 1 if real_position <= 10 else 0
322                     IN_TOP_20 += 1 if real_position <= 20 else 0
323                     IN_TOP_50 += 1 if real_position <= 50 else 0
324                     IN_TOP_100 += 1 if real_position <= 100 else 0
325                     IN_TOP_1000 += 1 if real_position <= 1000 else 0
326 
327                 items += len(batch_inputs)*1.0
328 
329                 PTF = IN_TOP_5 / items
330                 PTT = IN_TOP_10 / items
331                 PTTW = IN_TOP_20 / items
332                 PTFT = IN_TOP_50 / items
333                 PTCIEN = IN_TOP_100 / items
334                 PTMIL = IN_TOP_1000 / items
335 
336                 print(np.mean(real_positions), np.std(real_positions),
                        np.median(real_positions))
337                 print(str(PTF).replace(".", ",") + "\t" + str(PTT).replace(".", ",") + "\t"
338                       + str(PTTW).replace(".",",") + "\t" + str(PTFT).replace(".", ",") + "\t"
339                       + str(PTCIEN).replace(".", ",") + "\t" + str(PTMIL).replace(".",","))
340 
341                 #print(time.time()-tss)
342 
343 
344             print(np.mean(real_positions), np.std(real_positions), np.median(real_positions))
345 
346         #-------------------------------------------------------------------------------------------------------------
347 
348 ##########################################################################################################################
349 # Llamadas
350 ##########################################################################################################################
351 
352 TEST_NAME="test_063"
353 
354 os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
355 
356 #Se obtinenen cada uno de los embeddings aprendidos (incluidas las canciones)
357 D2V_DM_COMPLETE   = np.load('embeddings/d2v_dm/complete/EMB_MATRIX')
358 D2V_DBOW_COMPLETE = np.load('embeddings/d2v_dbow/complete/EMB_MATRIX')
359 W2V  = np.load('embeddings/w2v/EMB_MATRIX')
360 
361 #Se concatenan los embeddings de los usuarios para generar el perfil
362 D2V_EMB = np.concatenate((D2V_DM_COMPLETE,D2V_DBOW_COMPLETE),axis=1) # El consolidado =
        Consolidado DM + Consolidado DBOW
363 
364 #Se cargan los datos de train de las canciones para conocer los IDs reales
365 DATA = np.load("datos/Extend/ARRAY_ARRAYS_TRAIN.pkl")
366 
367 #Obtener una lista con los ID de canciones que no aestn el la matriz de embeddings
368 ALL = set(list(range(W2V.shape[0])))
369 REAL = set(list(set(list(map(int, DATA)))))
370 
371 ##########################################################################################################################
372 # TENSORFLOW
373 ##########################################################################################################################
374 
```

```
#-------------------------------------------------------------------------------
# Grid Search + Early Stopping
#-------------------------------------------------------------------------------
'''
X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))
rates = [0.000001,1, 0.01, 0.0001]

for r in rates:
    config = {"batch_size": 1024, "learning_rate": r, "epochs": 1000, "seed": 100}

    print("Learning Rate: "+str(r))
    print("-" * 50)
    trainNet(X, Y, DX, DY, TX, TY, config, user_size, train=True, save=False)
    print("-"*50)

exit()
'''
#-------------------------------------------------------------------------------
# Train
#-------------------------------------------------------------------------------
'''
X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))

#Para esta fase, se une TRAIN y DEV
X = np.concatenate((X,DX),axis=0)
Y = np.concatenate((Y,DY),axis=0)

DX=[]
DY=[]

config = {"batch_size": 1024, "learning_rate": 0.01, "epochs":597, "seed":100}
trainNet(X, Y, DX, DY, TX, TY, config, user_size, train=True, save=True)

exit()
'''
#-------------------------------------------------------------------------------
# Test
#-------------------------------------------------------------------------------

X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))

config = {"batch_size": 1024, "learning_rate": 0.01, "epochs":597, "seed":100}
trainNet(X, Y, DX, DY, TX, TY, config, user_size,train=False, save=True)
```

4.23. TEST 23

```
# -*- coding: utf-8 -*-

import os.path
import os
import tensorflow as tf
import numpy as np
import random
import time
import math
import pandas as pd
import pickle
from collections import Counter
from scipy.stats import linregress
from tensorflow.python.framework import ops as framework_ops

########################################################################################################################
# @Mtodos
########################################################################################################################

def getData(REAL_SONG_IDS=[],data ="datos/Append/ARRAY_ARRAYS_REC.pkl",shuffle=True,
    seed=100, train=.7):

    DATA = np.load(data)

    #DATA = DATA[:5];print("ELIMINAR FILTRO")

    #User, cancion 1, cancion 2, cancion 3
    TRAIN = np.empty([0, 4], dtype=np.int32)
    DEV = np.empty([0, 4], dtype=np.int32)
    TEST = np.empty([0, 4], dtype=np.int32)

    for u in range(len(DATA)):

