Universidad de Oviedo

Departamento de Informática
Centro de Inteligencia Artificial

Métodos de inferencia basados en búsqueda heurística para cadenas de clasificadores probabilísticos

Tesis Doctoral

Autor:
Deiner Mena Waldo

Directores:
Dr. Juan José Del Coz Velasco
Dra. Elena Montañés Roces

2017

A mi familia

Agradecimientos

Quiero agradecerle enormemente a mis directores de tesis Elena Montañés Roces y Juan José Del Coz Velasco y a mi asesor José Ramón Quevedo Pérez por el apoyo continuo en todo el proceso de formación y realización de esta tesis. Sin duda, sin ellos esta investigación no hubiera sido posible. De igual manera, quiero agradecer a cada uno de los miembros del Centro de Inteligencia Artificial (CIA) de la Universidad de Oviedo el apoyo recibido.

Quiero darle un agradecimiento especial a mi familia por el apoyo incondicional y larga espera de mi regreso a Colombia. Sin él no hubiera sido posible todo este proceso de formación.

De igual manera, quiero agradecerle a cada uno de los miembros de la Universidad Tecnológica del Chocó su participación directa o indirectamente en mi proceso de formación académica y profesional. A su Rector por su compromiso de vincularme como docente a esta institución tras terminar esta investigación.

Esta investigación ha sido financiada por:

1. Ministerio español de Economía y Competitividad (MINECO) mediante las subvenciones TIN2011-23558 y TIN2015-65069-C2-2-R.

2. Fondo Europeo de Desarrollo Regional (FEDER) mediante la subvención TIN2015-65069-C2-2-R.

Resumen

En la clasificación multietiqueta, una instancia puede ser etiquetada con varias de las clases o etiquetas de un conjunto predefinido, frente a la clasificación multiclase en la que solo es posible asignar una de esas clases. Varios problemas reales caen en este tipo de tarea de aprendizaje, por ejemplo, asignación de etiquetas a recursos en redes sociales, detección de objetos en imágenes o diagnóstico médico. Por lo general, estas etiquetas no suelen ocurrir de manera aislada, sino que, al contrario, la presencia de una etiqueta puede depender de la probabilidad con la que otras se produzcan. De hecho, la existencia de esta dependencia es la que ha despertado gran interés entre los investigadores de este campo, estudiando varios tipos de relaciones entre las etiquetas y la manera de explotarlas, pero con mayor atención en la dependencia condicional, que captura la probabilidad de que ciertas etiquetas sean relevantes a partir de la descripción de cada instancia específica.

El método *Probabilistic Classifier Chains* (PCC) ha ganado interés debido a su prometedora propiedad de estimar la probabilidad conjunta condicional, pudiendo así obtener predicciones óptimas en términos de la conocida medida *subset 0/1 loss*. Sin embargo, el método PCC original sufre de un alto coste computacional, debido a que realiza una búsqueda exhaustiva que explora todas las posibles soluciones para finalmente quedarse con la mejor. Es por ello por lo que algunos investigadores han centrado sus estudios en diseñar alternativas de inferencia eficientes que eviten explorar todas las posibles soluciones. El algoritmo ϵ-*Approximate* (ϵ-A), el algoritmo *Beam Search* (BS) y los métodos basados en *Monte Carlo* son técnicas que han surgido a raíz de todas estas investigaciones, pero solamente el algoritmo ϵ-A con $\epsilon = 0$ garantiza alcanzar teóricamente una solución óptima en términos de *subset 0/1 loss*.

Este trabajo de investigación continua en esta línea estudiando la posibilidad de utilizar el algoritmo A* como método alternativo para realizar inferencia en PCC. Su interés radica en que este algoritmo proporciona soluciones óptimas si el heurístico de búsqueda es admisible. Por ello, se proponen dos familias de heurísticos admisibles que garantizan que A*, al igual que ϵ-A con $\epsilon = 0$, alcanza soluciones óptimas en términos de la medida *subset 0/1 loss*. Así mismo, el diseño de estos heurísticos ha sido cuidadoso para que además de ser admisibles, sean lo más dominantes posibles para permitir obtener soluciones óptimas sin explorar

demasiadas soluciones.

La primera familia de heurísticos diseñada solo es aplicable cuando los clasificadores base son lineales. Esta asunción de linealidad se realiza en un intento por conseguir una solución de compromiso entre, un heurístico admisible que pueda calcularse con la máxima profundidad para incrementar su dominancia, pero que a la vez sea computacionalmente tratable. Sin embargo, los primeros experimentos demostraron que el algoritmo de inferencia resultante no era competitivo frente a otros métodos cuando el heurístico se calculaba para la máxima profundidad, debido precisamente al propio cómputo del heurístico. Por ello, se incluyó un parámetro de profundidad que nos permite controlar su nivel de dominancia y su complejidad computacional, logrando con ello que la inferencia mediante el algoritmo A* sea computacionalmente competitiva. La segunda familia de heurísticos surge con el fin de evitar la limitación de la primera, cuyos heurísticos solamente eran viables para clasificadores base lineales. La propuesta consiste en realizar una búsqueda exhaustiva pero de profundidad limitada por un parámetro, que de nuevo reduce su dominancia, pero permite que computacionalmente sea viable.

Por otro lado, y con el fin de proporcionar información acerca del potencial del algoritmo A* con las familias de heurísticos propuestas se realizó previamente un exhaustivo estudio de las propiedades, estrategias de búsqueda y efecto de variación de parámetros de los métodos ya existentes en la literatura que también llevan a cabo inferencia en PCC.

Los experimentos realizados sobre varios conjuntos de datos de referencia muestran que el algoritmo A* con las familias de heurísticos propuestos alcanzan predicciones óptimas en términos de *subset 0/1 loss* al igual que el algoritmo ϵ-A con $\epsilon = 0$, pero explora menos nodos que éste y otros métodos de la literatura. A pesar de esto, cuando la búsqueda es altamente dirigida no son tan rápidos como el algoritmo ϵ-A con $\epsilon = 0$, hecho debido al coste del cálculo del heurístico. Sin embargo, muestran su potencial cuando los conjuntos de datos presentan más incertidumbre, logrando en estos casos hacer predicciones más rápidas que el resto de métodos existentes en la literatura.

Abstract

In multi-label classification, a subset of a predefined set of labels or classes can be assigned to an instance unlike multi-class classification where just one class can be assigned. Several real problems are included into this type of learning task. For example, assigning tags to resources in social networks, detecting objects in images or medical diagnosis. In general, these labels do not usually occur in isolation, but, at contrary, the presence of a label may depend on the likelihood that others occur. In fact, this dependence between tags has aroused great interest among researchers in this field. Indeed, there already exist several studies about different types of relationships between tags and about ways of exploiting them, paying especial attention to conditional dependence, which captures the probability that certain tags are relevant from the description of a specific instance.

The Probabilistic Classifier Chains (PCC) method has gained interest because of its promising property of being able to estimate the conditional joint probability, thus obtaining optimal predictions in terms of the well-known measure subset 0/1 loss. However, the original PCC method suffers from a high computational cost, because it performs an exhaustive search that explores all possible solutions in order to take one of the optimal ones. That is the reason why some researchers have focused their studies on designing efficient inference alternatives that avoid exploring such amount of solutions. The ϵ-Approximate (ϵ-A) algorithm, the Beam Search (BS) algorithm and the Monte Carlo sampling methods are techniques that have emerged from all these efforts, but only the ϵ-A algorithm with $\epsilon = 0$ theoretically guarantees reaching an optimal solution in terms of subset 0/1 loss.

This research work continues in this line, studying the possibility of using the algorithm A* as an alternative method to perform inference in PCC. Its interest lies in the fact that this algorithm guarantees optimal solutions if the search heuristic is admissible. Therefore, this work is not only intended to formulate the problem in terms of the algorithm A*, but to design admissible heuristics. In fact, it gives rise to two families of admissible heuristics that make the algorithm A* reach optimal solutions in terms of the subset 0/1 loss measure as ϵ-A algorithm with $\epsilon = .0$ does. Also, the design of these families of heuristics has been careful in order, not only to make the heuristics be admissible, but also to make them as most dominant as possible, so that optimal solutions are reached without exploring too many solutions previously.

The first family of heuristics designed is just suitable when the base classifiers are linear. This linearity assumption intends to obtain a trade-off solution between an admissible heuristic that can be computed for the maximum depth, increasing its dominance, and a computationally tractable heuristic. However, some preliminary experiments showed that such heuristic is not competitive when it estimates the total cost from a node to the leaves, since its calculation is quite costly. That is the reason of including a depth parameter that allow to control the dominance of the heuristic and its computational cost, making the resulting inference method based on A* algorithm more competitive in computational time. The second family of heuristics arises in order to avoid the limitation of the first family of just being viable for linear base classifiers. The proposal is to carry out an exhaustive search, but limiting the depth of the search by a parameter in order to make the heuristic be computationally viable, although at the cost of reducing its dominance.

Previously, and in order to provide information about the potential of the algorithm A* with the families of heuristics proposed in this work, we carried out an exhaustive study of the properties, search strategies and the influence of the parameters of other existing methods in the literature for inference in PCC.

Experiments performed on several benchmark datasets show that the algorithm A* with the proposed families of heuristics reaches optimum predictions in terms of subset 0/1 loss, as ϵ-A algorithm with $\epsilon = .0$ does, but it explores fewer nodes than it and other methods in the literature. Despite this fact, it is not so fast as the ϵ-A algorithm with $\epsilon = .0$ when the search is highly directed. However, they show their potential when the data sets present more uncertainty, providing in this case faster predictions than other existing methods in the literature.

Índice general

Capítulo 1

Introducción

La clasificación multietiqueta (MLC, *Multi-label Classification*) es un problema de aprendizaje que consiste en asignar un subconjunto de etiquetas (clases) a cada instancia, a diferencia de la clasificación convencional que implica predecir solo una clase. Los problemas de MLC ocurren de manera natural en múltiples aplicaciones, por ejemplo, al asignar: palabras clave a un artículo, etiquetas a recursos de una red social, objetos a imágenes o expresiones emocionales a rostros humanos.

En general, el aprendizaje multietiqueta presenta tres desafíos fundamentales:

(i) El primer problema está relacionado con el origen de los datos, debido a que normalmente se parte de datos que son etiquetados manualmente por expertos de manera subjetiva, donde diferentes expertos pueden etiquetar el mismo objeto de distintas formas. Esto conlleva a que habitualmente se trabaje con datos en los que su etiquetado no es del todo perfecto y, por ende, los algoritmos deberán enfrentarse a esta situación.

(ii) El segundo problema se refiere a la complejidad computacional de los algoritmos. Si el número de etiquetas es grande, entonces un enfoque complejo podría no ser aplicable en la práctica. En consecuencia, la escalabilidad de los algoritmos es un tema clave en este campo ya que la complejidad de los métodos aumenta con el número de etiquetas.

(iii) El tercer problema se relaciona con la naturaleza propia de los datos con múltiples etiquetas. No solo el número de etiquetas suele ser grande, sino que cada instancia puede estar etiquetada por un subconjunto de etiquetas de tamaño variable. Adicionalmente, las etiquetas normalmente no aparecen de forma independiente entre ellas, sino que, lo más natural es que presenten dependencias estadísticas. Desde el punto de vista del aprendizaje y la predicción, las dependencias estadísticas entre las etiquetas constituyen una fuente de información que es posible explotar. Por lo tanto, no es de extrañar que la investigación sobre MLC se haya centrado en el diseño de

nuevos métodos capaces de detectar y beneficiarse de las interdependencias entre las etiquetas. En este sentido, se distinguen formalmente dos tipos de dependencia, la condicional y la marginal (incondicional) Dembczyński et al. [2012]; Gibaja and Ventura [2014]. Por un lado, la dependencia condicional captura las dependencias entre las etiquetas dada una instancia específica, reflejando la probabilidad de que ciertas etiquetas sean a la vez relevantes a partir de la descripción de una instancia específica. Por otro lado, la dependencia incondicional es independiente de una instancia específica y se refiere a la probabilidad de que ciertas etiquetas sean relevantes.

Se han propuesto ya varios métodos para hacer frente a los desafíos de MLC. En primer lugar, los investigadores trataron de adaptar y extender diferentes algoritmos de clasificación binaria o multiclase Elisseeff and Weston [2005]; McCallum [1999]; Zhang and Zhou [2007]. En segundo lugar, analizaron con mayor profundidad la dependencia de etiquetas e intentaron diseñar nuevos métodos que explotaran la correlación de etiquetas Dembczyński et al. [2010a]; Montañés et al. [2011a, 2014]; Read et al. [2011] y la dependencia marginal (incondicional) Cheng and Hüllermeier [2009]. Además, se han estudiado relaciones entre pares de etiquetas Elisseeff and Weston [2005], relaciones en conjuntos de diferentes tamaños Read et al. [2011]; Tsoumakas and Vlahavas [2007] y relaciones en el conjunto total de etiquetas Cheng and Hüllermeier [2009]; Montañés et al. [2011a].

En cuanto a la dependencia condicional de las etiquetas, el método *Probabilistic Classifier Chains* (PCC) ha suscitado gran interés entre la comunidad multietiqueta, ya que ofrece la excelente propiedad de poder estimar la distribución conjunta condicional de las etiquetas, pudiendo optimizarse la conocida medida *subset 0/1 loss*. Sin embargo, el algoritmo PCC original Dembczyński et al. [2010a] sufre de un alto coste computacional, debido a que realiza una búsqueda exhaustiva (ES - *Exhaustive Search*) como estrategia de inferencia para obtener soluciones óptimas en términos de una función de pérdida dada. Para mitigar este inconveniente los investigadores han desarrollado algunos algoritmos entre los que se encuentra el método basado en búsqueda voraz (GS - *Greedy Search*), originalmente llamado CC Read et al. [2011]. Este algoritmo explora un solo camino siguiendo en cada nodo la rama con la probabilidad condicional marginal más alta. Aunque este método intente optimizar la medida *subset 0/1 loss*, un estudio previo Dembczynski et al. [2012] demuestra que obtiene resultados pobres y en general no alcanza una solución óptima. Un método más eficiente es el algoritmo ϵ-*Approximate* (ϵ-A) Dembczynski et al. [2012] que surge como una alternativa al alto coste computacional de la búsqueda exhaustiva y al pobre rendimiento de la búsqueda voraz. Este método expande solamente aquellos nodos cuya probabilidad condicional conjunta marginal excede un umbral determinado por el parámetro ϵ. Cuando dicho parámetro es igual a 0, el algoritmo realiza una búsqueda de coste uniforme (UC - *Uniform Cost*) y siempre encuentra una solución óptima en términos de *subset 0/1 loss*. Otra alternativa también disponible en la literatura es la búsqueda en haz (BS - *Beam Search*) Kumar et al. [2013] que explora más de

un camino del árbol probabilístico a través de un parámetro que limita el número de nodos que se exploran en cada nivel. La reducción en el tiempo computacional para este método es notable, a pesar de no garantizar predicciones óptimas en términos de *subset 0/1 loss*. También están los métodos basados en *Monte Carlo* Dembczynski et al. [2012]; Read et al. [2014] que generan un cierto número de combinaciones de etiquetas, las cuales deben agregarse para proporcionar la predicción final. En este sentido, se puede tomar la combinación de etiquetas más frecuente (la moda) Dembczynski et al. [2012] o la combinación de etiquetas con la probabilidad condicional conjunta más alta Read et al. [2014]. En cualquier caso, los enfoques basados en *Monte Carlo* convergen para optimizar la *subset 0/1 loss* cuando el número de predicciones extraídas es lo suficientemente grande, aunque no garantizan alcanzar predicciones óptimas.

En los apartados sucesivos se enunciará formalmente el problema de la clasificación multietiqueta y se detallará el método PCC así como los algoritmos de inferencia propuestos en la literatura. Finalmente, se describirán las contribuciones de este trabajo y se detallará la estructura de esta memoria.

1.1. Clasificación multietiqueta

Antes de describir formalmente en qué consiste la clasificación multietiqueta, se va a fijar la nomenclatura que se utilizará en el resto de este documento. Sea $\mathcal{L} = \{\ell_1, \ell_2, \ldots, \ell_m\}$ un conjunto finito y no vacío de m etiquetas y $S = \{(\boldsymbol{x}_1, \boldsymbol{y}_1), \ldots, (\boldsymbol{x}_n, \boldsymbol{y}_n)\}$ un conjunto de ejemplos de entrenamiento independientes y aleatorios obtenidos a partir de una distribución de probabilidad desconocida $\mathbf{P}(\mathbf{X}, \mathbf{Y})$ sobre $\mathcal{X} \times \mathcal{Y}$, donde $\mathcal{X}$ y $\mathcal{Y}$ son el espacio de entrada y salida respectivamente. El primero es el espacio usado para describir las instancias, mientras que el espacio de salida $\mathcal{Y}$ se corresponde con el conjunto de las partes de $\mathcal{L}$, es decir, $\mathcal{P}(\mathcal{L})$. Para facilitar la notación en este trabajo se define $\boldsymbol{y}_i$ como un vector binario $\boldsymbol{y}_i = (y_{i,1}, y_{i,2}, \ldots, y_{i,m})$ en el que $y_{i,j} = 1$ indica la presencia (relevancia) y $y_{i,j} = 0$ la ausencia (irrelevancia) de ℓ_j en el etiquetado de $\boldsymbol{x}_i$. Por lo tanto, $\boldsymbol{y}_i$ es una muestra de la variable aleatoria correspondiente $\mathbf{Y} = (\mathbf{Y_1}, \mathbf{Y_2}, \ldots, \mathbf{Y_m})$. Con esta conversión, el espacio de salida se puede definir como $\mathcal{Y} = \{0, 1\}^m$.

El objetivo en MLC es inducir desde S una hipótesis $\boldsymbol{f} : \mathcal{X} \longrightarrow \mathcal{Y}$ que minimice el riesgo en términos de una cierta función de pérdida $L(\cdot)$ cuando se predice la relevancia de cada etiqueta, $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x}) = (f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_m(\boldsymbol{x}))$, para las instancias no etiquetadas, $\boldsymbol{x}$, que se obtienen a partir de la distribución de probabilidad del problema. Este riesgo puede definirse como la pérdida esperada sobre la distribución conjunta $\mathbf{P}(\mathbf{X}, \mathbf{Y})$, es decir,

$$R_L(\boldsymbol{f}) = \mathbb{E}_{\mathbf{X},\mathbf{Y}} L(\mathbf{Y}, \boldsymbol{f}(\mathbf{X})). \tag{1.1}$$

Así, si se denota por $\mathbf{P}(\boldsymbol{y} \,|\, \boldsymbol{x})$ la distribución condicional $\mathbf{Y} = \boldsymbol{y}$ dada $\mathbf{X} = \boldsymbol{x}$, el

minimizador de riesgo $\boldsymbol{f}^*$ se puede expresar mediante:

$$\boldsymbol{f}^*(\boldsymbol{x}) = \arg\min_{\boldsymbol{f}} \sum_{\boldsymbol{y}\in\mathcal{Y}} \mathbf{P}(\boldsymbol{y} \mid \boldsymbol{x}) L(\boldsymbol{y}, \boldsymbol{f}(\boldsymbol{x})). \tag{1.2}$$

La distribución condicional $\mathbf{P}(\boldsymbol{y} \mid \boldsymbol{x})$ presenta diferentes propiedades que son cruciales para optimizar distintas funciones de pérdida. De esta manera, la estrategia seguida por un algoritmo de MLC para modelar la dependencia de etiquetas determina la función de pérdida optimizada. Desafortunadamente, por lo general es bastante complejo y confuso encontrar la función de pérdida optimizada por algunos algoritmos.