        # Eliminar canciones escuchadas que no tienen embedding
        USER_PLAYS = DATA[u]
        USER_PLAYS = pd.DataFrame(list(map(int, USER_PLAYS)))
        USER_PLAYS = USER_PLAYS.loc[USER_PLAYS[0].isin(REAL_SONG_IDS)][0].values

        #Separar en train dev test
        TRAIN_ITEMS = int(len(USER_PLAYS)*train)
        DEV_ITEMS = int((len(USER_PLAYS)-TRAIN_ITEMS)/2)

        TRAIN_DATA = USER_PLAYS[:TRAIN_ITEMS]
        DEV_DATA = USER_PLAYS[TRAIN_ITEMS:TRAIN_ITEMS+DEV_ITEMS]
        TEST_DATA = USER_PLAYS[TRAIN_ITEMS+DEV_ITEMS:]

        # Crear ejemplos
        USR_TRAIN = np.empty([len(TRAIN_DATA)-2, 4], dtype=np.int32)
        USR_TRAIN[:,0] = u
        USR_TRAIN[:,1] = TRAIN_DATA[:len(TRAIN_DATA)-2]
        USR_TRAIN[:,2] = TRAIN_DATA[1:-1]
        USR_TRAIN[:,3] = TRAIN_DATA[2:]

        USR_DEV = np.empty([len(DEV_DATA) -2, 4], dtype=np.int32)
        USR_DEV[:, 0] = u
```

```
55          USR_DEV[:, 1] = DEV_DATA[:len(DEV_DATA)-2]
56          USR_DEV[:, 2] = DEV_DATA[1:-1]
57          USR_DEV[:, 3] = DEV_DATA[2:]
58
59          USR_TEST = np.empty([len(TEST_DATA)-2, 4], dtype=np.int32)
60          USR_TEST[:, 0] = u
61          USR_TEST[:, 1] = TEST_DATA[:len(TEST_DATA) -2]
62          USR_TEST[:, 2] = TEST_DATA[1:-1]
63          USR_TEST[:, 3] = TEST_DATA[2:]
64
65          # nAadir al global
66          TRAIN = np.concatenate([TRAIN,USR_TRAIN])
67          DEV = np.concatenate([DEV,USR_DEV])
68          TEST = np.concatenate([TEST,USR_TEST])
69
70      #Mezclar
71      if(shuffle):
72          np.random.seed(seed)
73          np.random.shuffle(TRAIN)
74          np.random.shuffle(DEV)
75          np.random.shuffle(TEST)
76
77      return
            TRAIN[:,:3],np.matrix(TRAIN[:,-1]).T,DEV[:,:3],np.matrix(DEV[:,-1]).T,TEST[:,:3],np.matrix(TEST
78
79  def getBatchData(X, Y, batch_size, step, ITRS, RES_ITMS):
80
81      # Si se no aadi una iteracin, introducir los datos restantes
82      if (step == ITRS - 1 and RES_ITMS > 0):
83          batch_inputs = X[(ITRS - 1) * batch_size:]
84          batch_context = Y[(ITRS - 1) * batch_size:]
85      # Si no, es un batch normal
86      else:
87          st = step * batch_size
88          nd = (step + 1) * batch_size
89
90          batch_inputs = X[st:nd]
91          batch_context = Y[st:nd]
92      return(batch_inputs,batch_context)
93
94  def trainNet(X,Y, DX,DY,TX,TY,config, user_size, train=True, save=True):
95
96      def getSlope(data):
97          r = linregress(range(len(data)), data)
98          return r.slope
99
100     batch_size = config['batch_size']
101     learning_rate = config['learning_rate']
102     epochs = config['epochs']
103
104     user_size = user_size
105     song_size = W2V.shape[0]
106
107     user_emb_size = 128
108     song_emb_size = 64
109     hidden_size = 32
110
```

```
111     profile_size = user_emb_size + song_emb_size + song_emb_size
112
113     num_sampled = 64  # Ejemplos negativos (Para el NSE)
114
115     model_name = 'model'
116     model_path = 'models/' + TEST_NAME + '/model'
117     complete_path = model_path + "/" + model_name
118
119     # Se almacena la loss de dev de las 5 ultimas epochs
120     slope_size = 100
121     train_hist = []
122     dev_hist = []
123
124     # Creacin del grafo de TF.
125     graph = tf.Graph()
126
127     with graph.as_default():
128
129         if(config["seed"]!=1):
130             tf.set_random_seed(config["seed"])
131
132         # Nmero global de iteraciones
133         global_step = tf.Variable(0, dtype=tf.int32, trainable=False, name='global_step')
134
135         # Datos de entrada
            ------------------------------------------------------------------------------
136         # Array del tamao del batch con las X
137         train_dataset = tf.placeholder(tf.int32, shape=[None, 3], name="train_inputs")
138
139         # Array del tamao del batch con las Y asociadas
140         train_labels = tf.placeholder(tf.float32, shape=[None, 1], name="train_labels")
141
142         # Embeddings
            ------------------------------------------------------------------------------
143
144         D2V_EMB_VRB = tf.Variable(D2V_EMB, trainable=False, name="doc_embeddings")
145
146         O2 = tf.Variable(tf.truncated_normal([song_size, song_emb_size], mean=0.0, stddev=1.0
                / math.sqrt(song_emb_size)), name="O2")
147         BO2 = tf.Variable(tf.zeros([song_emb_size]), name="BO2")
148
149         # Operaciones
            ------------------------------------------------------------------------------
150
151         # Embedding de documento
152         ed = tf.nn.embedding_lookup(D2V_EMB_VRB, train_dataset[:, 0])
153
154         # Embedding de cancion 1
155         ec1 = tf.nn.embedding_lookup(O2, train_dataset[:, 1])
156         ec1 = ec1 + BO2
157
158         # Embedding de cancion 2
159         ec2 = tf.nn.embedding_lookup(O2, train_dataset[:, 2])
160         ec2 = ec2 + BO2
161
162
163         # Clculo de LOSS y optimizador
```

```
164
165         embed = []
166         embed.append(ed)
167         embed.append(ec1)
168         embed.append(ec2)
169         embed = tf.concat(embed, 1)
170
171         # softmax weights, W and D vectors should be concatenated before applying softmax
172         weights = tf.Variable(tf.truncated_normal([song_size, profile_size], stddev=1.0 /
                math.sqrt(song_size)), name="weights")
173         # softmax biases
174         biases = tf.Variable(tf.zeros([song_size]), name="biases")
175
176         # aClculo de LOSS y optimizador—
            ----------------------------------------------------------------------------------
177         nce_loss = tf.nn.nce_loss(weights=weights,
178                                   biases=biases,
179                                   labels=train_labels,
180                                   inputs=embed,
181                                   num_sampled=num_sampled,
182                                   num_classes=song_size,
183                                   name="nce_loss")
184
185         batch_loss = nce_loss
186         nce_loss = tf.reduce_mean(nce_loss)
187
188         logits = tf.matmul(embed, tf.transpose(weights))
189         logits = tf.nn.bias_add(logits, biases)
190
191         top_v,top_i = tf.nn.top_k(logits,k=song_size)
192
193         # logits = tf.nn.softmax(logits,axis=0)
194
195         # optimizer =
                tf.train.AdamOptimizer(learning_rate).minimize(nce_loss,global_step=global_step)
196         train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(nce_loss,
                global_step=global_step)
197
198         # Inicializar variables
199         init = tf.global_variables_initializer()
200
201         # Crear objeto encargado de almacenar la red
202         saver = tf.train.Saver(max_to_keep=1)
203
204     config = tf.ConfigProto()
205     config.gpu_options.allow_growth = True
206     with tf.Session(graph=graph, config=config) as session:
207     #with tf.Session(graph=graph) as session:
208
209         # We must initialize all variables before we use them.
210         init.run()
211
212
213         if(save):
214             # Obtener el checkpoint
215             ckpt = tf.train.get_checkpoint_state(model_path)
```

```
216
217             # Si existe el checkpoint restaurar
218             if ckpt and ckpt.model_checkpoint_path:
219                 saver.restore(session, ckpt.model_checkpoint_path)
220
221         #Si se desea entrenar
222         if(train):
223             #-----------------------------------------------------------------------------------------
224             # TRAIN
225             #-----------------------------------------------------------------------------------------
226
227             ITRS= int(len(X)/batch_size)
228             RES_ITMS = len(X) - (ITRS*batch_size)
229             if(RES_ITMS>0):ITRS+=1 #Si no es divisin exacta, se aade una iteracin
230
231             for e in range(epochs):
232                 TRAIN_LOSS = 0.0
233
234                 print("Epoch "+str(e))
235
236                 for step in range(ITRS):
237
238                     batch_inputs, batch_context =
                            getBatchData(X,Y,batch_size,step,ITRS,RES_ITMS)
239
240                     feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
241
242                     _, loss_val, batch_lss, gs = session.run([train_step,
                            nce_loss,batch_loss,global_step], feed_dict=feed_dict)
243
244                     TRAIN_LOSS += np.sum(batch_lss)
245
246                 #print('Epoch '+str(e)+', last batch loss: '+str(loss_val))
247
248                 if (save): saver.save(session, complete_path, global_step=global_step)
249
250                 #-----------------------------------------------------------------------------------------
251                 # DEV
252                 #-----------------------------------------------------------------------------------------
253
254                 if(len(DX)>0):
255
256                     ITRS_TEST = int(len(DX) / batch_size)
257                     RES_ITMS_TEST = len(DX) - (ITRS_TEST * batch_size)
258                     TEST_LOSS = 0.0
259                     if (RES_ITMS_TEST > 0): ITRS_TEST += 1 # Si no es divisin exacta, se
                            aade una iteracin
260
261
262                     for step in range(ITRS_TEST):
263
264                         batch_inputs, batch_context =
                                getBatchData(DX,DY,batch_size,step,ITRS_TEST,RES_ITMS_TEST)
265
266                         feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
267                         loss_val,batch_lss = session.run([nce_loss,batch_loss],
                                feed_dict=feed_dict)
```