En cuanto a las funciones de pérdida, hay muchas medidas de rendimiento para evaluar la clasificación multietiqueta. Las más específicas son tal vez *subset 0/1 loss* y *Hamming loss*, pero existen otras medidas que se han tomado de diferentes problemas de aprendizaje, como la medida F_1 o el índice de *Jaccard*.

Subset 0/1 loss es una medida orientada a instancias que evalúa para cada instancia si el conjunto de etiquetas predichas coincide exactamente con el conjunto de etiquetas reales o no. Formalmente, se define como [1]

$$L_{S_{0/1}}(\boldsymbol{y}, \boldsymbol{f}(\boldsymbol{x})) = [\![\boldsymbol{y} \neq \boldsymbol{f}(\boldsymbol{x})]\!]. \tag{1.3}$$

Teniendo en cuenta *subset 0/1* como función de pérdida, la expresión del minimizador de riesgo se reduce a optimizar la distribución condicional conjunta, es decir,

$$r_{S_{0/1}}(\boldsymbol{f}(\boldsymbol{x})) = \arg\max_{\boldsymbol{y}\in\mathcal{Y}} \mathbf{P}(\boldsymbol{y} \mid \boldsymbol{x}). \tag{1.4}$$

Esta simplificación del minimizador de riesgo para *subset 0/1 loss* permite proporcionar predicciones óptimas considerando todas las combinaciones posibles para los valores de las etiquetas, aunque esta tarea continúa siendo de orden exponencial. Esta es la razón por la que en los últimos años un foco de atención ha sido diseñar algoritmos de inferencia que permitiesen disminuir este coste computacional.

Hamming loss es otra medida común en MLC para la cual el minimizador de riesgo también se simplifica considerablemente. Esta función de pérdida también es una medida orientada a instancias pero evalúa de forma independiente si el valor predicho para cada etiqueta coincide con el valor real o no. Formalmente, se define como

$$L_H(\boldsymbol{y}, \boldsymbol{f}(\boldsymbol{x})) = \frac{1}{m} \sum_{i=1}^{m} [\![y_i \neq f_i(\boldsymbol{x})]\!]. \tag{1.5}$$

Consecuentemente, el minimizador de riesgo para *Hamming loss* es

$$r_H(\boldsymbol{f}(\boldsymbol{x})) = (r_{H,1}(f_1(\boldsymbol{x})), \ldots, r_{H,m}(f_m(\boldsymbol{x}))), \tag{1.6}$$

[1] $[\![p]\!]$ es igual 1 si p es verdadero y 0 en caso contrario

donde

$$r_{H,i}(f_i(\boldsymbol{x})) = \arg \max_{v \in \{0,1\}} \mathbf{P}(y_i = v \mid \boldsymbol{x}). \tag{1.7}$$

Por lo tanto, proporcionar predicciones óptimas en términos de pérdida de Hamming significa obtener predicciones óptimas para cada etiqueta independientemente del resto de etiquetas Read et al. [2011]. De hecho, aplicando el conocido enfoque *Binary Relevance* Luaces et al. [2012] se puede optimizar esta medida y no se requiere inferencia. Por lo tanto, este trabajo solo se centra en *subset 0/1 loss*, cuya optimización requiere inferencia para obtener predicciones óptimas.

1.2. El método PCC

El método *Probabilistic Classifier Chains* (PCC) está basado en el algoritmo Classifier Chains (CC) Read et al. [2011], de hecho, su fase de entrenamiento es idéntica. Básicamente, consiste en:

(i) establecer un orden para el conjunto de etiquetas,

(ii) el entrenamiento de un clasificador binario probabilístico capaz de estimar $\mathbf{P}(y_j \mid \boldsymbol{x}, y_1, \ldots, y_{j-1})$ por cada etiqueta ℓ_j siguiendo el orden de la cadena.

Por lo tanto, el modelo probabilístico f_j obtenido para predecir la probabilidad de la etiqueta ℓ_j tiene la forma:

$$f_j : \mathcal{X} \times \{0,1\}^{j-1} \longrightarrow [0,1].$$

El conjunto de entrenamiento para cada f_j es $S_j = \{(\overline{\boldsymbol{x}}_1, y_{1,j}), \ldots, (\overline{\boldsymbol{x}}_n, y_{n,j})\}$ donde $\overline{\boldsymbol{x}}_i = (\boldsymbol{x_i}, y_{i,1}, \ldots, y_{i,j-1})$, es decir, $\boldsymbol{x}_i$ suplementado por la relevancia de las etiquetas $\ell_1, \ldots, \ell_{j-1}$ que preceden a ℓ_j en la cadena. La categoría es la relevancia de la etiqueta ℓ_j.

En la etapa de test, los métodos basados en CC realizan inferencia para cada instancia, que consiste en estimar el minimizador de riesgo para una función de pérdida dada sobre la distribución condicional conjunta total estimada. La idea gira en torno a aplicar repetidamente la regla general del producto de probabilidades a la distribución conjunta de las etiquetas $\mathbf{Y} = (\mathbf{Y_1}, \mathbf{Y_2}, \ldots, \mathbf{Y_m})$, es decir,

$$\mathbf{P}(\boldsymbol{y} \mid \boldsymbol{x}) = \prod_{j=1}^{m} \mathbf{P}(y_j \mid \boldsymbol{x}, y_1, \ldots, y_{j-1}). \tag{1.8}$$

Antes de continuar es importante resaltar que desde el punto de vista teórico, esta expresión se aplica a cualquier orden considerado para las etiquetas. Pero en la práctica, estos métodos son dependientes del orden de etiqueta por varias razones. Por un lado, no es posible asegurar que los modelos obtenidos en la etapa de entrenamiento estimen perfectamente la probabilidad condicional conjunta

$P(\boldsymbol{y} \mid \boldsymbol{x})$. Por otro lado, los valores predichos se toman sucesivamente en la etapa de test en lugar de los valores reales. Esta última situación es más grave si los errores más altos ocurren al principio de la cadena, ya que las predicciones erróneas se propagan sucesivamente a lo largo de la cadena Montañés et al. [2014]; Senge et al. [2012, 2013]. En cualquier caso, en este trabajo se usará siempre el orden original de las etiquetas, ya que el objetivo es analizar el rendimiento de los métodos sin tener en cuenta el efecto de los diferentes órdenes.

1.3. Inferencia en el método PCC

En este apartado veremos distintos métodos para llevar a cabo inferencia en PCC. Como se comentó anteriormente, realizar inferencia en PCC consiste en estimar el minimizador de máximo riesgo para una medida de pérdida determinada. Centrándonos en la medida *subset 0/1 loss*, anteriormente llegamos a la conclusión de que se trata de obtener

$$r_{S_{0/1}}(\boldsymbol{f}(\boldsymbol{x})) = \arg\max_{\boldsymbol{y}\in\mathcal{Y}} \prod_{j=1}^{m} \mathbf{P}(y_j \mid \boldsymbol{x}, y_1, \ldots, y_{j-1}). \tag{1.9}$$

Antes de pasar a ver los métodos de inferencia, resulta útil comentar que la inferencia en PCC puede verse como las distintas maneras de explorar un árbol binario de probabilidades. En dicho árbol, el nodo raíz está etiquetado por el conjunto vacío, mientras un nodo genérico k de nivel $j<m$ con $k \leq 2^j$, está etiquetado por $y_j^k = (v_1, v_2, \ldots, v_j)$ con $v_i \in \{0,1\}$ for $i = 1, \ldots, j$. Este nodo tiene dos hijos etiquetados respectivamente como $y_{j+1}^{2k-1} = (v_1, v_2, \ldots, v_j, 0)$ y $y_{j+1}^{2k} = (v_1, v_2, \ldots, v_j, 1)$ y con una probabilidad condicional conjunta marginal $\mathbf{P}(y_1 = v_1, \ldots, y_j = v_j, y_{j+1} = 0 \mid \boldsymbol{x})$ y $\mathbf{P}(y_1 = v_1, \ldots, y_j = v_j, y_{j+1} = 1 \mid \boldsymbol{x})$. Los pesos de los arcos entre el padre y los hijos son, respectivamente $\mathbf{P}(y_{j+1} = 0 \mid \boldsymbol{x}, y_1 = v_1, \ldots, y_j = v_j)$ y $\mathbf{P}(y_{j+1} = 1 \mid \boldsymbol{x}, y_1 = v_1, \ldots, y_j = v_j)$, que se estiman respectivamente utilizando $1 - f_{j+1}(\boldsymbol{x}, v_1, \ldots, v_j)$ y $f_{j+1}(\boldsymbol{x}, v_1, \ldots, v_j)$. La probabilidad condicional conjunta marginal de los hijos se calcula mediante la regla del producto de la probabilidad:

$$\begin{aligned}\mathbf{P}(y_1 = v_1, \ldots, y_j = v_j, y_{j+1} = v_{j+1} \mid \boldsymbol{x}) = \mathbf{P}(y_{j+1} = v_{j+1} \mid \boldsymbol{x}, y_1 = v_1, \ldots, y_j = v_j) \\ \mathbf{P}(y_1 = v_1, \ldots, y_j = v_j \mid \boldsymbol{x}).\end{aligned} \tag{1.10}$$

La Figura 1.1 muestra un nodo genérico y sus hijos en un árbol binario de probabilidad.

Han sido varios los enfoques propuestos en la literatura para hacer inferencia en PCC. El método propuesto en primer lugar fue un método basado en búsqueda voraz (GS - *Greedy Search*), siendo la parte integral del método CC original Read et al. [2009]. Su sucesor fue un método basado en búsqueda exhaustiva (ES - *Exhaustive Search*), denominado método PCC Dembczyński et al. [2010a].

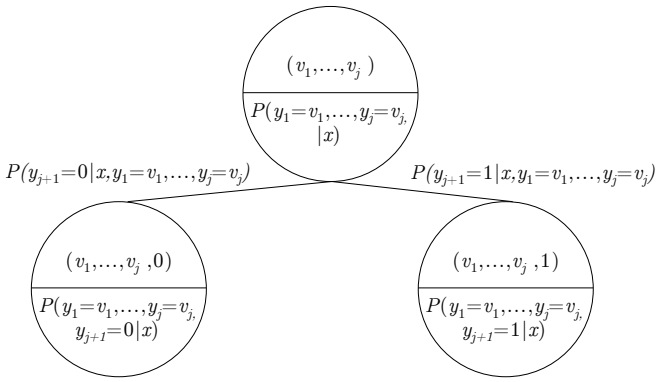


Figura 1.1: Un nodo genérico y sus hijos en un árbol binario de probabilidad. La parte superior de cada nodo contiene la combinación de etiquetas y la parte inferior incluye la probabilidad conjunta de tal combinación. Los arcos se etiquetan con la probabilidad condicional

El algoritmo $\epsilon-$*Approximate* (ϵ-A) surgió como una alternativa al alto coste computacional de la búsqueda exhaustiva y el pobre rendimiento de GS. Este método expande solamente aquellos nodos cuya probabilidad condicional conjunta marginal excede un umbral determinado por el parámetro ϵ. Un enfoque más reciente basado en la búsqueda en haz (BS - *Beam Search*) Kumar et al. [2012, 2013] presenta un buen comportamiento tanto en términos de rendimiento como de coste computacional. Por último, *Monte Carlo sampling* Dembczynski et al. [2012]; Read et al. [2014] es una alternativa atractiva y más simple para superar el alto coste computacional de ES.

Antes de proceder a describir detalladamente las particularidades de estos métodos de inferencia, se debe tener presente que la fase de entrenamiento es común a todos ellos, por lo tanto, los modelos f_j inducidos por los clasificadores binarios serán los mismos independientemente del método de inferencia utilizando en la fase de test. En consecuencia, en la descripción de cada método que se hará a continuación solo se tratará la fase de test para una instancia no etiquetada $\boldsymbol{x}$.

1.3.1. Greedy Search

El método basado en búsqueda voraz (GS - *Greedy Search*), originalmente llamado CC Read et al. [2011, 2009], produce una salida $\hat{\boldsymbol{y}} = (\hat{y}_1, \ldots, \hat{y}_m)$ para una nueva instancia no etiquetada $\boldsymbol{x}$, mediante la evaluación sucesiva de cada clasificador f_j de la cadena que estima la probabilidad condicional $\mathbf{P}(y_j \,|\, \boldsymbol{x}, y_1, \ldots, y_{j-1})$. El método explora un solo camino del árbol binario antes descrito. Cuando se expande un nodo en el nivel $j-1$, se elige el nodo con mayor probabilidad condicional conjunta marginal $\mathbf{P}(y_1, \ldots, y_{j-1}, y_j|\boldsymbol{x})$ entre los

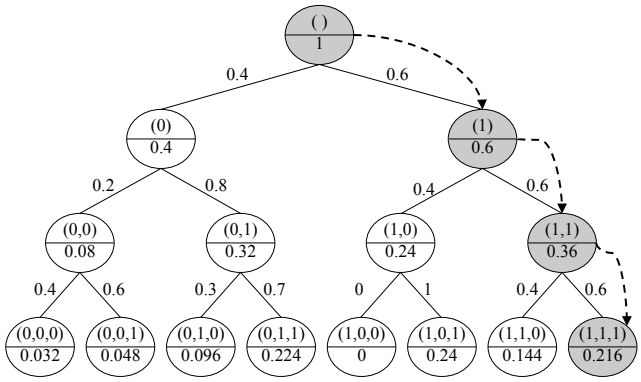


Figura 1.2: Un ejemplo del camino recorrido por *Greedy Search* (GS). Las flechas punteadas muestran el camino que sigue el algoritmo

dos hijos ($y_j = 0$ o $y_j = 1$) del nodo explorado. Aplicando la regla de la cadena $\mathbf{P}(y_1, \ldots, y_{j-1}, y_j|\boldsymbol{x}) = \mathbf{P}(y_1, \ldots, y_{j-1}|\boldsymbol{x}) \cdot \mathbf{P}(y_j|\boldsymbol{x}, y_1, \ldots, y_{j-1})$ y puesto que ambos hijos tienen el mismo padre, el primer término de la expresión anterior es idéntico para ambos y por tanto solo difieren en el segundo término. Consecuentemente, el nodo hijo elegido será aquél que tenga la mayor probabilidad condicionada marginal $\mathbf{P}(y_j|\boldsymbol{x}, y_1, \ldots, y_{j-1})$. De esta manera, para estimar $\hat{y}_j$ se aplica f_j tanto sobre el vector de características $\boldsymbol{x}$ como sobre los valores para las etiquetas desde ℓ_1 a ℓ_{j-1}. Debido a que $y_1, \ldots, y_{j-1}$ no están disponibles en la fase de test se toman en su lugar las respectivas predicciones $\hat{y}_1 = f_1(\boldsymbol{x})$, $\hat{y}_2 = f_2(\boldsymbol{x}, \hat{y}_1), \ldots, \hat{y}_{j-1} = f_{j-1}(\boldsymbol{x}, \hat{y}_1, \hat{y}_2, \ldots, \hat{y}_{j-2})$. De esta manera, la predicción para una instancia $\boldsymbol{x}$ tiene la forma:

$$\hat{y} = (\, f_1(\boldsymbol{x}),\, f_2(\boldsymbol{x}, f_1(\boldsymbol{x})),\, f_3(\boldsymbol{x}, f_1(\boldsymbol{x}), f_2(\boldsymbol{x}, f_1(\boldsymbol{x}))), \ldots). \tag{1.11}$$

La Figura 1.2 muestra el camino recorrido para una instancia utilizando esta estrategia. En este ejemplo, solo se explora el nodo derecho en cada nivel, sin alcanzar la solución óptima, ya que la solución óptima es la que termina en la sexta hoja con la combinación de etiquetas (1,0,1), mientras que el método retorna la última hoja, correspondiente a la combinación etiquetas (1,1,1).

En cuanto a la optimización de *subset 0/1 loss*, un análisis riguroso Dembczynski et al. [2012] establece límites para la convergencia del método GS a una solución óptima. Para este propósito, los autores definen el nivel de fallo r de un clasificador f para una pérdida dada $L(\cdot, \cdot)$ como la diferencia entre el riesgo de ese clasificador y el riesgo del clasificador óptimo de Bayes f^B, es decir

$$r_L(f) = \mathbb{E}_{\mathbf{X},\mathbf{Y}} L(\mathbf{Y}, \boldsymbol{f}(\mathbf{X})) - \mathbb{E}_{\mathbf{X},\mathbf{Y}} L(\mathbf{Y}, \boldsymbol{f^B}(\mathbf{X})). \tag{1.12}$$

En el caso de *subset 0/1 loss*, se puede analizar la expectativa sobre $\mathbf{Y}$ dado $\boldsymbol{x}$, ya que esta pérdida se puede descomponer para las instancias individuales. Por lo tanto, el nivel de fallo se convierte en

$$r_L(f) = \mathbb{E}_{\mathbf{Y}} L(\mathbf{Y}, \boldsymbol{f}(\boldsymbol{x})) - \mathbb{E}_{\mathbf{Y}} L(\mathbf{Y}, \boldsymbol{f}^{\boldsymbol{B}}(\boldsymbol{x})). \tag{1.13}$$

Suponiendo que se obtiene una estimación perfecta de la probabilidad condicional conjunta $\mathbf{P}(\boldsymbol{y} \mid \boldsymbol{x})$, es posible establecer una cota superior igual a $2^{-1} - 2^{-m}$ para el nivel de fallo del clasificador h inducido por el método GS para *subset 0/1 loss*. Esta cota superior muestra que este método puede ofrecer un rendimiento bastante bajo para esta función de pérdida, es decir, el método no logra proporcionar una buena estimación del minimizador de riesgo para *subset 0/1 loss*. Sin embargo, el estudio Dembczynski et al. [2012] sobre el rendimiento de GS concluye que GS trata de optimizar el *subset 0/1 loss* en lugar de otras medidas, como por ejemplo *Hamming loss*, ya que tiende a estimar la distribución condicional conjunta en lugar de la distribución condicional marginal.

1.3.2. Exhaustive Search

La búsqueda exhaustiva (ES - *Exhaustive Search*) explora todos los caminos posibles del árbol a diferencia de GS que solo explora una combinación de etiquetas. Esto significa que ES estima toda la distribución condicional conjunta $\mathbf{P}(y \mid \boldsymbol{x})$ para una nueva instancia $\boldsymbol{x}$. En consecuencia, proporciona inferencia óptima de Bayes. Esto es, utilizando $f(\boldsymbol{x})$ repetidamente se calcula $\mathbf{P}(\boldsymbol{y} \mid \boldsymbol{x})$ para todas las combinaciones de etiquetas $\boldsymbol{y} = (y_1, y_2, \ldots, y_m)$ y, teniendo en cuenta $L(\boldsymbol{y}, \boldsymbol{f}(\boldsymbol{x}))$, se selecciona la combinación $\hat{\boldsymbol{y}} = (\hat{y}_1, \ldots, \hat{y}_m) = \boldsymbol{f}^*(\boldsymbol{x})$ con el menor riesgo para la función de pérdida dada $L(\cdot, \cdot)$. Al hacer esto generalmente se mejora en términos de rendimiento, ya que se estima perfectamente el minimizador de riesgo, aunque a costa de un coste computacional excesivamente alto, ya que supone la evaluación de un número exponencial (2^m) de combinaciones de etiquetas para cada $f(\boldsymbol{x})$.