```
268
269                     TEST_LOSS+=np.sum(batch_lss)
270
271                 dev_hist.append(TEST_LOSS / (len(DX) * 1.0))
272                 train_hist.append(TRAIN_LOSS / (len(X) *  1.0))
273
274             if (len(dev_hist) >= slope_size):
275                 # Calcular la pendiente
276                 print(getSlope(dev_hist[-slope_size:]))
277                 print(dev_hist[-slope_size:])
278
279                 if (getSlope(dev_hist[-slope_size:]) > -1e-5 or e==epochs-1):
280
281                     def printArray(data,data2):
282                         for i in range(len(data)):
283                             a=data[i]
284                             b=data2[i]
285                             print(str(a).replace(".",",")+"\t"+str(b).replace(".",","))
286
287                     print("-" * 50)
288                     print(e)
289                     printArray(dev_hist,train_hist)
290                     return
291         else:
292             #
                -----------------------------------------------------------------------------------------------
293             # TEST FINAL
294             #
                -----------------------------------------------------------------------------------------------
295             batch_size = 300
296
297             IN_TOP_5 = 0
298             IN_TOP_10 = 0
299             IN_TOP_20 = 0
300             IN_TOP_50 = 0
301             IN_TOP_100 = 0
302             IN_TOP_1000 = 0
303             real_positions = []
304
305             items = 0
306
307             ITRS_TEST = int(len(TX) / batch_size)
308             RES_ITMS_TEST = len(TX) - (ITRS_TEST * batch_size)
309             if (RES_ITMS_TEST > 0): ITRS_TEST += 1  # Si no es @divisin exacta, se ñaade una
                    @iteracin
310
311             for step in range(ITRS_TEST):
312                 #tss = time.time()
313
314                 batch_inputs, batch_context = getBatchData(TX, TY, batch_size, step,
                        ITRS_TEST, RES_ITMS_TEST)
315                 top_values,top_indx = session.run([top_v,top_i], feed_dict={train_dataset:
                        batch_inputs, train_labels: batch_context})
316
317                 for j in range(len(batch_context)):
318                     real_position = int(np.where(top_indx[j, :] == batch_context[j, 0])[0])
319                     real_positions.append(real_position)
```

```
320
321                     IN_TOP_5 += 1 if real_position <= 5 else 0
322                     IN_TOP_10 += 1 if real_position <= 10 else 0
323                     IN_TOP_20 += 1 if real_position <= 20 else 0
324                     IN_TOP_50 += 1 if real_position <= 50 else 0
325                     IN_TOP_100 += 1 if real_position <= 100 else 0
326                     IN_TOP_1000 += 1 if real_position <= 1000 else 0
327
328                 items += len(batch_inputs)*1.0
329
330                 PTF = IN_TOP_5 / items
331                 PTT = IN_TOP_10 / items
332                 PTTW = IN_TOP_20 / items
333                 PTFT = IN_TOP_50 / items
334                 PTCIEN = IN_TOP_100 / items
335                 PTMIL = IN_TOP_1000 / items
336
337                 print(np.mean(real_positions), np.std(real_positions),
                        np.median(real_positions))
338                 print(str(PTF).replace(".", ",") + "\t" + str(PTT).replace(".", ",") + "\t"
339                       + str(PTTW).replace(".",",") + "\t" + str(PTFT).replace(".", ",") + "\t"
340                       + str(PTCIEN).replace(".", ",") + "\t" + str(PTMIL).replace(".",","))
341
342                 #print(time.time()-tss)
343
344
345             print(np.mean(real_positions), np.std(real_positions), np.median(real_positions))
346
347         #-----------------------------------------------------------------------------------------------------------
348
349 ##########################################################################################################################
350 # Llamadas
351 ##########################################################################################################################
352
353 TEST_NAME="test_073"
354
355 os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
356
357 #Se obtinenen cada uno de los embeddings aprendidos (incluidas las canciones)
358 D2V_DM_PRESENT   = np.load('embeddings/d2v_dm/present/EMB_MATRIX')
359 D2V_DBOW_PRESENT  = np.load('embeddings/d2v_dbow/present/EMB_MATRIX')
360 W2V  = np.load('embeddings/w2v/EMB_MATRIX')
361
362 #Se concatenan los embeddings de los usuarios para generar el perfil
363 D2V_EMB = np.concatenate((D2V_DM_PRESENT,D2V_DBOW_PRESENT),axis=1) # El reciente = Reciente
    DM + Reciente DBOW
364
365 #Se cargan los datos de train de las canciones para conocer los IDs reales
366 DATA = np.load("datos/Extend/ARRAY_ARRAYS_TRAIN.pkl")
367
368 #Obtener una lista con los ID de canciones que no aestn el la matriz de embeddings
369 ALL = set(list(range(W2V.shape[0])))
370 REAL = set(list(set(list(map(int, DATA)))))
371
372 ##########################################################################################################################
373 # TENSORFLOW
374 ##########################################################################################################################
```

```
375
376 #-------------------------------------------------------------------------------
377 # Grid Search + Early Stopping
378 #-------------------------------------------------------------------------------
379 '''
380 X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))
381 rates = [0.000001,1, 0.01, 0.0001]
382
383 for r in rates:
384     config = {"batch_size": 1024, "learning_rate": r, "epochs": 1000, "seed": 100}
385
386     print("Learning Rate: "+str(r))
387     print("-" * 50)
388     trainNet(X, Y, DX, DY, TX, TY, config, user_size, train=True, save=False)
389     print("-"*50)
390
391 exit()
392 '''
393 #-------------------------------------------------------------------------------
394 # Train
395 #-------------------------------------------------------------------------------
396 '''
397 X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))
398
399 #Para esta fase, se une TRAIN y DEV
400 X = np.concatenate((X,DX),axis=0)
401 Y = np.concatenate((Y,DY),axis=0)
402
403 DX=[]
404 DY=[]
405
406 config = {"batch_size": 1024, "learning_rate": 0.01, "epochs":1000, "seed":100}
407 trainNet(X, Y, DX, DY, TX, TY, config, user_size, train=True, save=True)
408
409 exit()
410 '''
411 #-------------------------------------------------------------------------------
412 # Test
413 #-------------------------------------------------------------------------------
414
415 X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))
416
417 config = {"batch_size": 1024, "learning_rate": 0.01, "epochs":1000, "seed":100}
418 trainNet(X, Y, DX, DY, TX, TY, config, user_size,train=False, save=True)
```

4.24. TEST 24

```
# -*- coding: utf-8 -*-

import os.path
import os
import tensorflow as tf
import numpy as np
import random
import time
import math
import pandas as pd
import pickle
from collections import Counter
from scipy.stats import linregress
from tensorflow.python.framework import ops as framework_ops

########################################################################################################################
# @Mtodos
########################################################################################################################

def getData(REAL_SONG_IDS=[],data ="datos/Append/ARRAY_ARRAYS_REC.pkl",shuffle=True, seed=100, train=.7):

    DATA = np.load(data)

    #DATA = DATA[:5];print("ELIMINAR FILTRO")

    #User, cancion 1, cancion 2, cancion 3
    TRAIN = np.empty([0, 4], dtype=np.int32)
    DEV = np.empty([0, 4], dtype=np.int32)
    TEST = np.empty([0, 4], dtype=np.int32)

    for u in range(len(DATA)):

        # Eliminar canciones escuchadas que no tienen embedding
        USER_PLAYS = DATA[u]
        USER_PLAYS = pd.DataFrame(list(map(int, USER_PLAYS)))
        USER_PLAYS = USER_PLAYS.loc[USER_PLAYS[0].isin(REAL_SONG_IDS)][0].values

        #Separar en train dev test
        TRAIN_ITEMS = int(len(USER_PLAYS)*train)
        DEV_ITEMS = int((len(USER_PLAYS)-TRAIN_ITEMS)/2)

        TRAIN_DATA = USER_PLAYS[:TRAIN_ITEMS]
        DEV_DATA = USER_PLAYS[TRAIN_ITEMS:TRAIN_ITEMS+DEV_ITEMS]
        TEST_DATA = USER_PLAYS[TRAIN_ITEMS+DEV_ITEMS:]

        # Crear ejemplos
        USR_TRAIN = np.empty([len(TRAIN_DATA)-2, 4], dtype=np.int32)
        USR_TRAIN[:,0] = u
        USR_TRAIN[:,1] = TRAIN_DATA[:len(TRAIN_DATA)-2]
        USR_TRAIN[:,2] = TRAIN_DATA[1:-1]
        USR_TRAIN[:,3] = TRAIN_DATA[2:]

        USR_DEV = np.empty([len(DEV_DATA) -2, 4], dtype=np.int32)
        USR_DEV[:, 0] = u
```

```
        USR_DEV[:, 1] = DEV_DATA[:len(DEV_DATA)-2]
        USR_DEV[:, 2] = DEV_DATA[1:-1]
        USR_DEV[:, 3] = DEV_DATA[2:]

        USR_TEST = np.empty([len(TEST_DATA)-2, 4], dtype=np.int32)
        USR_TEST[:, 0] = u
        USR_TEST[:, 1] = TEST_DATA[:len(TEST_DATA) -2]
        USR_TEST[:, 2] = TEST_DATA[1:-1]
        USR_TEST[:, 3] = TEST_DATA[2:]

        # nAadir al global
        TRAIN = np.concatenate([TRAIN,USR_TRAIN])
        DEV = np.concatenate([DEV,USR_DEV])
        TEST = np.concatenate([TEST,USR_TEST])