La Figura 1.3 ilustra este enfoque en el que se exploran todos los caminos y se alcanza siempre la solución óptima.

1.3.3. Algoritmo ϵ-Approximate

El algoritmo ϵ-*Approximate* (ϵ-A) Dembczynski et al. [2012] surge como una alternativa al alto coste computacional de ES y al pobre rendimiento de CC. En términos de un árbol de probabilidad, expande solo los nodos cuya probabilidad condicional conjunta marginal excede el umbral $\epsilon = 2^{-k}$ con $1 \leq k \leq m$. Esta probabilidad condicional conjunta marginal para un nodo en el nivel j para una instancia no etiquetada $\boldsymbol{x}$ es

$$\mathbf{P}(y_1, \ldots, y_j \mid \boldsymbol{x}) = \prod_{i=1}^{j} \mathbf{P}(y_i \mid \boldsymbol{x}, y_1, \ldots, y_{i-1}), \tag{1.14}$$

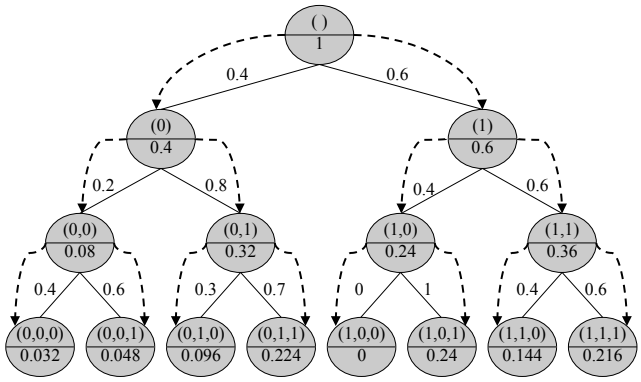


Figura 1.3: Un ejemplo del camino recorrido por *Exhaustive Search* (ES). Las flechas punteadas muestran el camino que sigue el algoritmo

donde $\mathbf{P}(y_i, \mid \boldsymbol{x}, y_1, \ldots, y_{i-1})$ se estima mediante $f_i(\boldsymbol{x}, y_1, \ldots, y_{i-1})$.

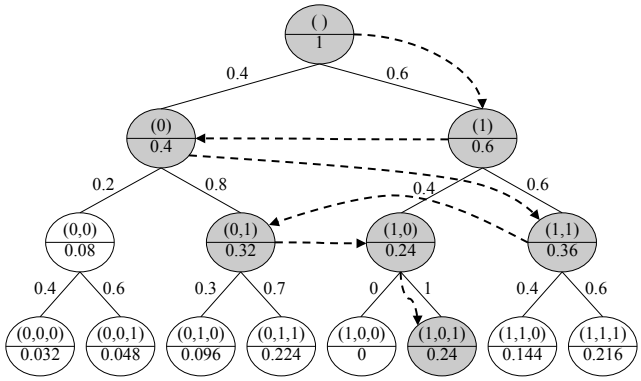
Los nodos se expanden en el orden establecido por esta probabilidad (estrategia de búsqueda primero el mejor), calculando la probabilidad condicional conjunta marginal para sus hijos. Por lo tanto, el algoritmo no sigue un camino específico, sino que cambia de un camino a otro en función de las probabilidades condicionales conjuntas marginales. Al final, se pueden encontrar dos situaciones:

(i) el nodo expandido es una hoja,

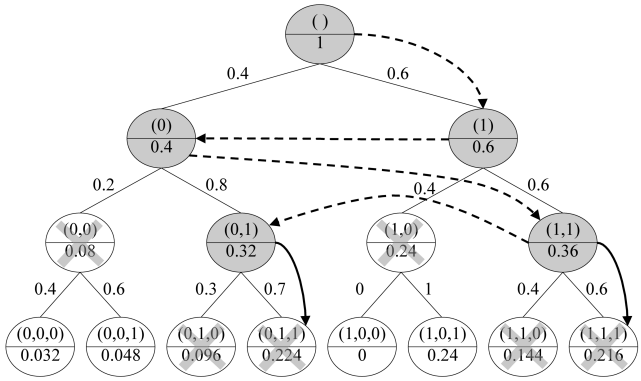
(ii) no hay más nodos que excedan el umbral.

Si se presenta la primera situación, la predicción para la instancia no etiquetada $\boldsymbol{x}$ será $\hat{\boldsymbol{y}} = (\hat{y}_1, \ldots, \hat{y}_m)$ correspondiente a la combinación de la hoja alcanzada (ver Figura 1.4(a)). Por el contrario, si ocurre la segunda situación, se aplica GS a los nodos cuyos hijos no exceden el umbral y la predicción $\hat{\boldsymbol{y}} = (\hat{y}_1, \ldots, \hat{y}_m)$ para la instancia no etiquetada $\boldsymbol{x}$ en este caso será la que tiene probabilidad condicional conjunta más alta $\mathbf{P}(y_1, \ldots, y_m \mid \boldsymbol{x})$.

El parámetro ϵ juega un papel importante en el algoritmo. El caso particular de $\epsilon = 0$ (o cualquier valor en el intervalo $[0, 2^{-m}]$, es decir, $k = m$) tiene un interés especial, ya que el algoritmo realiza una búsqueda de coste uniforme (UC) que siempre encuentra la solución óptima. Esto ocurre porque está garantizando que la probabilidad condicional conjunta para al menos una hoja será superior a ϵ, ya que todas las probabilidades de las hojas son positivas (o nulas en el peor de los casos) y suman 1. Al menos una de ellas debe ser mayor o igual que 2^{-m}. Incluso, esta hoja es una solución óptima, ya que las probabilidades condicionales conjuntas marginales disminuyen a medida que se baja en el árbol. La Figura 1.4(a) ilustra



(a) Algoritmo ϵ-A con $\epsilon = .0 (k = m)$



(b) Algoritmo ϵ-A con $\epsilon = .25 (k = 2)$

Figura 1.4: Ejemplos del camino recorrido por el algoritmo ϵ-A para $\epsilon = .0$ y $\epsilon = .25$. Los nodos marcados con una cruz son aquellos que tienen una probabilidad condicional conjunta marginal inferior a ϵ y no son explorados. Las flechas punteadas muestran el camino seguido por el algoritmo. Las flechas sólidas indican el camino seguido por el algoritmo cuando la probabilidad condicional conjunta marginal no excede el valor de ϵ y se un aplica GS hasta las hojas

esta situación. En este caso, todos los nodos son candidatos a ser explorados, ya que todos ellos tendrán una probabilidad condicional conjunta marginal mayor que ϵ. En primer lugar, se explora el nodo derecho del primer nivel. Sus hijos tienen una probabilidad condicional conjunta marginal (0.24 y 0.36 respectivamente) menor que su tío (0.4), por lo que su tío es explorado antes que ellos. Después de eso, se explora el hijo más a la derecha del primer nodo porque tiene la probabilidad condicional marginal más alta (0.36) entre su hermano y sus primos. El siguiente nodo a explorar es su primo de la parte derecha con una probabilidad de 0.32, que es más alta que sus hijos (0.144 y 0.216). Luego, se explora su hermano (0.24) y finalmente su sobrino derecho 0.24, que ya es una hoja y como se esperaba es la hoja de la solución óptima.

Nótese que este método tiende a comportarse como GS conforme ϵ crece, siendo GS en el caso de $\epsilon = .5$ (o equivalente a $\epsilon = 2^{-1}$, es decir, $k = 1$). Esto ocurre por las siguientes situaciones:

(i) solo un nodo tiene una probabilidad condicional conjunta marginal mayor que ϵ, en cuyo caso el algoritmo sigue un único camino, o

(ii) ningún nodo tiene una probabilidad condicional conjunta marginal mayor que ϵ, en cuyo caso se aplica un GS desde aquí hasta alcanzar una hoja.

La Figura 1.4(b) muestra el caso particular de $\epsilon = .25$. En la figura, algunos nodos no cumplen la restricción de tener una probabilidad condicional conjunta marginal mayor que ϵ. Incluso, ninguna hoja se alcanza sin aplicar la estrategia GS y por tanto el algoritmo no puede garantizar devolver una solución óptima.

Por tanto, este algoritmo estima el minimizador de riesgo para *subset 0/1 loss* en mayor o menor medida dependiendo del valor de ϵ. De hecho, un análisis teórico de esta estimación Dembczynski et al. [2012] establece que este algoritmo con $\epsilon = 2^{-k}$ necesita menos de $\mathbf{O}(m2^k)$ iteraciones para encontrar una predicción que acota el peor caso del nivel de fallo $r_L(f)$ del clasificador f por $2^{-k} - 2^{-m}$ para *subset 0/1 loss* y $k \leq m$, nuevamente bajo la suposición de que se obtiene una estimación perfecta de la probabilidad condicional conjunta $P(\boldsymbol{y} \mid \boldsymbol{x})$. En el caso particular de $\epsilon = 0$ (o $k = m$) el límite se convierte en 0, lo que demuestra que para este valor de ϵ, el algoritmo alcanza la solución óptima.

1.3.4. Beam Search

La búsqueda en haz (BS - *Beam Search*) Kumar et al. [2012, 2013] también explora más de un camino en el árbol de probabilidades. Este método incluye un parámetro b llamado *beam width* que limita el número de combinaciones de etiquetas que se exploran en cada nivel. La idea es explorar b posibles conjuntos de etiquetas en cada nivel del árbol, aquellos con mayor probabilidad condicional conjunta marginal. Esto significa que para los $k^* - 1$ primeros niveles, se exploran todas las combinaciones de etiquetas posibles, mientras que solo se exploran b combinaciones en los niveles restantes, siendo k^* el entero más bajo tal que $b < 2^{k^*}$

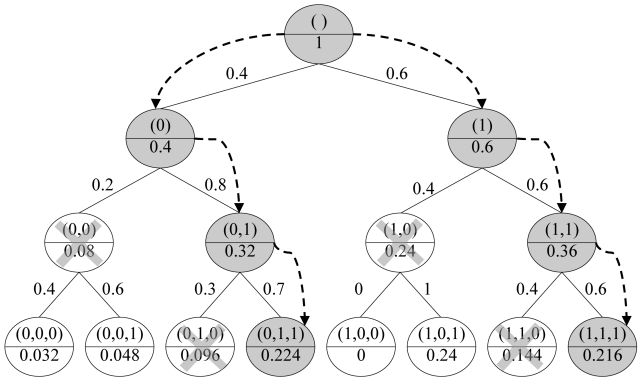


Figura 1.5: Un ejemplo del camino seguido por una instancia utilizando BS con $b=2$. Los nodos marcados con una cruz no tienen ninguna de las probabilidades condicionales marginales más altas de su nivel y, por lo tanto, no se exploran más. Las flechas punteadas muestran el camino que sigue el algoritmo

Las b combinaciones exploradas desde el nivel k^* hasta las hojas son aquellas con la probabilidad condicional conjunta marginal más alta vista en cada nivel. Esta probabilidad condicional conjunta marginal para un nodo de nivel j dada una instancia no etiquetada $\boldsymbol{x}$ es la misma que utiliza el algoritmo ϵ-A (ver Ecuación (1.14)). En el último nivel, el algoritmo produce una salida $\hat{\boldsymbol{y}} = (\hat{y}_1, \ldots, \hat{y}_m)$ con la probabilidad condicional conjunta más alta $\mathbf{P}(y_1, \ldots, y_m \,|\, \boldsymbol{x})$.

BS difiere de GS, en primer lugar en que BS explora más de una combinación, y en segundo lugar, en la probabilidad tomada para decidir un camino a seguir en el árbol. En cuanto a la segunda diferencia, ambos toman la probabilidad condicional conjunta marginal $\mathbf{P}(y_1, \ldots, y_j|\boldsymbol{x})$, pero en el caso de GS, es equivalente a tomar la probabilidad condicional marginal $\mathbf{P}(y_j, |\, \boldsymbol{x}, y_1, \ldots, y_{j-1})$ ya que, como se explicó anteriormente, los dos nodos explorados en cada nivel del árbol tienen el mismo padre. Sin embargo, en el caso de BS, los b nodos explorados no tienen que tener el mismo padre (excepto el nodo raíz) incluso en el caso en el que se expanden dos nodos ($b=2$) por nivel. Analizando valores extremos para b, concluimos que por un lado, para $b=1$, BS expande solo un nodo en cada nivel, aquel con la probabilidad condicional marginal más alta, por tanto, recorre un solo camino que coincide con el que sigue GS, y por otro lado para $b \geq 2^m$, BS realiza una ES. Por lo tanto, BS encapsula GS y ES, respectivamente, considerando $b=1$ y $b=2^m$. Esto hace que sea posible controlar el equilibrio entre el coste computacional y el rendimiento del método ajustando b entre 1 y 2^m. El número de nodos expandidos por BS está limitado por $\mathbf{O}(bm)$.

La Figura 1.5 muestra un ejemplo de los caminos explorados por BS cuando

$b=2$. Por lo tanto, todos los nodos del primer nivel son explorados. A partir de ahí, se exploran solo dos nodos en cada nivel, aquellos con probabilidades condicionales marginales más altas entre los nodos generados en ese nivel al expandirse sus respectivos padres. De esta manera se puede observar que el nodo con la solución óptima no se explora ya que su padre no ha sido previamente explorado. Este ejemplo confirma que BS no puede garantizar alcanzar soluciones óptimas a menos que lleve a cabo una ES, es decir, $b=2^m$.

Finalmente, el hecho de que BS considere probabilidades condicionales marginales conjuntas hace que el método tienda a estimar el minimizador de riesgo para *subset 0/1 loss*. Aunque es posible incluir otras funciones de pérdida en el algoritmo de búsqueda. En este sentido, los autores que propusieron BS para la inferencia en PCC Kumar et al. [2012, 2013] no incluyeron ningún análisis teórico sobre el nivel de fallo en la estimación del minimizador de riesgo. Sin embargo, muestran empíricamente que al tomar ciertos valores de b ($b \approx 15$), el minimizador de riesgo proporcionado por el método converge hacia soluciones óptimas.

1.3.5. Métodos basados en Monte Carlo

Monte Carlo sampling es una técnica basada en la repetición del muestreo aleatorio para obtener una aproximación de una distribución probabilística desconocida. Existen varias maneras de implementar un método Monte Carlo, pero tienden a seguir un patrón común:

(i) Definen un dominio de valores posibles.

(ii) Generan valores aleatoriamente a partir de una distribución de probabilidad sobre el dominio definido.

(iii) Realizan un cálculo determinista sobre los valores.

(iv) Finalmente, agregan los resultados.

Con respecto a utilizar esta estrategia para realizar inferencia en PCC, son dos los algoritmos que se han propuesto recientemente Dembczynski et al. [2012]; Read et al. [2014]. Ambos generan un conjunto de observaciones (combinaciones de etiquetas en nuestro caso), cada una de ellas formada por m valores en el dominio $\{0, 1\}$. En ambos enfoques, los valores aleatorios de las combinaciones se construyen utilizando cada clasificador f_j, que estima la probabilidad $\mathbf{P}(y_j \mid \boldsymbol{x}, y_1, \ldots, y_{j-1})$ de que la etiqueta ℓ_j sea relevante para una nueva instancia no etiquetada $\boldsymbol{x}$. Si el valor aleatorio es menor o igual que 1 menos esa probabilidad se asigna un 0 para esa etiqueta y si no un 1 (ver ejemplo en Figura 1.6). En la iteración para generar la combinación i-ésima y en ella determinar el valor para la etiqueta ℓ_j, ambos algoritmos toman $\hat{y}_1^{(i)}, \ldots, \hat{y}_{j-1}^{(i)}$ previamente obtenidos en la cadena para determinar el valor de $\hat{y}_j^{(i)}$. Por lo tanto, la probabilidad condicional $\mathbf{P}(y_j \mid \boldsymbol{x}, \hat{y}_1^{(i)}, \ldots, \hat{y}_{j-1}^{(i)})$ estimada por $f_j(\boldsymbol{x}, \hat{y}_1^{(i)}, \ldots, \hat{y}_{j-1}^{(i)}))$ se calcula cuando el

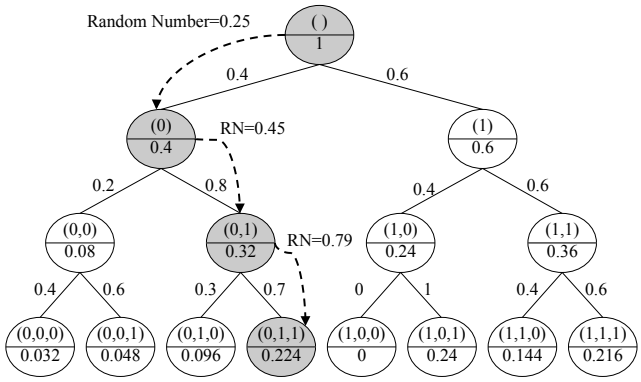


Figura 1.6: Un ejemplo de una observación construida utilizando el método *Monte Carlo sampling*. En cada nodo v, se lanza una moneda sesgada para decidir si la etiqueta es relevante o no. Se da la probabilidad de cara y cruz respectivamente, por los pesos de $\pi(lc(v))$ y $\pi(rc(v))$ de los hijos izquierdo y derecho de un nodo v. Si el número aleatorio es menor o igual que $\pi(lc(v))$ se selecciona el hijo izquierdo y la etiqueta es irrelevante, de lo contrario, se selecciona el hijo derecho. Una observación de la distribución condicional se obtiene cuando se alcanza una hoja

proceso aleatorio tiene lugar para la etiqueta ℓ_j. La diferencia entre los métodos basados en *Monte Carlo* y GS es que, en este último la predicción $\hat{y}_j$ para ℓ_j se obtiene directamente de la evaluación de f_j, mientras que en los métodos basados en Monte Carlo tal valor se obtiene después de un proceso aleatorio que utiliza la distribución inducida por f_j. Esta diferencia hace posible repetir el proceso un cierto número de iteraciones, produciendo predicciones diferentes cada vez, aunque algunas de ellas pueden repetirse en varias iteraciones.

La Figura 1.6 muestra un ejemplo de una observación. En cada nodo v, se genera un número aleatorio para decidir si la siguiente etiqueta es relevante o no. Si el número aleatorio es menor o igual que $\Pi(lc(v))$ entonces el hijo izquierdo se selecciona y la siguiente etiqueta es irrelevante, de lo contrario, la etiqueta es relevante. El proceso se mueve hacia abajo en el árbol de acuerdo con esta regla hasta que se alcanza una hoja, obteniendo una nueva observación. Esto implica que el número de nodos expandidos por los métodos de *Monte Carlo* sea siempre $m \cdot q$, siendo q el número de observaciones de la muestra.