    #Mezclar
    if(shuffle):
        np.random.seed(seed)
        np.random.shuffle(TRAIN)
        np.random.shuffle(DEV)
        np.random.shuffle(TEST)

    return
        TRAIN[:,:3],np.matrix(TRAIN[:,-1]).T,DEV[:,:3],np.matrix(DEV[:,-1]).T,TEST[:,:3],np.matrix(TEST

def getBatchData(X, Y, batch_size, step, ITRS, RES_ITMS):

    # Si se no aadi una iteracin, introducir los datos restantes
    if (step == ITRS - 1 and RES_ITMS > 0):
        batch_inputs = X[(ITRS - 1) * batch_size:]
        batch_context = Y[(ITRS - 1) * batch_size:]
    # Si no, es un batch normal
    else:
        st = step * batch_size
        nd = (step + 1) * batch_size

        batch_inputs = X[st:nd]
        batch_context = Y[st:nd]
    return(batch_inputs,batch_context)

def trainNet(X,Y, DX,DY,TX,TY,config, user_size, train=True, save=True):

    def getSlope(data):
        r = linregress(range(len(data)), data)
        return r.slope

    batch_size = config['batch_size']
    learning_rate = config['learning_rate']
    epochs = config['epochs']

    user_size = user_size
    song_size = W2V.shape[0]

    user_emb_size = 128
    song_emb_size = 64
    hidden_size = 32

```

```
111     profile_size = user_emb_size + song_emb_size + song_emb_size
112     user_concat_size = user_emb_size * 2
113 
114     num_sampled = 64  # Ejemplos negativos (Para el NSE)
115 
116     model_name = 'model'
117     model_path = 'models/' + TEST_NAME + '/model'
118     complete_path = model_path + "/" + model_name
119 
120     # Se almacena la loss de dev de las 5 ultimas epochs
121     slope_size = 100
122     train_hist = []
123     dev_hist = []
124 
125     # Creacin del grafo de TF.
126     graph = tf.Graph()
127 
128     with graph.as_default():
129 
130         if(config["seed"]!=1):
131             tf.set_random_seed(config["seed"])
132 
133         # Nmero global de iteraciones
134         global_step = tf.Variable(0, dtype=tf.int32, trainable=False, name='global_step')
135 
136         # Datos de entrada
            ------------------------------------------------------------------------------------------
137         # Array del tamao del batch con las X
138         train_dataset = tf.placeholder(tf.int32, shape=[None, 3], name="train_inputs")
139 
140         # Array del tamao del batch con las Y asociadas
141         train_labels = tf.placeholder(tf.float32, shape=[None, 1], name="train_labels")
142 
143         # Embeddings
            ------------------------------------------------------------------------------------------
144 
145         D2V_EMB_VRB = tf.Variable(D2V_EMB, trainable=False, name="doc_embeddings")
146 
147         T0 = tf.Variable(tf.truncated_normal([user_concat_size, user_emb_size], mean=0.0,
                stddev=1.0 / math.sqrt(hidden_size)), name="T0")
148         B0 = tf.Variable(tf.zeros([user_emb_size]), name="B0")
149 
150         O2 = tf.Variable(tf.truncated_normal([song_size, song_emb_size], mean=0.0, stddev=1.0
                / math.sqrt(song_emb_size)), name="O2")
151         BO2 = tf.Variable(tf.zeros([song_emb_size]), name="BO2")
152 
153         # Operaciones
            ------------------------------------------------------------------------------------------
154 
155         # Embedding de documento
156         ed = tf.nn.embedding_lookup(D2V_EMB_VRB, train_dataset[:, 0])
157 
158         #Transformar concat documento de 256 a 128
159         ed = tf.matmul(ed,T0)+B0
160 
161         # Embedding de cancion 1
162         ec1 = tf.nn.embedding_lookup(O2, train_dataset[:, 1])
```

```
        ec1 = ec1 + BO2

        # Embedding de cancion 2
        ec2 = tf.nn.embedding_lookup(O2, train_dataset[:, 2])
        ec2 = ec2 + BO2

        # aClculo de LOSS y optimizador—
            ------------------------------------------------------------------------------------------

        embed = []
        embed.append(ed)
        embed.append(ec1)
        embed.append(ec2)
        embed = tf.concat(embed, 1)

        # softmax weights, W and D vectors should be concatenated before applying softmax
        weights = tf.Variable(tf.truncated_normal([song_size, profile_size], stddev=1.0 /
            math.sqrt(song_size)), name="weights")
        # softmax biases
        biases = tf.Variable(tf.zeros([song_size]), name="biases")

        # aClculo de LOSS y optimizador—
            ------------------------------------------------------------------------------------------
        nce_loss = tf.nn.nce_loss(weights=weights,
                                  biases=biases,
                                  labels=train_labels,
                                  inputs=embed,
                                  num_sampled=num_sampled,
                                  num_classes=song_size,
                                  name="nce_loss")

        batch_loss = nce_loss
        nce_loss = tf.reduce_mean(nce_loss)

        logits = tf.matmul(embed, tf.transpose(weights))
        logits = tf.nn.bias_add(logits, biases)

        top_v,top_i = tf.nn.top_k(logits,k=song_size)