Finalmente, se pueden realizar diferentes métodos de agregación de las predicciones obtenidas dependiendo de la función de pérdida que se va a optimizar. El minimizador de riesgo en este caso se estima sobre el subconjunto de $\mathcal{Y}$ formado por las predicciones obtenidas en las iteraciones en lugar de sobre todo el conjunto $\mathcal{Y}$. Por lo tanto, la estimación puede ser más pobre que en el caso de ES, y en

consecuencia, no se garantiza que se alcance una solución óptima. Sin embargo, el hecho de utilizar la distribución inducida por f_j en el proceso aleatorio garantiza que la predicción agregada final $\hat{\boldsymbol{y}} = (\hat{y}_1, \ldots, \hat{y}_m)$ de estos métodos converja al minimizador de riesgo cuando la muestra es suficientemente grande y se toma un procedimiento de agregación adecuado para la función de pérdida a minimizar.

Los dos métodos de *Monte Carlo* propuestos hasta el momento difieren en que el primero (llamado aquí MC) Dembczynski et al. [2012] toma la combinación más frecuente de $\hat{\boldsymbol{y}} = (\hat{y}_1, \ldots, \hat{y}_m)$, es decir, la moda, mientras que el propuesto más recientemente (llamado aquí EMC, MC Eficiente) Read et al. [2014] toma la combinación que alcanza la probabilidad condicional conjunta más alta. Los autores de este último, etiquetan este método como eficiente porque su procedimiento de agregación permite que el método converja más rápido a una solución óptima, y además, no es necesario almacenar todas las combinaciones, consumiendo, por tanto, menos memoria. En cualquier caso, ambos tienden a optimizar *subset 0/1 loss*, y lo hace en mayor medida cuando mayor sea el número de observaciones generadas, a pesar de que en cualquier caso no es posible garantizar que alcancen soluciones óptimas.

1.4. Contribuciones

Las contribuciones de este trabajo de investigación se centran en explorar el uso de búsqueda heurística para realizar inferencia en PCC como estrategia alternativa a las existentes en la literatura. Para ello se ha realizado:

(i) Un estudio exhaustivo de los principales algoritmos basados en PCC para clasificación multietiqueta existentes hasta el momento en la literatura, analizando y comparando para cada uno de ellos su comportamiento, propiedades, estrategia de búsqueda utilizada y el efecto causado por la variación de sus parámetros. Los detalles de este estudio se publicaron en Mena et al. [2016].

(ii) Como contribución y novedad en esta línea de investigación, un estudio del uso de métodos de búsqueda heurística para realizar inferencia en PCC. Particularmente, se propone utilizar el algoritmo A* que permite encontrar siempre una solución óptima si se utiliza un heurístico admisible. Esta contribución se subdivide en dos partes que se detallan a continuación:

   (a) En primera instancia, con el fin de obtener un heurístico admisible para garantizar soluciones óptimas en términos de *subset 0/1 loss* y lo más dominante posible para que el número de nodos explorados se minimice, surge un heurístico admisible que solo es aplicable a modelos base lineales en PCC. Debido a que su cálculo incrementa en exceso el coste computacional del algoritmo, se introduce un parámetro que da lugar a una familia de heurísticos que permite controlar el equilibrio

entre el número de nodos explorados (dominancia del heurístico) y el coste computacional del heurístico. El resultado es una familia de heurísticos que garantiza predicciones óptimas y explora menor número de nodos que otros métodos existentes en la literatura que también obtienen predicciones óptimas. Si bien, en tiempo computacional no es el mejor método cuando la búsqueda es más dirigida, sí que muestran su potencial en situaciones complejas donde existe mayor incertidumbre en la búsqueda de soluciones óptimas. Los resultados de esta contribución se publicaron en Mena et al. [2015a, 2017b].

(b) En segunda instancia, debido a que la familia de heurísticos admisibles mencionados en el punto anterior solo es aplicable a modelos base lineales en PCC, se diseñó un heurístico admisible que elimina esta restricción y por tanto también es válido para modelos base no lineales. Surge así una familia de heurísticos que: 1) también son admisibles, por lo tanto, garantizan predicciones óptimas en términos de *subset 0/1 loss*, 2) incluye un parámetro de profundidad para equilibrar el número de nodos explorados y el tiempo computacional de cálculo del heurístico, pero 3) supera la limitación de ser aplicable solo cuando se usan clasificadores lineales como métodos base en PCC, 4) es más dominante para la misma profundidad, por lo tanto, teóricamente explora menos nodos que el heurístico del punto anterior y 5) su tiempo de cálculo es menor también para la misma profundidad, ya que se reduce el número de modelos evaluados. Los experimentos llevados a cabo también confirman que esta familia de heurísticos propuesta es competitiva con respecto a otros enfoques existentes en la literatura, especialmente para problemas más complejos y con mayor incertidumbre en la búsqueda. Los resultados de esta contribución se publicaron en Mena et al. [2015b] y Mena et al. [2017a] (aceptado, pendiente de publicación).

Los resultados de esta investigación se publicaron en tres artículos de revista y dos contribuciones congreso, que se listan a continuación:

(i) Artículos de revista:

[1] D. Mena, E. Montañés, J. R. Quevedo, and J. J. Coz, "An overview of inference methods in probabilistic classifier chains for multilabel classification," Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, vol. 6, no. 6, pp. 215–230, 2016. ISSN: 1942-4787 DOI: 10.1002/widm.1185

[2] D. Mena, E. Montañés, J. R. Quevedo, and J. J. del Coz, "A family of admissible heuristics for A* to perform inference in probabilistic classifier chains," Machine Learning, vol. 106, no. 1, pp. 143–169, 2017. ISSN: 0885-6125 DOI:10.1007/s10994-016-5593-5

[3] D. Mena, E. Montañés, J. R. Quevedo, and J. J. Del Coz, "A heuristic in A* for inference in nonlinear Probabilistic Classifier Chains," Knowledge-Based Systems, 2017. ISSN: 0950-7051 DOI: 10.1016/j.knosys.2017.03.015

(ii) Contribuciones a congresos:

[1] D. Mena, E. Montañés, J. R. Quevedo, and J. J. Del Coz, "Using A* for inference in probabilistic classifier chains," in Proceedings of the 24th International Conference on Artificial Intelligence, 2015, pp. 3707–3713.

[2] D. Mena, E. Montañés, J. R. Quevedo, and J. J. Del Coz, "Heurístico Exhaustivo de Profundidad Limitada para CC Probabilístico," in Actas de la XVI Conferencia de la Asociación Española para la Inteligencia Artificial, 2015, pp. 801–810.

1.5. Estructura de la tesis

Esta investigación se presenta bajo la modalidad de compendio de publicaciones. En el Capítulo 1 se introduce el problema objeto de estudio e investigación, se describen las soluciones existentes hasta el momento en la literatura y se enumeran las contribuciones de este trabajo. En el Capítulo 2, se presentan los objetivos de la presente investigación. Los Capítulos 3 y 4 describen respectivamente los resultados y conclusiones que se pueden extraer del trabajo realizado. El capítulo 5 agrupa los artículos publicados en revistas derivados de esta investigación. Finalmente, en el Capítulo 6 se analiza la calidad de las publicaciones que se aportan como compendio.

Capítulo 2

Objetivos

Uno de los problemas inherentes a la clasificación multietiqueta es que el número de etiquetas suele ser elevado y éstas a su vez suelen tener dependencias estadísticas entre ellas. Debido a ello, una de las principales líneas de investigación en MLC se ha centrado en el diseño de nuevos métodos capaces de explotar estas interdependencias entre las etiquetas. El método *Probabilistic Classifier Chains* (PCC) es el que más interés ha causado entre la comunidad multietiqueta, debido a que a través de la regla del producto de probabilidades es capaz de estimar la probabilidad condicional conjunta de todas las combinaciones de etiquetas para quedarse con una que alcance un mayor valor como predicción óptima, maximizando así una de las principales medidas de rendimiento utilizadas en la clasificación multietiqueta, *subset 0/1 loss*. Sin embargo, su versión original sufre de un alto coste computacional por utilizar como estrategia de inferencia una búsqueda exhaustiva. Con el fin de ofrecer alternativas que reduzcan dicho coste computacional surgen en la literatura varios métodos, entre los que se encuentran el algoritmo ϵ-A, búsqueda en haz (BS) y los métodos basados en *Monte Carlo sampling*, además del ya existente cuya estrategia es búsqueda voraz (GS) denominado CC.

El principal problema de todos ellos es que únicamente ϵ-A con $\epsilon = 0$ es capaz de garantizar predicciones óptimas. Por ello, el objetivo principal de esta investigación es estudiar y desarrollar nuevos métodos de inferencia basados en búsqueda heurística, que también garanticen predicciones óptimas en términos de *subset 0/1 loss* como ϵ-A con $\epsilon = 0$ pero más eficientes, en términos de número de nodos explorados y/o en tiempo de ejecución. Para esto se definieron los siguientes objetivos específicos.

- Estudiar las estrategias de búsqueda, propiedades y comportamiento de los principales algoritmos de inferencia para PCC. Para ello se hará un estudio exhaustivo de su rendimiento en función de sus parámetros.
- Desarrollar nuevos métodos alternativos de inferencia en PCC basados en

búsqueda heurística, capaces de obtener predicciones óptimas sin que el coste computacional sea elevado.

- Comparar los métodos de inferencia en PCC propuestos con los métodos existentes en la literatura para el mismo fin.
- Analizar los efectos de la adición de incertidumbre en los métodos de inferencia para PCC tanto existentes previamente como propuestos en este trabajo, con el fin de conocer el potencial de los nuevos métodos alternativos.

Capítulo 3

Resultados

En este capítulo se exponen y discuten los resultados obtenidos mediante el desarrollo del presente trabajo de investigación, dando cumplimiento a los objetivos enumerados en el capítulo anterior y exponiendo las principales contribuciones de la investigación. De esta manera, en la Sección 3.1 se presenta un estudio comparativo de los principales algoritmos basados en cadenas de clasificadores probabilísticos para clasificación multietiqueta, donde se analizan sus propiedades y parámetros y el efecto de la variación de éstos en cuanto a coste computacional y rendimiento en términos de *subset 0/1 loss*. En la Sección 3.2 se explica el algoritmo A* y los aspectos necesarios para diseñar heurísticos admisibles que puedan emplearse con dicho método de búsqueda. Las Secciones 3.3 y 3.4 presentan respectivamente las dos familias de heurísticos propuestas: la primera que puede emplearse solamente con modelos lineales y la segunda que admite el uso de modelos no lineales. En cada una de esas secciones, se incluyen apartados para explicar las ideas en las que se basa el diseño de los heurísticos, sus propiedades, la complejidad computacional que tienen y los detalles necesarios para poder implementarlos de forma eficiente. La Sección 3.5 amplía el estudio experimental presentando en la Sección 3.1 incluyendo las familias de heurísticos propuestas y añadiendo tests estadísticos para comparar los diferentes métodos estudiados. Por último, la Sección 3.6 analiza los efectos de la adición de datos ruidosos o incertidumbre en los métodos que garantizan alcanzar soluciones óptimas.

3.1. Estudio experimental de los métodos de inferencia existentes para PCC

En esta sección se presenta un análisis experimental exhaustivo de los principales métodos basados en cadenas de clasificadores probabilísticos para clasificación multietiqueta. El estudio se centra en examinar para cada método su rendimiento en términos de *subset 0/1*, el coste computacional basado en el

Tabla 3.1: Propiedades de los conjuntos de datos

Conjunto de dato	Instancias	Atributos	Etiquetas	Cardinalidad
bibtex	7395	1836	159	2.40
corel5k	5000	499	374	3.52
emotions	593	72	6	1.87
enron	1702	1001	53	3.38
flags	194	19	7	3.39
image	2000	135	5	1.24
mediamill*[1]	5000	120	101	4.27
medical	978	1449	45	1.25
reuters	7119	243	7	1.24
scene	2407	294	6	1.07
slashdot	3782	1079	22	1.18
yeast	2417	103	14	4.24

número de nodos explorados y el tiempo de predicción y el efecto de variación del orden de las etiquetas. Adicionalmente esto permite determinar qué enfoques efectivamente garantizan alcanzar una solución óptima o en qué grado se acercan a ella según la variación de su configuración.

3.1.1. Configuración de los experimentos

Los métodos analizados corresponden a los presentados en el Capítulo 1, excepto ES debido a su alto coste computacional. En concreto, se comparan el método GS, el algoritmo $\epsilon-$A para diferentes valores de ϵ $(.0, .25, .5)$, BS para diferentes valores de b $(1, 2, 3, 10)$, y los métodos MC y EMC con diferentes tamaños de muestras $(10, 50, 100)$ tal como se sugiere en Read et al. [2014]. Recordemos que, tal como se mencionó anteriormente, GS es equivalente a ϵ-A con $\epsilon = .5$ y a BS con $b = 1$ y que ϵ-A con $\epsilon = .0$ es equivalente a UC.

Para llevar a cabo el estudio, se realizaron experimentos sobre varios conjuntos de datos de referencia cuyas principales propiedades se muestran en la Tabla 3.1. Tal como se puede ver, existen diferencias significativas en el número de atributos, instancias y etiquetas. La cardinalidad —número de etiquetas por instancia— varía entre 1.07 y 4.27. En cuanto al número de etiquetas, hay algunos conjuntos de datos con solo 5, 6 o 7 etiquetas, mientras que otros tienen más de un centenar, incluso uno de ellos tiene 374 etiquetas. El algoritmo base empleado para obtener los clasificadores binarios f_i fue regresión logística Lin et al. [2008] con salida probabilística. El parámetro de regularización C se estableció para cada clasificador binario realizando un *grid search* sobre los valores $C \in \{10^{-3}, 10^{-2}, \ldots, 10^{2}, 10^{3}\}$ optimizando la medida *Brier score* (Brier [1950])

$$\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m}\left(y_j - P(y_j|\boldsymbol{x}_i)\right)^2,$$

[1] mediamill* corresponde a la abreviación de mediamillSS5M

Tabla 3.2: *Subset 0/1 loss*. Aquellos resultados que son iguales o mejores que las predicciones óptimas se muestran en negrita

Conjuntos	ES/UC ϵ-A(.0)	ϵ-A (.25)	GS BS(1) ϵ-A(.5)	BS (2)	BS (3)	BS (10)	MC (10)	MC (50)	MC (100)	EMC (10)	EMC (50)	EMC (100)
bibtex	**81.92**	81.95	82.19	**81.88**	**81.92**	**81.92**	84.02	82.85	82.46	82.73	82.22	82.11
corel5k	**97.48**	98.62	98.90	98.30	98.04	**97.48**	99.64	98.98	98.26	98.72	97.70	**97.42**
emotions	**71.16**	71.82	72.83	72.16	71.32	**71.16**	80.77	73.68	73.84	72.83	72.33	72.50
enron	**83.14**	84.26	85.43	83.43	83.37	**83.14**	92.95	85.90	84.61	87.43	83.61	83.26
flags	**87.13**	87.16	**86.13**	88.21	**87.13**	**87.13**	96.39	91.21	89.24	91.29	90.71	90.18
image	**68.35**	**68.35**	69.75	**68.35**	**68.35**	**68.35**	69.20	**65.25**	**62.65**	**64.95**	**65.70**	**62.95**
mediamill*	**83.86**	84.58	85.80	84.10	**83.86**	**83.86**	90.90	85.88	84.88	85.34	**83.70**	**83.40**
medical	**30.37**	**30.37**	30.67	**30.37**	**30.37**	**30.37**	31.19	**30.16**	30.67	**30.16**	**30.16**	**30.16**
reuters	**22.73**	**22.70**	23.60	**22.69**	**22.73**	**22.73**	25.37	23.18	22.83	22.97	22.83	22.83
scene	**31.86**	**31.86**	33.28	31.90	**31.86**	**31.86**	33.53	**30.28**	**29.70**	**29.24**	**29.24**	**29.29**
slashdot	**51.80**	52.22	54.49	**51.77**	**51.80**	**51.80**	56.45	52.96	52.70	52.22	52.35	52.35
yeast	**76.95**	77.62	79.77	**76.83**	77.08	**76.95**	85.52	79.93	79.35	79.06	77.53	77.62

estimada mediante una validación cruzada de 2 particiones con 5 repeticiones.

3.1.2. Resultados para *subset 0/1 loss*

La Tabla 3.2 muestra los resultados de *subset 0/1 loss* utilizando validación cruzada de 10 particiones con una repetición. Antes de discutir a fondo los resultados de esta tabla, recordemos que solo el algoritmo ϵ-A con $\epsilon = .0$ proporciona una inferencia óptima en términos de *subset 0/1 loss*, de igual manera que lo hace ES. Esto significa que siempre predice la combinación de etiquetas con la probabilidad condicional conjunta más alta. A pesar de ello, otros métodos aún prediciendo una combinación de etiquetas con probabilidad condicional conjunta menor para algunos ejemplos, en algunos pocos casos obtienen mejores resultados en términos de *subset 0/1 loss*, situaciones que pueden ser causadas por alguna de las siguientes razones: a) el tamaño del conjunto de test es relativamente pequeño, y/o b) los modelos f_j obtenidos para estimar la probabilidad condicional conjunta $P(\boldsymbol{y} \mid \boldsymbol{x})$ no siempre devuelven estimaciones exactas. No obstante, ϵ-A con $\epsilon = .0$ teóricamente obtendría el mejor resultado cuando se presenten condiciones perfectas (conjuntos de test grandes y modelos perfectos) y en la práctica, sería suficiente con que el conjunto de test fuera lo suficientemente grande en número de ejemplos, ya que en ese caso sería sumamente extraño que algún método mejorase los resultados obtenidos haciendo predicciones óptimas, como hace ϵ-A con $\epsilon = .0$.

Según los resultados de *subset 0/1 loss*, tal como se esperaba teóricamente, el rendimiento del algoritmo ϵ-A disminuye a medida que el valor de ϵ aumenta, obteniendo mejores resultados con $\epsilon = .0$ y peores con $\epsilon = .5$. Tal como se mencionó anteriormente ocurrieron algunas excepciones. El algoritmo ϵ-A con $\epsilon = .25$ en tres de los 12 casos alcanza al óptimo (*image*, *medical* y *scene*), mientras que en un caso lo supera ligeramente (*reuters*). Por otro lado, el algoritmo ϵ-A con $\epsilon = .5$ supera al óptimo en un solo caso, sin embargo, en ningún caso lo iguala.

Respecto al método BS, el rendimiento tiende a aumentar a medida que b

aumenta, tal como se espera teóricamente, con algunas excepciones en *bibtex*, *image*, *medical*, *reuters* y *slashdot* donde su rendimiento alcanza o supera al óptimo a partir de $b=2$. En general, con $b=2$ la mitad de los conjuntos de datos alcanzan o superan al óptimo, mientras que con $b=3$, el método logra una ligera estabilidad alcanzando o superando el óptimo en ocho de los doce conjuntos de datos. Cuando se tiene $b=10$ el método efectivamente converge al óptimo[2] alcanzado el mismo rendimiento que ϵ-A con $\epsilon = .0$, UC y ES, aunque esto implica explorar muchas más soluciones, como veremos más adelante.

Los métodos MC y EMC mejoran su rendimiento a medida que aumenta el tamaño de la muestra. Sin embargo, también presentan algunas excepciones. En este sentido, ocurre que MC alcanza un rendimiento ligeramente superior en términos de *subset 0/1 loss* para un tamaño de muestra de 50 que para uno de 100 en los casos de *emotions* y *medical*. Desafortunadamente, EMC tiene más excepciones, pero esto es bastante lógico, ya que EMC es más sensible al tamaño de la muestra. Esto se debe a que toma la máxima probabilidad condicional conjunta, que es más probable que cambie a medida que varía el tamaño de la muestra. Comparando ambos enfoques, está claro que EMC converge más rápido al óptimo que MC, ya que los valores de *subset 0/1 loss* de EMC son menores que los de MC para el mismo tamaño de muestra, especialmente cuando ésta es igual a 10. Este hecho se nota más cuando MC con una muestra de tamaño igual a 50 apenas puede alcanzar EMC con una muestra de tamaño igual a 10. Adicionalmente, EMC con un tamaño de muestra de 50 supera a MC con el mismo tamaño de muestra en todos los conjuntos de datos a excepción *image*. Por último, EMC con un valor de muestra de 100 alcanza o supera al óptimo en cinco casos, mientras que MC solo lo hace en dos casos.

Finalmente, comparando los diferentes métodos que no garantizan alcanzar soluciones óptimas, MC con un tamaño de muestra de 10 obtiene los peores resultados sin alcanzar o superar al óptimo en ningún caso. Le sigue el algoritmo ϵ-A con $\epsilon = 0.5$ (equivalente a GS y a BS con $b = 1$), superando al óptimo en un único caso, aunque en ninguna ocasión logra una solución igual al óptimo. Posteriormente, MC con tamaños de muestra de 100 y 50 alcanza o supera al óptimo en 2 y 3 ocasiones respectivamente, siendo superado por el algoritmo ϵ-A con $\epsilon = .25$ que alcanza el óptimo en 4 ocasiones. Excluyendo a BS con $b = 1$ (equivalente a GS y al algoritmo ϵ-A con $\epsilon = .5$), BS en el peor de los casos ($b=2$) logra alcanzar o superar al óptimo en 6 ocasiones, mientras que en el mejor de los casos ($b=10$) siempre logra una solución óptima. Por otro lado, EMC en el mejor de los casos (con tamaño de muestra de 100) alcanza o supera al óptimo solo en 5 ocasiones. Por todo ello, podemos concluir que, entre los métodos que no garantizan alcanzar soluciones óptimas, BS con $b > 1$ es el método más competitivo.

[2] En sus artículos Kumar et al. [2012, 2013], los autores que proponen BS indican que el algoritmo converge al óptimo para valores de $b \approx 15$. Sin embargo, en nuestros experimentos el método converge para valores más pequeños de b. La explicación que encontramos es que el proceso de selección de parámetros para cada clasificador es más exhaustiva en nuestros experimentos

Tabla 3.3: Promedio de nodos expandidos por ejemplo de test

Conjuntos	ϵ-A (.0)	ϵ-A (.25)	GS BS(1) ϵ-A(.5)	BS (2)	BS (3)	BS (10)	MC EMC (10)	MC EMC (50)	MC EMC (100)
bibtex	289.3	184.0	160.0	319.0	477.0	1575.0	1590.0	7950.0	15900.0
corel5k	1474.2	517.1	375.0	749.0	1122.0	3725.0	3740.0	18700.0	37400.0
emotions	10.7	10.8	7.0	13.0	18.0	45.0	60.0	300.0	600.0
enron	114.8	77.3	54.0	107.0	159.0	515.0	530.0	2650.0	5300.0
flags	22.6	16.3	8.0	15.0	21.0	55.0	70.0	350.0	700.0
image	7.3	7.7	6.0	11.0	15.0	35.0	50.0	250.0	500.0
mediamill*	191.8	142.4	102.0	203.0	303.0	995.0	1010.0	5050.0	10100.0
medical	46.6	46.6	46.0	91.0	135.0	435.0	450.0	2250.0	4500.0
reuters	8.2	8.3	8.0	15.0	21.0	55.0	70.0	350.0	700.0
scene	7.2	7.3	7.0	13.0	18.0	45.0	60.0	300.0	600.0
slashdot	25.3	24.9	23.0	45.0	66.0	205.0	220.0	1100.0	2200.0
yeast	26.1	26.0	15.0	29.0	42.0	125.0	140.0	700.0	1400.0

3.1.3. Estudio de nodos expandidos

La Tabla 3.3 muestra el promedio de nodos explorados por instancia de test para los diferentes métodos comparados. En la tabla se puede observar que GS (equivalente la algoritmo ϵ-A con $\epsilon = .5$ y a BS con $b = 1$) es el método que explora el menor número de nodos, ya que solo recorre una rama del árbol cuya longitud es igual al número de etiquetas más uno, debido a que la raíz del árbol se considera como un nodo explorado. Le sigue el algoritmo ϵ-A con $\epsilon = .25$, ya que BS con $b \geq 2$ aumenta rápidamente el número de nodos explorados. Entre los métodos que no garantizan alcanzar una solución óptima, los métodos MC y EMC siempre exploran más nodos que el resto, ya que expanden la misma cantidad de nodos que GS (sin considerar la raíz como un nodo explorado) multiplicado por el tamaño de la muestra, causando que el número de nodos aumente considerablemente a medida que aumenta el tamaño de la muestra. Por lo tanto, requieren explorar muchos más nodos para acercarse a una solución óptima, a pesar de que a veces su rendimiento podría ser accidentalmente mejor.

En cuanto al método que teóricamente alcanza el valor óptimo (el algoritmo ϵ-A con $\epsilon = .0$), ocurre que el algoritmo ϵ-A frecuentemente explora más número de nodos que el mismo algoritmo con otros valores para ϵ, especialmente a medida que crece el número de etiquetas. Sin embargo, el algoritmo ϵ-A con $\epsilon = .0$ siempre explora menos nodos que BS con $b = 2$, excepto en los conjuntos de datos *corel5k*, *enron* y *flags*. La distancia entre ϵ-A con $\epsilon = .0$ y BS comienza a crecer significativamente a medida que aumenta el valor b.

Tabla 3.4: Promedio de tiempo (en milisegundos) de predicción por ejemplo

Conjuntos	ES ϵ-A(.0)	ϵ-A (.25)	GS BS(1) ϵ-A(.5)	BS (2)	BS (3)	BS (10)	MC (10)	MC (50)	MC (100)	EMC (10)	EMC (50)	EMC (100)
bibtex	16.2	9.9	7.9	11.0	15.6	50.7	322	1609.9	3225.6	320.7	1602.1	3206.5
corel5k	136.0	16.1	11.0	27.8	40.4	136.1	760.7	3807.6	7629.4	756.3	3781.5	7578.1
emotions	0.6	0.6	0.4	0.5	0.6	1.3	12.0	59.0	117.7	11.7	58.5	117.2
enron	7.2	3.0	1.9	3.9	5.5	17.2	104.6	522.7	1045.7	103.9	519	1038.8
flags	1.2	0.7	0.4	0.5	0.7	1.6	13.9	69.1	138.1	13.6	68.0	136.0
image	0.5	0.5	0.4	0.4	0.5	0.9	10.0	49.1	97.8	9.8	48.5	97.0
mediamill*	11.5	5.5	3.7	7.2	10.5	33.3	199.2	996.3	1993.7	198.4	991	1982.1
medical	2.7	2.7	2.7	3.3	4.6	14.4	88.6	442.2	885.7	88.1	440.4	882.3
reuters	0.5	0.5	0.5	0.5	0.7	1.6	13.8	68.4	137.3	13.7	68.2	136.8
scene	0.5	0.5	0.4	0.5	0.6	1.3	11.9	58.7	117.2	11.8	58.4	117.1
slashdot	1.5	1.4	1.2	1.6	2.2	6.3	43.3	215.9	431.7	42.9	214.9	429.6
yeast	1.5	1.0	0.6	1.0	1.4	4.0	27.7	138	275.9	27.4	136.9	273.7

3.1.4. Estudio del tiempo computacional

La Tabla 3.4 muestra el promedio de tiempo computacional (en milisegundos) por instancia de test para los diferentes métodos comparados. El tiempo computacional para los métodos MC y EMC se ha estimado mediante el producto del tiempo de GS multiplicado por el tamaño de la muestra, aunque hay que tener en cuenta que se pueden producir ligeras diferencias en el tiempo debido a la inicialización u otros aspectos relacionados con su implementación.

Teóricamente se espera que el tiempo computacional sea mayor a medida que se exploran más nodos y los resultados de la Tabla 3.4 confirman que esto es lo que realmente sucede. Con respecto al tiempo computacional, el algoritmo ϵ-A es en realidad más rápido que BS, MC y EMC. Del mismo modo, considerando cada método por separado y variando sus parámetros se muestra que el algoritmo ϵ-A es más rápido a medida que ϵ aumenta, mientras que BS es más rápido cuando b disminuye. Sin embargo, esto no es sorprendente, ya que incrementar el valor de ϵ o disminuir el valor de b significa explorar menos nodos del árbol. Análogamente, el tiempo necesario para los métodos MC y EMC se incrementa a medida que crece el tamaño de la muestra.

No obstante, el aspecto más destacable de estos resultados es que el algoritmo ϵ-A con $\epsilon = .0$ obtiene tiempos de predicción muy competitivos, teniendo en cuenta además que ofrece predicciones óptimas. En algunos casos, su tiempo de predicción apenas se incrementa en relación al resto de versiones del mismo algoritmo cuando $\epsilon > 0$, especialmente en los conjuntos con pocas etiquetas. Solamente en los conjuntos grandes, por ejemplo *bibtex*, *corel5k* y *mediamill*, aparecen diferencias apreciables.

Tabla 3.5: *Subset 0/1 loss* considerando una muestra de diferentes órdenes de etiquetas. El número de órdenes de etiquetas están entre paréntesis al lado del nombre del conjunto de datos. El número de órdenes representan el 20 % de los posibles órdenes de etiquetas, excepto en el caso de *reuters* que representa el 10 %

Conjuntos	ϵ-A(.0)	ϵ-A(.25)	ϵ-A(.5)	BS(2)	MC(10)	EMC(10)
image(24)	**64.44 ± 2.62**	65.02 ± 2.45	66.69 ± 1.93	64.61 ± 2.72	71.05 ± 1.97	66.03 ± 2.69
emotions(144)	**71.55 ± 1.06**	71.71 ± 1.00	73.16 ± 1.17	71.75 ± 1.02	78.81 ± 1.38	72.36 ± 1.07
scene(144)	32.09 ± 1.26	32.11 ± 1.27	34.35 ± 1.68	32.09 ± 1.26	35.63 ± 1.50	**32.00 ± 1.30**
reuters(500)	**22.78 ± 0.29**	**22.78 ± 0.29**	23.69 ± 0.25	**22.78 ± 0.29**	25.29 ± 0.33	22.87 ± 0.28

3.1.5. Estudio de la variación del orden de las etiquetas

Adicionalmente, se realizaron experimentos para analizar la posible influencia de tomar diferentes órdenes de etiquetas. La Tabla 3.5 contiene el promedio de mediciones de *subset 0/1 loss* para un conjunto de órdenes de etiquetas aleatorias. Concretamente, se tomó el 20 % de los órdenes de etiquetas posibles para los conjuntos de datos con 5 o 6 etiquetas (24 y 144 diferentes órdenes de etiquetas, respectivamente) y 10 % para el conjunto de datos *reuters* que tiene 7 etiquetas (500 diferentes órdenes de etiquetas). Los resultados confirman que el algoritmo ϵ-A obtiene el mejor rendimiento. La única excepción es el conjunto de datos *scene* en el que el rendimiento de EMC es un poco mejor cuando evalúa 10 muestras.

3.2. Algoritmo A* para inferencia en PCC

En esta sección se define formalmente el algoritmo A* y los detalles que deben tenerse en cuenta a la hora de diseñar un heurístico para este algoritmo para realizar inferencia en PCC.

El algoritmo A* es la forma más conocida de búsqueda de primero el mejor Pearl [1984b], en la que se expande el mejor nodo en cada iteración, según una función de evaluación e. La particularidad del algoritmo A* es que la función de evaluación e puede verse como un función E de otras dos funciones g y h, $e(k) = E(g(k), h(k))$, donde $g(k)$ evalúa el coste de alcanzar un nodo k desde la raíz y $h(k)$ evalúa el coste de llegar a una solución (una hoja) desde el nodo k. Por lo tanto, $e(k)$ evalúa el coste total de llegar a una solución desde la raíz pasando por el nodo k. En general, es posible obtener el valor exacto de la información conocida (g), pero la información desconocida (h) se debe estimar a través de un heurístico. Para obtener una solución óptima, h no debe sobreestimar el coste real de llegar a una solución, es decir, debe ser un heurístico admisible. Esta clase de heurísticos son optimistas, porque estiman que el coste de obtener una solución es menor de lo que realmente es.

Para adaptar A* para inferencia en PCC, se debe tener en cuenta que se tienen probabilidades en lugar de costes. Por lo tanto,

(i) A* debe seleccionar el nodo con la probabilidad estimada más alta,

(ii) h no debe subestimar la probabilidad desde el nodo que se está evaluando hasta una hoja, es decir, h debe ser un heurístico admisible, y

(iii) E debe ser la función producto, es decir, $e = g \cdot h$.

Hay dos conceptos clave en el diseño de heurísticos: la admisibilidad y la dominancia. En nuestro problema de inferencia para PCC, un heurístico h es admisible si y solo si satisface que $h^*(k) \leq h(k)$ para cualquier nodo k, donde $h^*(k)$ es la probabilidad más alta (y desconocida) desde el nodo k hasta una hoja. Es decir, un heurístico será admisible siempre y cuando no subestime la probabilidad dada por $h^*(k)$. Por otro lado, entre dos heurísticos admisibles, la dominancia, si existe, nos indica cuál de ellos es mejor y por tanto hará la búsqueda más rápida. En nuestro problema, un heurístico h_1 domina a otro h_2, lo cual se denota por $h_1 \prec h_2$, cuando $h_1(k) \leq h_2(k)$ para todo nodo k. Nótese, que si ambos heurísticos son admisibles, eso quiere decir que h_1 está más próximo a h^*, con lo cual está más cerca del óptimo y se dice que es un heurístico más informado.

Considerando todos estos aspectos, e proporcionará una estimación de la probabilidad condicional conjunta $\mathbf{P}(y_1, \ldots, y_m \,|\, \boldsymbol{x})$ para optimizar *subset 0/1 loss*. Para obtener g y h, se utiliza la regla del producto de la probabilidad para la distribución conjunta de las etiquetas $\mathbf{Y} = (\mathbf{Y_1}, \mathbf{Y_2}, \ldots, \mathbf{Y_m})$ como

$$\begin{aligned} \mathbf{P}(y_1, \ldots, y_m \,|\, \boldsymbol{x}) &= \mathbf{P}(y_1, \ldots, y_j \,|\, \boldsymbol{x}) \times \\ &\quad \mathbf{P}(y_{j+1}, \ldots, y_m \,|\, \boldsymbol{x}, y_1, \ldots, y_j). \end{aligned} \tag{3.1}$$

Por lo tanto, para un nodo en el nivel j, se considera que g es la probabilidad condicional conjunta marginal $\mathbf{P}(y_1, \ldots, y_j \,|\, \boldsymbol{x})$ y que h es un heurístico que no subestime la probabilidad condicional conjunta marginal $\mathbf{P}(y_{j+1}, \ldots, y_m \,|\, \boldsymbol{x}, y_1, \ldots, y_j)$.

Antes de discutir el heurístico h, se debe tener presente que g es la misma probabilidad condicional conjunta marginal que calculan tanto el algoritmo ϵ-A como el algoritmo BS para seleccionar los nodos a expandir. Incluso, ϵ-A con $\epsilon = 0$ no solo es equivalente a UC, sino también a A* con el heurístico constante $h = 1$. Este heurístico es también admisible, ya que ninguna probabilidad es mayor que 1, sin embargo, también es el peor heurístico admisible, ya que cualquier otro heurístico admisible lo dominará y, en consecuencia, el algoritmo A* que use dicho heurístico nunca expandirá más nodos que el algoritmo A* utilizando $h = 1$.

Debido a que este heurístico para ser admisible no debe subestimar la probabilidad condicional conjunta marginal $\mathbf{P}(y_{j+1}, \ldots, y_m \,|\, \boldsymbol{x}, y_1, \ldots, y_j)$, es bastante sencillo escoger el valor máximo de dicha probabilidad para obtener un heurístico óptimo h^*:

$$\begin{aligned} h^* &= \max_{(y_{j+1}, \ldots, y_m) \in \{0,1\}^{m-j}} \mathbf{P}(y_{j+1}, \ldots, y_m \,|\, \boldsymbol{x}, y_1, \ldots, y_j) \\ &= \max_{\substack{(y_{j+1}, \ldots, y_m) \\ \in \{0,1\}^{m-j}}} \prod_{i=j+1}^{m} \mathbf{P}(y_i \,|\, \boldsymbol{x}, y_1, \ldots, y_j, y_{j+1}, \ldots, y_{i-1}). \end{aligned} \tag{3.2}$$

Sin embargo, para obtener tal máximo es necesario aplicar búsqueda exhaustiva sobre el conjunto de etiquetas, haciendo que este heurístico óptimo no sea computacionalmente aplicable. Por consiguiente, se debe renunciar a obtener un heurístico óptimo a cambio de obtener un heurístico computacionalmente tratable. Esto conlleva diseñar un heurístico que también sea admisible, pero a la vez sea menos dominante que h^*.

A partir de la definición de h^* se pueden hacer diferentes simplificaciones o relajación de condiciones que conviertan al heurístico resultante en computacionalmente viable. En este trabajo, se proponen dos formas de obtener un heurístico admisible, lo más dominante posible y lo menos costoso desde un punto de vista computacional. La primera forma involucra que el heurístico solo sea aplicable a modelos base lineales. Sin embargo, la segunda forma permite utilizar el heurístico en el caso de disponer de modelos base no lineales. En ambos casos, se introduce un parámetro que equilibra el número de nodos explorados y el tiempo de cálculo del heurístico, dando lugar a dos familias de heurísticos.

En las siguientes secciones se detallan las dos familias de heurísticos (una de ellas solamente válida para para modelos lineales y la otra válida tanto para modelos lineales como para modelos no lineales) propuestas en este trabajo que pueden emplearse con el algoritmo A*. Para cada una de ellas se describe la idea en la que se basa la construcción de estos heurísticos admisibles y la forma de parametrizar y controlar su complejidad computacional. Además, en ambos casos se indican los aspectos necesarios para optimizar su implementación, junto con un análisis de su complejidad.

3.3. Familia de heurísticos admisibles para clasificadores lineales

En esta sección se presenta un heurístico admisible que, con el objetivo de garantizar soluciones óptimas en términos de *subset 0/1 loss* y siendo lo más dominante posible para explorar la menor cantidad de nodos se limita al uso de clasificadores lineales. Dado que el cálculo de este heurístico puede incrementar en exceso el coste computacional, se introduce un parámetro que permite controlar el equilibrio entre el número de nodos explorados y el coste computacional. A continuación se exponen los detalles del diseño de este heurístico (ver Mena et al. [2015a, 2017b] para más información sobre esta familia de heurísticos).