        # logits = tf.nn.softmax(logits,axis=0)

        # optimizer =
            tf.train.AdamOptimizer(learning_rate).minimize(nce_loss,global_step=global_step)
        train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(nce_loss,
            global_step=global_step)

        # Inicializar variables
        init = tf.global_variables_initializer()

        # Crear objeto encargado de almacenar la red
        saver = tf.train.Saver(max_to_keep=1)

    config = tf.ConfigProto()
    config.gpu_options.allow_growth = True
    with tf.Session(graph=graph, config=config) as session:
    #with tf.Session(graph=graph) as session:

```

```
215         # We must initialize all variables before we use them.
216         init.run()
217
218
219         if(save):
220             # Obtener el checkpoint
221             ckpt = tf.train.get_checkpoint_state(model_path)
222
223             # Si existe el checkpoint restaurar
224             if ckpt and ckpt.model_checkpoint_path:
225                 saver.restore(session, ckpt.model_checkpoint_path)
226
227         #Si se desea entrenar
228         if(train):
229             #-----------------------------------------------------------------------------------------
230             # TRAIN
231             #-----------------------------------------------------------------------------------------
232
233             ITRS= int(len(X)/batch_size)
234             RES_ITMS = len(X) - (ITRS*batch_size)
235             if(RES_ITMS>0):ITRS+=1 #Si no es divisin exacta, se naade una iteracin
236
237             for e in range(epochs):
238                 TRAIN_LOSS = 0.0
239
240                 print("Epoch "+str(e))
241
242                 for step in range(ITRS):
243
244                     batch_inputs, batch_context =
                            getBatchData(X,Y,batch_size,step,ITRS,RES_ITMS)
245
246                     feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
247
248                     _, loss_val, batch_lss, gs = session.run([train_step,
                            nce_loss,batch_loss,global_step], feed_dict=feed_dict)
249
250                     TRAIN_LOSS += np.sum(batch_lss)
251
252                 #print('Epoch '+str(e)+', last batch loss: '+str(loss_val))
253
254                 if (save): saver.save(session, complete_path, global_step=global_step)
255
256                 #-----------------------------------------------------------------------------------
257                 # DEV
258                 #-----------------------------------------------------------------------------------
259
260                 if(len(DX)>0):
261
262                     ITRS_TEST = int(len(DX) / batch_size)
263                     RES_ITMS_TEST = len(DX) - (ITRS_TEST * batch_size)
264                     TEST_LOSS = 0.0
265                     if (RES_ITMS_TEST > 0): ITRS_TEST += 1 # Si no es divisin exacta, se
                            naade una iteracin
266
267
268                     for step in range(ITRS_TEST):
```

```
269
270                     batch_inputs, batch_context =
                            getBatchData(DX,DY,batch_size,step,ITRS_TEST,RES_ITMS_TEST)
271
272                     feed_dict = {train_dataset: batch_inputs, train_labels: batch_context}
273                     loss_val,batch_lss = session.run([nce_loss,batch_loss],
                            feed_dict=feed_dict)
274
275                     TEST_LOSS+=np.sum(batch_lss)
276
277                 dev_hist.append(TEST_LOSS / (len(DX) * 1.0))
278                 train_hist.append(TRAIN_LOSS / (len(X) *  1.0))
279
280             if (len(dev_hist) >= slope_size):
281                 # Calcular la pendiente
282                 print(getSlope(dev_hist[-slope_size:]))
283                 print(dev_hist[-slope_size:])
284
285                 if (getSlope(dev_hist[-slope_size:]) > -1e-5 or e==epochs-1):
286
287                     def printArray(data,data2):
288                         for i in range(len(data)):
289                             a=data[i]
290                             b=data2[i]
291                             print(str(a).replace(".",",")+"\t"+str(b).replace(".",","))
292
293                     print("-" * 50)
294                     print(e)
295                     printArray(dev_hist,train_hist)
296                     return
297         else:
298             #
                ------------------------------------------------------------------------------
299             # TEST FINAL
300             #
                ------------------------------------------------------------------------------
301             batch_size = 300
302
303             IN_TOP_5 = 0
304             IN_TOP_10 = 0
305             IN_TOP_20 = 0
306             IN_TOP_50 = 0
307             IN_TOP_100 = 0
308             IN_TOP_1000 = 0
309             real_positions = []
310
311             items = 0
312
313             ITRS_TEST = int(len(TX) / batch_size)
314             RES_ITMS_TEST = len(TX) - (ITRS_TEST * batch_size)
315             if (RES_ITMS_TEST > 0): ITRS_TEST += 1  # Si no es divisin exacta, se aade una
                    iteracin
316
317             for step in range(ITRS_TEST):
318                 #tss = time.time()
319
320                 batch_inputs, batch_context = getBatchData(TX, TY, batch_size, step,
```