Si $(\overline{y}_{j+1}, \ldots, \overline{y}_m)$ son los valores que definen h^*, es decir, con los que se alcanza el máximo en la Ecuación (3.2):

$$h^* = \prod_{i=j+1}^{m} \mathbf{P}(y_i \,|\, \boldsymbol{x}, y_1, \ldots, y_j, \overline{y}_{j+1}, \ldots, \overline{y}_{i-1}), \tag{3.3}$$

entonces la probabilidad con la combinación de valores $(\overline{y}_{j+1}, \ldots, \overline{y}_m)$ que maximizan el producto es siempre menor o igual que el máximo que alcanza cada

probabilidad individual, es decir,

$$\mathbf{P}(y_i \,|\, \boldsymbol{x}, y_1, \ldots, y_j, \overline{y}_{j+1}, \ldots, \overline{y}_{i-1}) \leq \max_{\substack{(y_{j+1},\ldots,y_{i-1}) \\ \in\{0,1\}^{i-1-j}}} \mathbf{P}(y_i \,|\, \boldsymbol{x}, y_1, \ldots, y_j, y_{j+1}, \ldots, y_{i-1}). \tag{3.4}$$

Por lo tanto, si se define h como

$$h = \prod_{i=j+1}^{m} \max_{\substack{(y_{j+1},\ldots,y_{i-1}) \\ \in\{0,1\}^{i-1-j}}} \mathbf{P}(y_i \,|\, \boldsymbol{x}, y_1, \ldots, y_j, y_{j+1}, \ldots, y_{i-1}), \tag{3.5}$$

es fácil deducir que h es admisible y menos dominante que h^* ($h^* \prec h$). Pero, de nuevo, h no es computacionalmente aplicable en general. Sin embargo, esto no ocurre si nos limitamos al caso de los modelos lineales. En efecto, recordemos que $\mathbf{P}(y_i \,|\, \boldsymbol{x}, y_1, \ldots, y_{i-1})$ se estima a través del modelo $f_i(\boldsymbol{x}, y_1, \ldots, y_{i-1})$. Si se utiliza, por ejemplo, regresión logística[3], es decir, una función sigmoide para transformar la salida $f_i(\boldsymbol{x}, y_1, \ldots, y_{i-1})$ en una probabilidad, entonces, la expresión para calcular la probabilidad $\mathbf{P}(y_i|\boldsymbol{x}, y_1, \ldots, y_{i-1})$ para cada uno de los valores 0 y 1 quedaría así:

$$\mathbf{P}(y_i = 1 \,|\, \boldsymbol{x}, y_1, \ldots, y_{i-1}) = \frac{1}{1 + \exp^{-f_i(\boldsymbol{x}, y_1, \ldots, y_{i-1})}}, \tag{3.6}$$

$$\mathbf{P}(y_i = 0 \,|\, \boldsymbol{x}, y_1, \ldots, y_{i-1}) = 1 - \frac{1}{1 + \exp^{-f_i(\boldsymbol{x}, y_1, \ldots, y_{i-1})}}. \tag{3.7}$$

A partir de las Ecuaciones (3.6) y (3.7), se deduce que $\mathbf{P}(y_i|\boldsymbol{x}, y_1, \ldots, y_{i-1})$ es máxima para $y_i = 1$ cuando f_i alcanza el máximo y es máxima para $y_i = 0$ cuando f_i alcanza el mínimo. Por lo tanto, debemos primeramente averiguar cuándo f_i alcanza el máximo y el mínimo, para después decantarnos por el máximo entre $y_i = 1$ e $y_i = 0$.

Centrémonos entonces en calcular el máximo y el mínimo de f_i. Considerando que f_i es un modelo lineal de PCC, entonces f_i adopta la siguiente forma

$$f_i(\boldsymbol{x}, y_1, \ldots, y_{i-1}) = \langle \boldsymbol{w}_x^i, \boldsymbol{x} \rangle + \langle \boldsymbol{w}_y^i, (y_1, \ldots, y_{i-1}) \rangle + \beta^i. \tag{3.8}$$

Desglosando el segundo término en la parte conocida (desde ℓ_1 hasta ℓ_j) y la parte desconocida (desde ℓ_{j+1} hasta ℓ_i) conduce a

$$f_i(\boldsymbol{x}, y_1, \ldots, y_j, y_{j+1}, \ldots, y_{i-1}) = \langle \boldsymbol{w}_x^i, \boldsymbol{x} \rangle + \sum_{k=1}^{j} w_{y,k}^i y_k + \sum_{k=j+1}^{i-1} w_{y,k}^i y_k + \beta^i. \tag{3.9}$$

[3]La regresión logística es probablemente el algoritmo más empleado como clasificador base para entrenar modelos de CC y PCC ya que destaca por estimar con bastante precisión probabilidades

Debido a que los valores de $\boldsymbol{x}$ y los valores de $y_1, \ldots, y_j$ son conocidos y fijos, el segundo sumatorio es el que contiene las variables para las que se debe obtener el máximo y el mínimo. Sea C_i la parte fija de f_i, es decir,

$$C_i(\boldsymbol{x}, y_1, \ldots, y_j) = \langle \boldsymbol{w}_x^i, \boldsymbol{x} \rangle + \sum_{k=1}^{j} w_{y,k}^i y_k + \beta^i. \tag{3.10}$$

Entonces se puede reescribir f_i como

$$f_i(\boldsymbol{x}, y_1, \ldots, y_j, y_{j+1}, \ldots, y_{i-1}) = C_i(\boldsymbol{x}, y_1, \ldots, y_j) + \sum_{k=j+1}^{i-1} w_{y,k}^i y_k. \tag{3.11}$$

Por consiguiente, la función cuyo máximo debe obtenerse es

$$\sum_{k=j+1}^{i-1} w_{y,k}^i y_k = w_{y,j+1}^i y_{j+1} + \ldots + w_{y,i-1}^i y_{i-1}. \tag{3.12}$$

Denotando por $K_{i,j}^+$ y $K_{i,j}^-$ los índices positivos y negativos de los coeficientes $w_{y,k}^i$ con $j+1 \leq k \leq i-1$, es decir,

$$\begin{aligned} K_{i,j}^+ &= \{k \mid j+1 \leq k \leq i-1,\ w_{y,k}^i \geq 0\} \\ K_{i,j}^- &= \{k \mid j+1 \leq k \leq i-1,\ w_{y,k}^i < 0\}, \end{aligned} \tag{3.13}$$

la expresión de la Ecuación (3.12) alcanza el máximo cuando los valores de y_k para $j+1 \leq k \leq i-1$ son

$$y_k = \begin{cases} 1 & \text{si } k \in K_{i,j}^+ \\ 0 & \text{si } k \in K_{i,j}^- \end{cases} \tag{3.14}$$

y la Ecuación (3.12) alcanza el mínimo cuando los valores de y_k para $j+1 \leq k \leq i-1$ son

$$y_k = \begin{cases} 1 & \text{si } k \in K_{i,j}^- \\ 0 & \text{si } k \in K_{i,j}^+ \end{cases} \tag{3.15}$$

Por lo tanto,

(i) Sea $y_{j+1}^{i,1}, \ldots, y_{i-1}^{i,1} \in \{0,1\}$ los valores que maximizan la Ecuación (3.12), es decir, los valores que maximizan $f_i(\boldsymbol{x}, y_1, \ldots, y_j, y_{j+1}, \ldots, y_{i-1})$ y por lo tanto, los valores que hacen que $\mathbf{P}(y_i = 1 \,|\, \boldsymbol{x}, y_1, \ldots, y_j, y_{j+1}, \ldots, y_{i-1})$ sea máximo, y

(ii) Sea $y_{j+1}^{i,0}, \ldots, y_{i-1}^{i,0} \in \{0,1\}$ los valores que minimizan la Ecuación (3.12), es decir, los valores que minimizan $f_i(\boldsymbol{x}, y_1, \ldots, y_j, y_{j+1}, \ldots, y_{i-1})$ y por lo tanto, los valores que hacen que $\mathbf{P}(y_i = 0 \,|\, \boldsymbol{x}, y_1, \ldots, y_j, y_{j+1}, \ldots, y_{i-1})$ sea máximo.

Por consiguiente, $\max_{\substack{(y_{j+1},\ldots,y_{i-1})\\ \in\{0,1\}^{i-1-j}}} \mathbf{P}(y_i \mid \boldsymbol{x}, y_1, \ldots, y_j, y_{j+1}, \ldots, y_{i-1})$ será

$$\max_{v\in\{0,1\}} \{\mathbf{P}(y_i = v \mid \boldsymbol{x}, y_1, \ldots, y_j, y_{j+1}^{i,v}, \ldots, y_{i-1}^{i,v})\}. \tag{3.16}$$

Observemos que $y_k^{i,1} = 1 - y_k^{i,0}$ para $j+1 \leq k \leq i-1$ y que de acuerdo con la definición anterior de la función sigmoide, si $f_i(\boldsymbol{x}, y_1, \ldots, y_j, y_{j+1}^{i,1}, \ldots, y_{i-1}^{i,1}) \geq -f_i(\boldsymbol{x}, y_1, \ldots, y_j, y_{j+1}^{i,0}, \ldots, y_{i-1}^{i,0})$, entonces el máximo de $\mathbf{P}(y_i = v \mid \boldsymbol{x}, y_1, \ldots, y_j, y_{j+1}^{i,v}, \ldots, y_{i-1}^{i,v})$ se alcanza cuando $v = 1$ y en caso contrario cuando $v = 0$. Por lo tanto, la expresión final del heurístico será

$$h_l = \prod_{i=j+1}^{m} \max_{v\in\{0,1\}} \{\mathbf{P}(y_i = v \mid \boldsymbol{x}, y_1, \ldots, y_j, y_{j+1}^{i,v}, \ldots, y_{i-1}^{i,v})\}. \tag{3.17}$$

Donde el subíndice l indica que el heurístico es únicamente aplicable a clasificadores lineales.

Los heurísticos h^* y h_l solo difieren en los valores de $y_{j+1}, \ldots, y_{i-1}$. En el primero, los mismos valores son comunes para todos los factores del producto, mientras que en el último, estos valores dependen de cada término i del producto, y por lo tanto, pueden ser diferentes, lo que hace que h_l no sea un heurístico óptimo. Por otro lado, el coste de calcular h_l es de orden polinómico, a diferencia del de h^* que es de orden exponencial.

La Figura 3.1 muestra un ejemplo de cálculo de h_l en el que se puede ver que la probabilidad condicional conjunta marginal hasta del nodo (1) es $g = 0.6$. Para calcular h_l, primero se evalúan las probabilidades condicionales marginales máximas $P(y_2 \mid \boldsymbol{x}, y_1 = 1)$ proporcionadas por $f_2(\boldsymbol{x}, y_1)$ y $P(y_3 \mid \boldsymbol{x}, y_1 = 1, y_2)$ proporcionadas por $f_3(\boldsymbol{x}, y_1, y_2)$ que son 0.6 y 1.0 respectivamente. Luego, se realiza su producto aplicando la regla del producto de la probabilidad para estimar la probabilidad condicional conjunta marginal de ese nodo hasta una hoja. En la figura se puede observar que el máximo en cada nivel no corresponde a la misma rama del árbol. En otras palabras, el máximo en el primer nivel corresponde a $y_2 = 1$, mientras que el máximo en el segundo nivel es obtenido por $P(y_3 \mid \boldsymbol{x}, y_1 = 1, y_2 = 0)$ cuando $y_3 = 1$ estableciendo $y_2 = 0$. Esto es lo que hace que el heurístico h_l no sea el óptimo h^*.

La Figura 3.2 muestra el camino seguido por A* utilizando h_l. Como se puede apreciar, el algoritmo llega a la hoja óptima como teóricamente se esperaba. Comparando este gráfico con el de la Figura 1.4(a) que ilustra ϵ-A con $\epsilon = 0$ y teniendo en cuenta la propiedad relacionada con la dominancia, ϵ-A con $\epsilon = 0$ (equivalente a A* con $h = 1$ o UC) explora más nodos que A* con h_l.

Finalmente, el algoritmo A* con h_l estima perfectamente el minimizador de riesgo para *subset 0/1 loss*, tal como lo hacen ϵ-A con $\epsilon = .0$ y ES. Incluso, A* con h_l expande igual o menos nodos, ya que h_l es más dominante que el heurístico $h = 1$ ($h_l \prec h = 1$). Obviamente, calcular h_l es más costoso que calcular $h = 1$ o aplicar

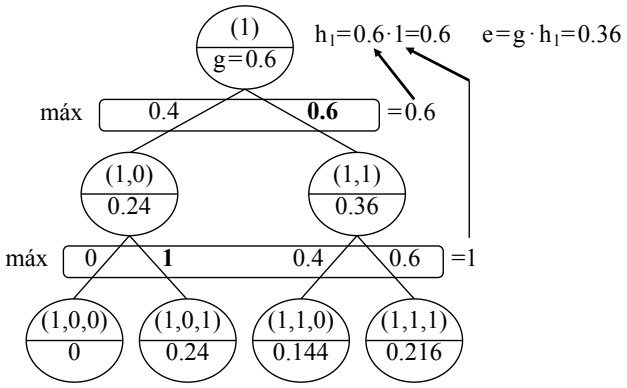


Figura 3.1: Un ejemplo del cálculo del heurístico h_l

UC. La cuestión entonces es si este tiempo computacional adicional compensa la garantía teórica de expandir menos nodos.

3.3.1. Limitación de la profundidad

En la sección anterior se comentó que tanto h_l como $h = 1$ son heurísticos admisibles y que utilizar h_l teóricamente garantiza que el algoritmo A* explore menos nodos que si se utiliza $h=1$. Sin embargo, el coste de calcular h_l podría ser demasiado alto en comparación con considerar solo un heurístico constante como el heurístico $h=1$. A partir de esta observación, surge la idea de incluir un parámetro d para limitar la profundidad del heurístico y establecer un equilibrio entre el número de nodos explorados y el coste computacional de calcular el heurístico. Por lo tanto, para un nodo de nivel j y un valor de d con $0 \leq d \leq m - j$, solo los términos $\mathbf{P}(y_i \,|\, \boldsymbol{x}, y_1, \ldots, y_j, y^i_{j+1}, \ldots, y^i_{i-1})$ de h_l se evalúan utilizando f_i para los nodos desde el nivel $j + 1$ hasta el nivel $j + d$, mientras que el resto de términos desde $j + d + 1$ hasta el nivel m se acotan superiormente por 1, es decir,

$$h^d_l = \prod_{i=j+1}^{j+d} \mathbf{P}(y_i \,|\, \boldsymbol{x}, y_1, \ldots, y_j, y^i_{j+1}, \ldots, y^i_{i-1}) \cdot \prod_{i=j+d+1}^{m} 1, \tag{3.18}$$

o equivalentemente

$$h^d_l = \prod_{i=j+1}^{j+d} \mathbf{P}(y_i \,|\, \boldsymbol{x}, y_1, \ldots, y_j, y^i_{j+1}, \ldots, y^i_{i-1}). \tag{3.19}$$

Está claro que h_l es más dominante que h^d_l ($h_l \prec h^d_l$) y que h^d_l sigue siendo admisible. A su vez, h^d_l es más dominante que $h=1$ ($h^d_l \prec h=1$). Por lo tanto, se

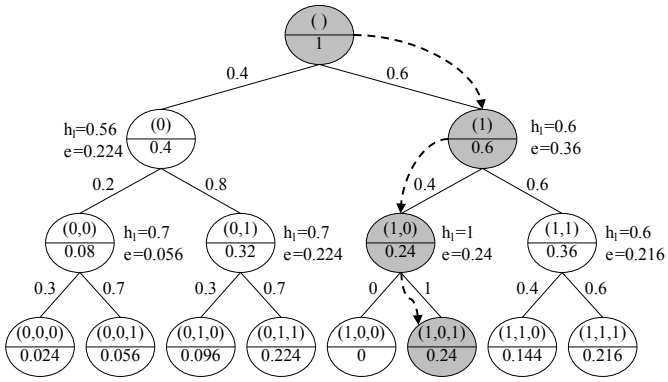


Figura 3.2: Un ejemplo de A* utilizando h_l. Las flechas punteadas muestran el camino recorrido por el algoritmo. Los valores de g se proporcionan dentro de cada nodo

espera que si se utiliza el heurístico h_l^d con $0 \leq d \leq m-j$ y $1 \leq j \leq m-1$, el algoritmo A* explore un número de nodos mayor o igual que si se utiliza h_l, pero un número de nodos menor o igual que si se utiliza $h=1$. De hecho, h_l^d encapsula ambos heurísticos. En efecto, por un lado, tomar el valor máximo de d conduce a h_l^{m-j} con $1 \leq j \leq m-1$ (este heurístico se denota aquí como h_l^∞), que es de hecho el heurístico h_l. Por otro lado, tomar el valor mínimo de d conduce a h_l^0, que es de hecho el heurístico $h=1$. En general, $h_l^{d_1}$ es más dominante que $h_l^{d_2}$ si $d_1 > d_2$. Sin embargo, el tiempo computacional de evaluar h_l^d aumenta a medida que d aumenta. Por lo tanto, seleccionando un valor adecuado de d se puede obtener un equilibrio entre el número de nodos explorados y el tiempo de cómputo empleado para evaluar el heurístico.

El caso de $d=1$ tiene un interés especial, ya que el heurístico también es válido para modelos no lineales f_j. La forma del heurístico es

$$h_l^1 = \prod_{i=j+1}^{j+1} \mathbf{P}(y_i \,|\, \boldsymbol{x}, y_1, \ldots, y_j, y_{j+1}^i, \ldots, y_{i-1}^i) \cdot \prod_{i=j+2}^{m} 1, \tag{3.20}$$

que al simplificar conduciría a

$$h_l^1 = \mathbf{P}(y_{j+1} \,|\, \boldsymbol{x}, y_1, \ldots, y_j). \tag{3.21}$$

Como se ve, este heurístico h_l^1 supone simplemente evaluar el modelo del nodo $f_j(\boldsymbol{x}, y_1, \ldots, y_j)$, por lo tanto, no tiene ninguna restricción sobre la linealidad de f_j. La Figura 3.3 muestra un ejemplo del camino seguido por el algoritmo A* cuando se toma como heurístico h_l^1. Como se ve, el algoritmo A* explora más

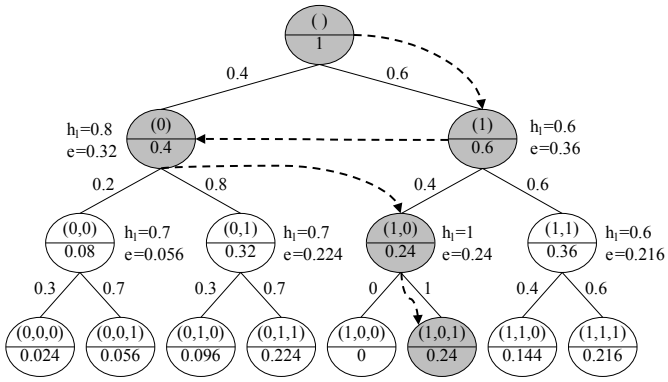


Figura 3.3: Un ejemplo de A* utilizando h_l^1. Las flechas punteadas muestran el camino seguido por el algoritmo. Los valores de g se proporcionan dentro de cada nodo

nodos con este heurístico que cuando se utiliza h_l, pero el tiempo computacional de evaluación del heurístico se reduce considerablemente, como se demostrará más adelante en los experimentos.

3.3.2. Implementación y análisis de complejidad

En este apartado se muestran los detalles de la implementación del heurístico h_l^d. El Algoritmo 1 muestra el pseudocódigo de A* con h_l^d. En este algoritmo se incluyen algunos elementos específicos para optimizar la implementación de A* cuando h_l^d se utiliza como heurístico, en concreto:

(i) Los modelos f_i (parámetros $\boldsymbol{w}_x^i$, $\boldsymbol{w}_y^i$ y $\boldsymbol{\beta}^i$) están ordenados según el orden de la etiqueta de la cadena.

(ii) Todos valores que se necesitan en más de una ocasión se calculan solo una vez y se almacenan para que puedan ser reutilizados:

  (a) La evaluación de $\langle \boldsymbol{w}_x^i, \boldsymbol{x} \rangle$ para cada modelo lineal f_i, que es común para todos los nodos correspondientes a f_i, se calcula una vez y se guarda en la variable WX (línea 2).

  (b) Los conjuntos K^+ y K^-, que son los mejores valores de y_k para la parte desconocida del heurístico, se calculan solo una vez antes de iniciar el bucle principal de A* (líneas 3-8).

(iii) Q almacena tuplas de cuatro elementos: combinación de etiquetas, nivel, e y g. Q se almacena en un vector redimensionable cuyas posiciones se reutilizan.

Algoritmo 1 Pseudocódigo de la implementación del algoritmo A* con h_l^d

```
1:  function A*
    Input: x, [W, β] un modelo lineal CC, m, d, BlockSize
    Output: Combinación de etiquetas con la mayor probabilidad conjunta dado x
2:      WX ← W_x * x      // Computa ⟨w_x, x⟩ para todas las etiquetas
3:      K+ ← AllocMemory(m)
4:      K− ← AllocMemory(m)
5:      for label = [2 : m] do      // No hay atributos de etiquetas en el 1°, inicio 2° nivel
6:          K+[label] ← ⟦W_y[label] >= 0⟧
7:          K−[label] ← ⟦W_y[label] < 0⟧
8:      end for
9:      Q ← AllocMemory(BlockSize)      // Tuplas {Labels, Level, e, g}
10:     Q[1] ← {[ ], 0, 1, 1}      // Nodo raíz, conjunto de etiquetas vacío, nivel 0, e = g=1
11:     last ← 1      // Último elemento usado en Q
12:     while true do
13:         [Best, Position] ← Max(Q, last)
14:         if Best.Level = m then
15:             return Best.Labels      // Nodo hoja
16:         end if
17:         level ← Best.Level + 1
18:         P ← 1/(1 + exp(−(WX[level] + β[level] + ⟨w_y[level], Best.Labels⟩)))
19:             // Hijo izquierdo
20:         h_l^d ← Heuristic(d, level, [Best.Labels 0], K+, K−, WX, W_y, x)      // Eq(3.19)
21:         Q[Position] ← {[Best.Labels 0], level, Best.g * (1 − P) * ĥ^d, Best.g * (1 − P)}
22:             // Hijo derecho
23:         last ← last + 1
24:         if last > Q.size then
25:             Q ← resize(Q, BlockSize)
26:         end if
27:         h_l^d ← Heuristic(d, level, [Best.Labels 1], K+, K−, WX, W_y, x)      // Eq(3.19)
28:         Q[last] ← {[Best.Labels 1], level, Best.g * P * ĥ^d, Best.g * P}
29:     end while
30: end function
```

El hijo de la izquierda se almacena en la posición de su padre (línea 21). No se requiere información de los padres para obtener la solución final, debido a que Q contiene toda la información requerida para cada nodo.

(iv) El diseño de Q nos permite optimizar la función Max, la operación para obtener el mejor nodo a expandir. La función Max solo tiene que evaluar los primeros elementos ($1..last$) de Q (línea 13), porque el resto del vector no se utiliza.

(v) El Algoritmo 1 contiene varias funciones auxiliares para hacer el pseudocódigo más corto y más fácil de entender. Sin embargo, en el programa actual todas estas funciones fueron codificadas en línea, haciendo que el código sea más rápido porque se evitan todas las llamadas a funciones con parámetros grandes, como la función *Heuristic*.

A pesar de las propiedades teóricas de optimalidad del algoritmo A*, puede no ser útil en algunos problemas porque puede explorar un número exponencial de nodos con respecto a la profundidad del árbol Pearl [1984a]; Russell and Norvig [2003]. Solo bajo ciertas condiciones del heurístico, es posible obtener un algoritmo

con una complejidad teórica menor que exponencial. Este sería el caso en el que se fuese capaz de probar que el heurístico h satisface la siguiente condición para cualquier nivel j con respecto al heurístico óptimo h^*:

$$|h^*(j) - h(j)| \leq \mathcal{O}(\log h^*(j)). \tag{3.22}$$

En general, ocurre que este error es por lo menos lineal con respecto al coste del camino, por lo tanto, conduce a un crecimiento exponencial. Sin embargo, al diseñar un buen heurístico se pueden obtener grandes beneficios en relación con aquellos que no proporcionan información heurística. Ese es el caso del heurístico h_l^d con respecto al heurístico $h=1$, que no proporciona ningún tipo de información heurística, a pesar de ser una mejora de ES.

En lo que sigue y para analizar la Ecuación (3.22) con respecto a la familia de heurísticos heurísticos h_l^d, este apartado se centra en el heurístico h_l^∞, debido a que es el heurístico más dominante de la familia. De acuerdo con los resultados en los que el ruido está aumentando (ver Figuras 3.8 y 3.9 de la Sección 3.6), no está claro que h_l^∞ sea exponencial en absoluto como en cambio sí muestran claramente el resto de los algoritmos, al menos para los conjuntos de datos tomados. Este comportamiento de h_l^∞ arroja luz sobre la realización de un análisis más profundo sobre otras situaciones de mayor incertidumbre, como puede ser la presencia de ruido. En este sentido, consideremos un caso teórico cuando es máxima la diferencia entre h_l^∞ y h^*. Por ejemplo, esta situación ocurre (ver Figura 3.4) cuando el primer nivel tiene probabilidad 0 en la rama izquierda y 1 en la rama derecha, todas las probabilidades del subárbol izquierdo son 0 (para las ramas izquierdas) y 1 (para las ramas derechas) y todas las probabilidades del subárbol derecho son 1/2 (para todas las ramas). En este caso, el error puede ser delimitado como sigue para cualquier nivel j

$$|h^*(j) - h_l^\infty(j)| \leq |1/2^{j-1} - 1|. \tag{3.23}$$

Este límite tiende a 1 a medida que j crece, por lo tanto, en este caso, no es posible garantizar que la complejidad teórica es menor que exponencial. Sin embargo, estos casos extremos difícilmente ocurrirán. Recordemos que las probabilidades se evalúan a partir de los modelos, donde la descripción $\boldsymbol{x}$ de los ejemplos juega un papel importante. Esta descripción es igual para obtener las probabilidades de todos los nodos del mismo nivel, entonces, tales probabilidades tal vez no difieran tanto entre ellas como en el caso extremo considerado.

3.4. Familia de heurísticos admisibles para clasificadores no lineales

Debido a que la familia de heurísticos propuestos en el apartado anterior es solo aplicable cuando se usan clasificadores lineales como métodos base en PCC, en este apartado se presenta un heurístico que elimina tal restricción, y por ende,

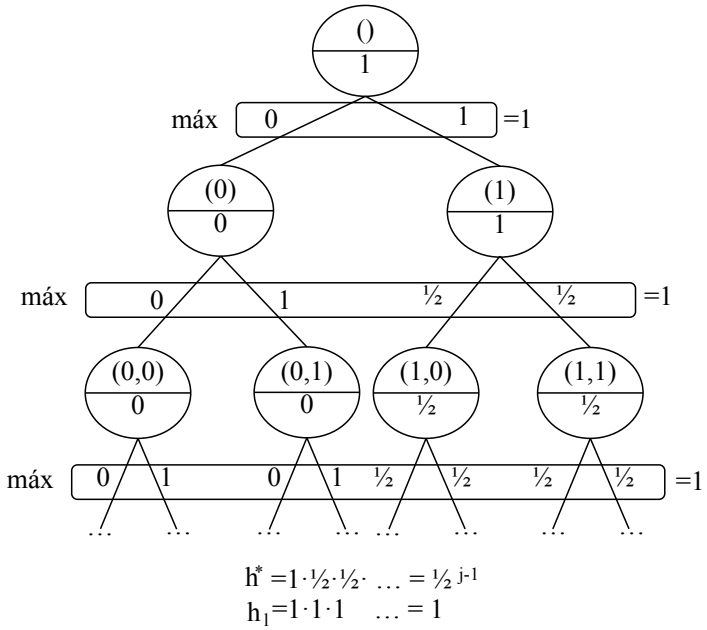


Figura 3.4: Un ejemplo de peores casos para heurística h_l^d

es aplicable tanto a clasificadores lineales como a clasificadores no lineales. Esta familia de heurísticos se caracteriza con respecto al anterior en que:

(i) También es admisible, por lo tanto, garantiza predicciones óptimas en términos de *subset 0/1 loss*.

(ii) También incluye un parámetro de profundidad para equilibrar el número de nodos explorados y el tiempo computacional de calcular el heurístico.

(iii) Supera la limitación de ser aplicable solo cuando se usan clasificadores lineales como métodos base en PCC.

(iv) Es más dominante que h_l^d para la misma profundidad, por lo tanto, explora teóricamente menos nodos.

(v) Se reduce el número de modelos que deben ser evaluados y supera a la familia de heurísticos h_l^d tanto en número de nodos explorados, como en tiempo computacional, tal como veremos en los experimentos.

Tal como se mencionó en el apartado 3.2, el cálculo de h^* definido en la Ecuación (3.2) no es computacionalmente aceptable, ya que significa aplicar una búsqueda exhaustiva para el subconjunto de etiquetas que van desde el nodo

actual hasta las hojas. La idea principal aquí consiste en relajar la optimalidad del heurístico h^* a cambio de diseñar un heurístico computacionalmente tratable, pero que se siga basando en realizar una búsqueda exhaustiva. Un heurístico basado en búsqueda heurística presenta algunas ventajas interesantes:

(i) Por definición, sabemos que dicho heurístico dará el óptimo al menos para la profundidad que se calcule. Esto es fundamental desde el punto de vista de la dominancia del heurístico resultante.

(ii) Las estimaciones que se realicen se harán sobre caminos reales, y no como ocurría en la familia de heurísticos anterior en los que el cálculo del heurístico podía estar formado por arcos que no formaban parte del mismo camino.

(iii) El hecho de calcularse sobre caminos reales permite que las evaluaciones de todos los modelos que se realicen para computar el heurístico de un nodo puedan ser reutilizadas por sus hijos. Como veremos en la Sección 3.4.2, una cuidadosa implementación permite reducir significativamente su coste computacional teniendo en cuenta este detalle.

Aunque en principio la idea de basarse en búsqueda exhaustiva pueda parecer computacionalmente intratable, sin embargo, en la inferencia en PCC aparece otro elemento crucial que permite reducir la complejidad del heurístico a la mitad. Es el hecho de que los dos nodos hijos de un nodo representan sucesos contrarios, con lo cuál para calcular sus probabilidades solamente necesitamos evaluar un modelo. Es decir, basta con calcular la probabilidad $\mathbf{P}(y_i = 1 \,|\, \boldsymbol{x}, y_1, \ldots, y_{i-1})$, ya que $\mathbf{P}(y_i = 0 \,|\, \boldsymbol{x}, y_1, \ldots, y_{i-1})$ es simplemente $1 - \mathbf{P}(y_i = 1 \,|\, \boldsymbol{x}, y_1, \ldots, y_{i-1})$. Si unimos este hecho, junto con la posibilidad de limitar la profundidad de la búsqueda exhaustiva, nos permite obtener un heurístico computacionalmente viable y más informado, para la misma profundidad, que el propuesto en la sección anterior.

Los detalles y los resultados de los experimentos completos de este heurístico se publicaron en Mena et al. [2015b, 2017a].

3.4.1. Limitación de la profundidad

Debido a que aplicar búsqueda exhaustiva para el subconjunto de etiquetas $\{\ell_{j+1}, \ldots, \ell_m\}$ es computacionalmente inviable, se proponer incluir un parámetro d para limitar la profundidad del heurístico y controlar el coste computacional. En este sentido, se lleva a cabo una búsqueda exhaustiva solo hasta d niveles de profundidad en el árbol. Esto es posible gracias al hecho de que cualquier probabilidad está delimitada por 1. Por lo tanto, para un nodo del nivel j y un valor de d tal que $0 \leq d \leq m - j$, las probabilidades $\mathbf{P}(y_i \,|\, \boldsymbol{x}, y_1, \ldots, y_j, y_{j+1}, \ldots, y_{i-1})$ de la Ecuación (3.2) se obtienen:

$$h^* = \max_{\substack{(y_{j+1},\ldots,y_m) \\ \in \{0,1\}^{m-j}}} \prod_{i=j+1}^{m} \mathbf{P}(y_i \,|\, \boldsymbol{x}, y_1, \ldots, y_j, y_{j+1}, \ldots, y_{i-1}).$$

(i) De forma exacta, aplicando una búsqueda exhaustiva desde nivel $j+1$ hasta $j+d$, es decir, evaluando $\mathbf{P}(y_i \mid \boldsymbol{x}, y_1, \ldots, y_j, y_{j+1}, \ldots, y_{i-1})$ para estos niveles y tomando la combinación que alcanza el máximo.

(ii) Haciendo una estimación de las mismas utilizando la cota 1 desde nivel $j+d+1$ hasta m.

Por lo tanto, se define un heurístico h^d_{nl}, en el que el subíndice nl indica que el heurístico también es válido para clasificadores no lineales, como:

$$h^d_{nl} = \max_{(y_{j+1},\ldots,y_{j+d})\in\{0,1\}^d} \prod_{i=j+1}^{j+d} \mathbf{P}(y_i \mid \boldsymbol{x}, y_1, \ldots, y_j, y_{j+1}, \ldots, y_{i-1}) \cdot \prod_{i=j+d+1}^{m} 1 \quad (3.24)$$

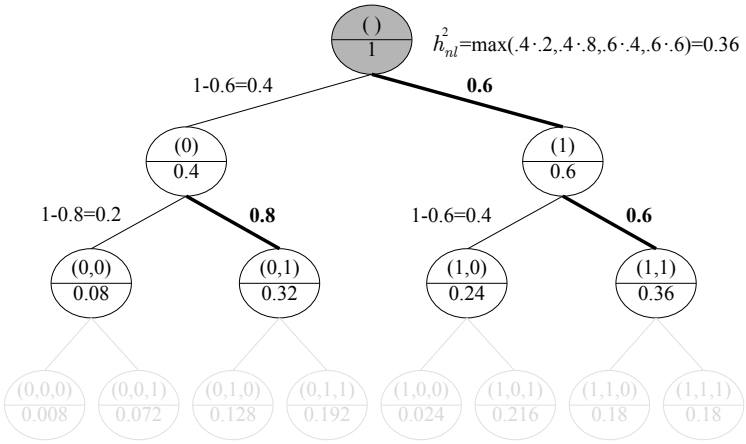
o equivalente a

$$h^d_{nl} = \max_{(y_{j+1},\ldots,y_{j+d})\in\{0,1\}^d} \prod_{i=j+1}^{j+d} \mathbf{P}(y_i \mid \boldsymbol{x}, y_1, \ldots, y_j, y_{j+1}, \ldots, y_{i-1}) \quad (3.25)$$

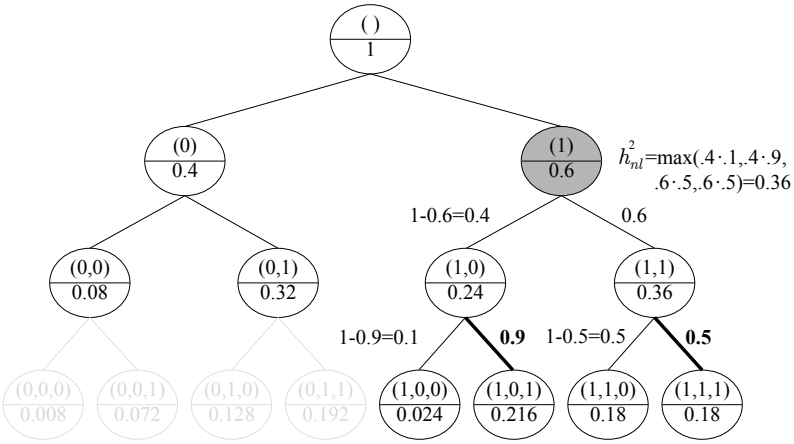
De esta manera, h^d_{nl} sigue siendo admisible, pero menos dominante que el heurístico óptimo h^*. El caso particular de $d=0$ es el heurístico constante $h=1$. Por lo tanto, h^d_{nl} es a su vez más dominante que h^0_{nl}. En general, h^d_{nl} se vuelve más dominante a medida que d aumenta, y entonces, llega a explorar menos nodos, pero a costa de incrementar el coste computacional. Este coste computacional depende principalmente del número de modelos que deben ser evaluados para calcular el heurístico. Teóricamente, h^d_{nl} requiere evaluar $2^d - 1$ modelos, lo que significa evaluar muchos más modelos que el número de modelos evaluados por h^d_l $(2d-1)$. Sin embargo, una implementación apropiada de h^d_{nl} (ver apartado siguiente) puede reducir ese número a 2^{d-1}, haciendo que el coste computacional de ambos heurísticos sea comparable cuando $d \leq 4$.

La Figura 3.5 ilustra las formas de calcular el heurístico h^d_{nl} para $d=2$ en dos casos diferentes: para un nodo padre y uno de sus hijos. En cuanto al primer caso (ver Figura 3.5(a)), el valor de h^2_{nl} para el nodo raíz es 0.36, que corresponde a la probabilidad condicional conjunta más alta de los nodos del segundo nivel (combinación de etiquetas (1,1)). Este valor es un límite superior del valor óptimo $h^*=0.211$ que corresponde al camino de la combinación de etiquetas (1,0,1). Para el segundo caso (ver Figura 3.5(b)), solo es necesario evaluar los clasificadores del último nivel porque los clasificadores de niveles anteriores se evaluaron cuando se calculó el heurístico para el nodo padre.