```
                    ITRS_TEST, RES_ITMS_TEST)
321                 top_values,top_indx = session.run([top_v,top_i], feed_dict={train_dataset:
                        batch_inputs, train_labels: batch_context})
322
323                 for j in range(len(batch_context)):
324                     real_position = int(np.where(top_indx[j, :] == batch_context[j, 0])[0])
325                     real_positions.append(real_position)
326
327                     IN_TOP_5 += 1 if real_position <= 5 else 0
328                     IN_TOP_10 += 1 if real_position <= 10 else 0
329                     IN_TOP_20 += 1 if real_position <= 20 else 0
330                     IN_TOP_50 += 1 if real_position <= 50 else 0
331                     IN_TOP_100 += 1 if real_position <= 100 else 0
332                     IN_TOP_1000 += 1 if real_position <= 1000 else 0
333
334                 items += len(batch_inputs)*1.0
335
336                 PTF = IN_TOP_5 / items
337                 PTT = IN_TOP_10 / items
338                 PTTW = IN_TOP_20 / items
339                 PTFT = IN_TOP_50 / items
340                 PTCIEN = IN_TOP_100 / items
341                 PTMIL = IN_TOP_1000 / items
342
343                 print(np.mean(real_positions), np.std(real_positions),
                        np.median(real_positions))
344                 print(str(PTF).replace(".", ",") + "\t" + str(PTT).replace(".", ",") + "\t"
345                       + str(PTTW).replace(".",",") + "\t" + str(PTFT).replace(".", ",") + "\t"
346                       + str(PTCIEN).replace(".", ",") + "\t" + str(PTMIL).replace(".",","))
347
348                 #print(time.time()-tss)
349
350
351             print(np.mean(real_positions), np.std(real_positions), np.median(real_positions))
352
353         #-------------------------------------------------------------------------------------------------------------
354
355 ##########################################################################################################################
356 # Llamadas
357 ##########################################################################################################################
358
359 TEST_NAME="test_083"
360
361 os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
362
363 #Se obtinenen cada uno de los embeddings aprendidos (incluidas las canciones)
364 D2V_DM_COMPLETE   = np.load('embeddings/d2v_dm/complete/EMB_MATRIX')
365 D2V_DBOW_COMPLETE = np.load('embeddings/d2v_dbow/complete/EMB_MATRIX')
366
367 #Se obtinenen cada uno de los embeddings aprendidos (incluidas las canciones)
368 D2V_DM_PRESENT  = np.load('embeddings/d2v_dm/present/EMB_MATRIX')
369 D2V_DBOW_PRESENT = np.load('embeddings/d2v_dbow/present/EMB_MATRIX')
370
371 W2V  = np.load('embeddings/w2v/EMB_MATRIX')
372
373 #Se concatenan los embeddings de los usuarios para generar el perfil
374 D2V_CONS = np.concatenate((D2V_DM_COMPLETE,D2V_DBOW_COMPLETE),axis=1) # El consolidado =
```

```
    Consolidado DM + Consolidado DBOW
D2V_PRES = np.concatenate((D2V_DM_PRESENT,D2V_DBOW_PRESENT),axis=1) # El presente = Presente
    DM + Presente DBOW

D2V_EMB = np.concatenate((D2V_CONS,D2V_PRES),axis=1) # El perfil completo CONS + PRES

#Se cargan los datos de train de las canciones para conocer los IDs reales
DATA = np.load("datos/Extend/ARRAY_ARRAYS_TRAIN.pkl")

#Obtener una lista con los ID de canciones que no aestn el la matriz de embeddings
ALL = set(list(range(W2V.shape[0])))
REAL = set(list(set(list(map(int, DATA)))))

#############################################################################################################
# TENSORFLOW
#############################################################################################################

#------------------------------------------------------------------------------------------------------------
# Grid Search + Early Stopping
#------------------------------------------------------------------------------------------------------------
'''
X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))
rates = [0.000001,1, 0.01, 0.0001]

for r in rates:
    config = {"batch_size": 1024, "learning_rate": r, "epochs": 1000, "seed": 100}

    print("Learning Rate: "+str(r))
    print("-" * 50)
    trainNet(X, Y, DX, DY, TX, TY, config, user_size, train=True, save=False)
    print("-"*50)

exit()
'''
#------------------------------------------------------------------------------------------------------------
# Train
#------------------------------------------------------------------------------------------------------------
'''
X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))

#Para esta fase, se une TRAIN y DEV
X = np.concatenate((X,DX),axis=0)
Y = np.concatenate((Y,DY),axis=0)

DX=[]
DY=[]

config = {"batch_size": 1024, "learning_rate": 0.0001, "epochs":1000, "seed":100}
trainNet(X, Y, DX, DY, TX, TY, config, user_size, train=True, save=True)

exit()
'''
#------------------------------------------------------------------------------------------------------------
# Test
#------------------------------------------------------------------------------------------------------------

X, Y, DX, DY, TX, TY, user_size = getData(REAL_SONG_IDS=list(REAL))
```

```
430
431 config = {"batch_size": 1024, "learning_rate": 0.0001, "epochs":1000, "seed":100}
432 trainNet(X, Y, DX, DY, TX, TY, config, user_size,train=False, save=True)
```