La razón por la que h^d_l no es capaz de tratar modelos no lineales es porque h^d_l se calcula teniendo en cuenta los pesos de los modelos lineales (ver Ecuación 3.19). En el caso de h^d_{nl}, estos pesos no son necesarios porque el cálculo de h^d_{nl} aplica con prudencia los modelos sobre las combinaciones de etiquetas en el nivel más profundo del subárbol correspondiente. Por ejemplo, para calcular h^2_{nl} para el nodo en gris de la Figura 3.5(b), solo se necesita aplicar el modelo f_3 sobre

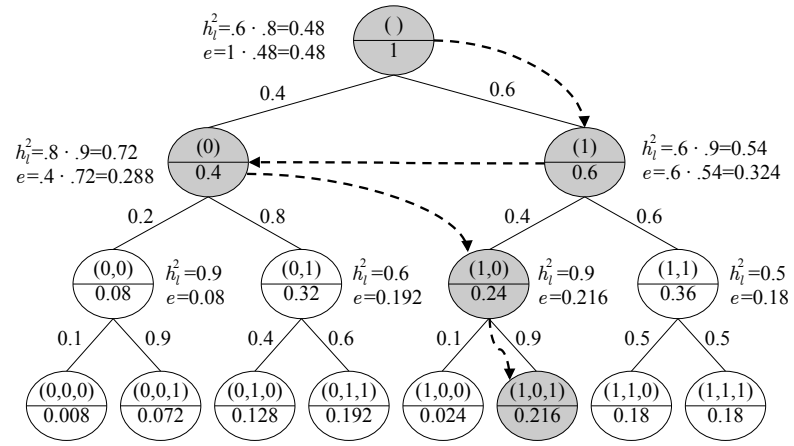


(a) Un nodo padre

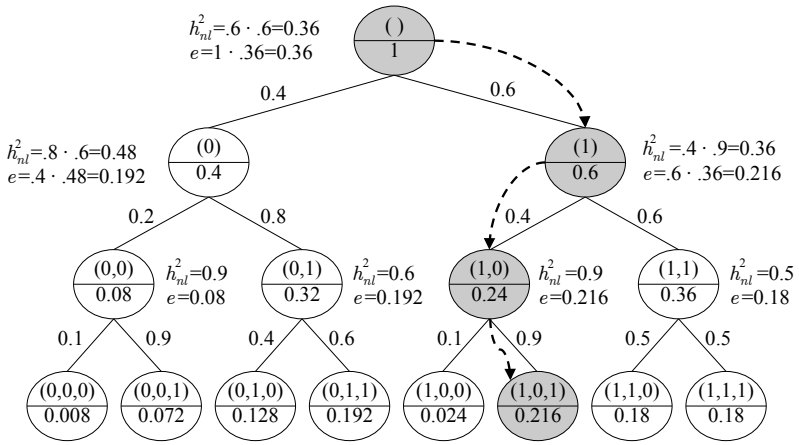


(b) Un nodo hijo

Figura 3.5: Un ejemplo del heurístico h^d_{nl} para $d=2$ niveles de profundidad, por lo tanto, se expanden solo dos niveles de profundidad desde allí. Los nodos en gris oscuro son los nodos para los que se calcula el heurístico: (a) Nodo raíz (nodo único de nivel $j=0$) (b) Un nodo hijo (el nodo derecho de dos nodos existentes en el nivel $j=1$). Las líneas en negrita indican las evaluaciones de los clasificadores que deben calcularse para obtener el valor del heurístico. Estas evaluaciones son solo aquellas que corresponden al clasificador del nivel más profundo, excepto para el nodo raíz en el que se deben evaluarse todos los modelos necesarios.



(a) h_l^d



(b) h_{nl}^d

Figura 3.6: Camino (flechas punteadas) seguido por el algoritmo A* utilizando (a) h_l^d y (b) h_{nl}^d, ambos con niveles de profundidad $d=2$. Ambos heurísticos alcanzan la solución óptima, pero h_{nl}^2 requiere explorar menos nodos que h_l^2

las combinaciones de etiquetas $(y_1 = 1,\ y_2 = 0)$ y $(y_1 = 1,\ y_2 = 1)$. Con estas dos evaluaciones, es posible calcular las probabilidades $\mathbf{P}(y_3 = 1 \,|\, \boldsymbol{x}, y_1 = 1, y_2 = 0) = 0.9$ y $\mathbf{P}(y_3 = 1 \,|\, \boldsymbol{x}, y_1 = 1, y_2 = 1) = 0.5$. Las otras dos probabilidades se obtienen trivialmente como $\mathbf{P}(y_3 = 0 \,|\, \boldsymbol{x}, y_1 = 1, y_2 = 0) = 1 - \mathbf{P}(y_3 = 1 \,|\, \boldsymbol{x}, y_1 = 1, y_2 = 0) = 1 - 0.9 = 0.1$ y $\mathbf{P}(y_3 = 0 \,|\, \boldsymbol{x}, y_1 = 1, y_2 = 1) = 1 - \mathbf{P}(y_3 = 1 \,|\, \boldsymbol{x}, y_1 = 1, y_2 = 1) = 1 - 0.5 = 0.5$.

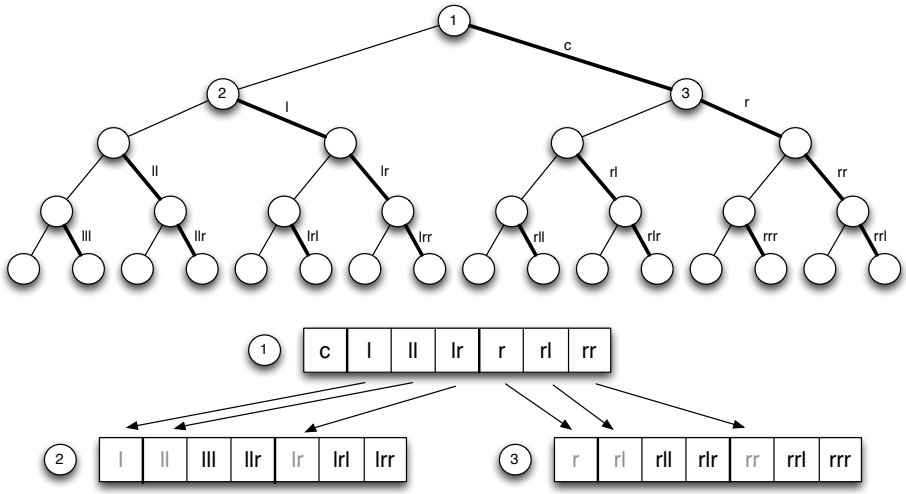


Figura 3.7: Cálculo óptimo del heurístico. El vector heurístico representa las probabilidades calculadas del árbol utilizando un recorrido en preorden. El ejemplo supone que el nodo 1 se expande, por lo que sus hijos (nodos 2 y 3) deben agregarse a la lista Q. El cálculo del heurístico para ambos nodos reutiliza la información contenida en la estructura del heurístico del nodo 1. Solo deben calcularse las probabilidades del nivel inferior (posiciones en negrita en los vectores 2 y 3)

Por último, la Figura 3.6 muestra los caminos seguidos por A* cuando se toman como heurísticos h_l^2 (parte superior) y h_{nl}^2 (parte inferior). En primer lugar, conviene destacar que ambos heurísticos alcanzan una solución óptima (única en este caso), que se corresponde con la combinación de etiquetas (1,0,1), cuya probabilidad condicional conjunta es la más alta. Sin embargo, h_{nl}^2 explora menos nodos que h_l^2, ya que sigue el camino más directo. Esto se debe al hecho de que las estimaciones calculadas por h_{nl}^2 son más exactas. En este caso, esto significa que las estimaciones son más pequeñas y más cercanas al óptimo. Observe que esto ocurre para los tres nodos superiores del árbol de la Figura 3.6 (0.36 frente a 0.48, 0.48 frente a 0.72 y 0.36 frente a 0.54).

3.4.2. Implementación y análisis de complejidad

Esta sección propone una implementación completa del algoritmo A* utilizando h_{nl}^d como heurístico, incluyendo un algoritmo eficiente para calcular h_{nl}^d que reduce el número de modelos que deben ser evaluado desde $2^d - 1$ a 2^{d-1}. La idea clave es que cada nodo almacena las probabilidades necesarias para calcular el heurístico (ver Figura 3.7). Entonces, cuando un nodo es expandido, estas probabilidades se copian y se reutilizan para calcular el heurístico para sus hijos. Aplicando

Algoritmo 2 Pseudocódigo de la implementación del algoritmo A* utilizando h^d_{nl}

```
 1: function A*
    Input: m, d, x, {[W = {w^i}, β = {β^i}], i = 1, ..., m} un modelo CC
    Output: Combinación de etiquetas con la mayor probabilidad conjunta dado x
 2:     v ← Inic(x, [W, β], d)      // Probabilidades de h^d_nl para el nodo raíz
 3:     e ← Max(CompProbs(1_{2^d}, v, d))
 4:     Q[1] ← {[ ], 0, 1, e, v}      // Raíz, conjunto vacío de etiquetas, nivel 0, g = 1
 5:     while true do
 6:         Node ← Max(Q)
 7:         if Node.Level = m then
 8:             return Node.Labels      // Nodo hoja
 9:         end if
10:         level ← Node.Level + 1
11:         P ← Node.v[1]      // Probabilidad de que la etiqueta sea 1
12:             // Hijo izquierdo
13:         g ← Node.g * (1 − P)
14:         vl ← CopyHeuristicVector(Left, v, d)
15:         CompNewProbs(vl, d, x, [W, β], [Node.Labels 0], level)
16:         h^d_nl ← Max(JointProbs(1_{2^d}, vl, d))
17:         Insert(Q, {[Node.Labels 0], level, g, g * h^d_nl, vl})
18:             // Hijo derecho
19:         g ← Node.g * P
20:         vr ← CopyHeuristicVector(Right, v, d)
21:         CompNewProbs(vr, d, x, [W, β], [Node.Labels 1], level)
22:         h^d_nl ← Max(JointProbs(1_{2^d}, vr, d))
23:         Insert(Q, {[Node.Labels 1], level, g, g * h^d_nl, vr})
24:     end while
25: end function
```

esta idea, solo es necesario calcular las probabilidades del nivel d porque el resto ya se calcularon para el nodo padre. Esta implementación hace que h^d_{nl} evalúe menos o igual número de modelos que h^d_l cuando $d \leq 3$. El Algoritmo 2 muestra el pseudocódigo de la implementación del algoritmo A* utilizando h^d_{nl}. En este pseudocódigo se puede observar que:

(i) Inicialmente, en el nodo raíz, se calculan todas las probabilidades hasta el nivel d (ver *CompProbs* en la línea 3).

(ii) Cuando se expande un nodo que no sea el nodo raíz, la información de su padre se copia y se reutiliza (ver *CopyHeuristicVector* en las líneas 14 y 20).

(iii) En ese caso, solo es necesario evaluar el clasificador del último nivel para obtener las nuevas probabilidades necesarias (ver *CompNewProbs* en las líneas 15 y 21 respectivamente para los hijos izquierdo y derecho del nodo expandido).

(iv) Los dos hijos de un nodo expandido se incluyen en la lista de nodos candidatos para ser expandidos (ver *Insert* en las líneas 17 y 23), tomando cada vez el nodo con la probabilidad condicional conjunta más alta (ver línea 6).

Algoritmo 3 Copia de las probabilidades reutilizables del vector $\boldsymbol{v}$

```
1:  function CopyHeuristicVector
    Input: Child, v, d
    Output: v' un vector incompleto, copiando los valores necesarios de v
2:      v' ← AllocMemory(2^d − 1)
3:      l ← 2    // Principio de la parte izquierda
4:      r ← 1 + 2^(d−1)    // Principio de la parte derecha
5:      if Child = Left then    // Principio de la parte que debe copiarse
6:          i ← l
7:      else
8:          i ← r
9:      end if
10:     v'[1] ← v[i]    // Probabilidad del nivel más alto
11:     i ← i + 1
12:     ElementsPerLevel ← 1    // El siguiente nivel tiene solo 1 elemento
13:     level ← 3
14:     while level ≤ d do
15:         for j = [1 : Elementsperlevel] do
16:             v'[l] ← v[i]
17:             l ← l + 1, i ← i + 1
18:         end for
19:         for j = [1 : Elementsperlevel] do
20:             v'[r] ← v[i]
21:             r ← r + 1, i ← i + 1
22:         end for
23:         ElementsPerLevel ← ElementsPerLevel * 2
24:         level = level + 1
25:     end while
26: end function
```

(v) El algoritmo termina cuando el nodo expandido es una hoja o equivalentemente el nivel del nodo expandido es m.

El Algoritmo 3 detalla el procedimiento para copiar las probabilidades reutilizables del vector $\boldsymbol{v}$ ($CopyHeuristicVector$). En concreto, cuando el heurístico debe ser calculado para uno nodo, primero copia las probabilidades previamente calculadas para su padre. Adicionalmente, el Algoritmo 4 describe la función recursiva que calcula todas las probabilidades conjuntas del vector de probabilidades ($\boldsymbol{v}$) de un nodo. Particularmente, se aprovecha de las probabilidades previamente guardadas de tal forma que ninguna probabilidad se calcula dos veces. Finalmente, el Algoritmo 5 presenta la función iterativa para calcular las probabilidades que faltan del vector $\boldsymbol{v}$ para un nuevo nodo. Por lo tanto, esta función evalúa el clasificador del nivel $j+d$ con el fin de obtener las nuevas probabilidades necesarias para calcular el heurístico para el nodo actual. También almacena dichas probabilidades para ser reutilizadas en caso de que deba ser calculado más adelante el heurístico para cualquier descendiente de ese nodo.

Todos estos algoritmos tienen sentido si $d \geq 2$, ya que para el caso particular de $d=1$, donde solo se explora un nivel, la implementación puede ser considerablemente simplificada. De esta manera, el cálculo de h^d_{nl} significa calcular 2^{d-1} nuevas evaluaciones del clasificador del último nivel $(j+d)$ cada vez que se calcula el heurístico para un nodo. En contraste con las $2d-1$ nuevas evaluaciones de d clasificadores diferentes para h^d_l Mena et al. [2017b] (ver Figura 3.7). Por

Algoritmo 4 Función recursiva para calcular todas las probabilidades conjuntas dado un vector de probabilidades ($\boldsymbol{v}$) de un nodo dado

function $JointProbs$
Input: $\boldsymbol{P}$, $\boldsymbol{v}$, d // P es el vector de las probabilidades conjuntas
Output: Cada elemento de P multiplicado por las probabilidades correspondientes de $\boldsymbol{v}$
  if $d = 1$ **then**
    return $[\boldsymbol{P}[1] * (1 - \boldsymbol{v}[1]), \boldsymbol{P}[2] * \boldsymbol{v}[1]]$
  else
    $\boldsymbol{v}' \leftarrow \boldsymbol{v}[2 : end]$
    return $[(1-\boldsymbol{v}[1]) * JointProbs(\boldsymbol{P}[1 : len(\boldsymbol{P})/2], \boldsymbol{v}'[1 : len(\boldsymbol{v}')/2], d-1),$
    $\boldsymbol{v}[1] * JointProbs(\boldsymbol{P}[len(\boldsymbol{P})/2 + 1 : end], \boldsymbol{v}'[len(\boldsymbol{v}')/2 + 1 : end], d - 1)]$
  end if
end function

Algoritmo 5 Función iterativa para calcular la probabilidades que faltan del vector $\boldsymbol{v}$ para un nuevo nodo

function $CompNewProbs$
Input: $\boldsymbol{v}$, d, $Labels$, $Level$, $\boldsymbol{x}$, $\{[\mathbf{W} = \{\boldsymbol{w}^i\}, \boldsymbol{\beta} = \{\beta^i\}], i = 1, \ldots, m\}$ un modelo CC
Output: $\boldsymbol{v}$
  $LabelComb \leftarrow BinaryPerm(d - 1)$ // Todos los patrones de $d-1$ bits
  $j \leftarrow 1$
  for $i = [1 + 2^{d-1} - 2^{d-2} : 2^{d-1}]$ **do** // Probabilidades lado izquierdo
    $\boldsymbol{v}[i] \leftarrow Prob(\boldsymbol{x}, [\mathbf{W}, \boldsymbol{\beta}], [Labels\ LabelComb(j)], Level)$
    $j \leftarrow j + 1$
  end for
  $j \leftarrow 1$
  for $i = [1 + len(\boldsymbol{v}) - 2^{d-2} : len(\boldsymbol{v})]$ **do** // Probabilidades lado derecho
    $\boldsymbol{v}[i] \leftarrow Prob(\boldsymbol{x}, [\mathbf{W}, \boldsymbol{\beta}], [Labels\ LabelComb(j)], Level)$
    $j \leftarrow j + 1$
  end for
end function

lo tanto, ambos heurísticos necesitan obtener las mismas evaluaciones de nuevos clasificadores para $d=1$ (1 evaluación). El número de evaluaciones de clasificadores de h^d_{nl} y h^d_l son aproximadamente igual para $2 \leq d \leq 4$ (2 vs 3 evaluaciones cuando $d=2$, 4 vs 5 evaluaciones cuando $d=3$ y 8 vs 7 evaluaciones cuando $d=4$). Desde $d=5$, el número de evaluaciones de clasificadores que se deben realizar comienza a diferir considerablemente, siendo 16 evaluaciones para h^d_{nl} y 9 para h^d_l. Este hecho sugiere no tomar valores altos de d, porque de hacerlo la ventaja obtenida por el carácter dominante del heurístico no compensaría el coste computacional involucrado en su cálculo. A cambio de eso, i) h^d_{nl} es más dominante que h^d_l para el mismo nivel de profundidad d, y ii) a diferencia de h^d_l, h^d_{nl} se puede utilizar cuando los modelos base de PCC son no lineales.

3.5. Análisis experimental de los métodos basados en búsqueda heurística

En esta sección se amplía el estudio presentado en la Sección 3.1 incluyendo las familias de heurísticos descritos en este capítulo como alternativa para mitigar el coste computacional de realizar inferencia en PCC sin perder optimalidad. Se

ha excluido los métodos MC y EMC, debido a que se demostró en la Sección 3.1 y en Mena et al. [2016] que su eficiencia está muy por debajo de ϵ-A, además de no garantizar la obtención de una predicción óptima. De igual manera, se ha excluido el método BS con $b=10$, debido a que tiene un coste computacional elevado y no se consiguen mejoras significativas a la hora de alcanzar predicciones óptimas.

Los experimentos se llevaron a cabo sobre los mismos conjuntos de datos de referencia descritos en la Sección 3.1.1. El algoritmo A* utilizando los heurísticos propuestos en este trabajo se compara con otros algoritmos de inferencia para PCC de la literatura. En concreto, se compara con GS, ϵ-A con diferentes valores de $\epsilon \in \{.0, .25, .5\}$ y BS para valores de $b \in \{1, 2, 3\}$. Conviene recordar que GS, ϵ-A (.5) y BS con $b=1$ exploran el mismo camino en el árbol, alcanzando resultados idénticos. De forma análoga, ϵ-A con $\epsilon=.0$, UC y el algoritmo A* con los heurísticos h_l^0 y h_{nl}^0, que equivalen al heurístico $h=1$, obtienen también predicciones idénticas.

3.5.1. Comparativa utilizando modelos lineales y no lineales

En este estudio, la configuración de los experimentos es la misma que la llevada a cabo en la Sección 3.1.1. La Tabla 3.6 muestra los resultados para *subset 0/1 loss* de los métodos comparados utilizando como base un clasificador lineal tal como se hizo en la Sección 3.1. Debido a que A* utilizando cualquiera de los heurísticos obtiene predicciones óptimas iguales a las obtenidas por ϵ-A con $\epsilon=.0$, aquí se pueden extraer las mismas conclusiones a las que se llegó anteriormente en el apartado 3.1.2.

Los *p-values* del *Wilcoxon signed-rank* (ver Tabla 3.7) confirman los resultados obtenidos anteriormente. Es decir:

(i) Los métodos que garantizan alcanzar predicciones óptimas son significativamente mejores que ϵ-A con $\epsilon = .25$ y GS (o BS con $b = 1$ o ϵ-A(.5)) para $\alpha=0.01$, y que BS con $b=2$ y BS con $b=3$ pero para $\alpha=0.05$.

(ii) Las predicciones de BS se acercan a las predicciones óptimas a medida que b aumenta, aunque para estos conjuntos de ejemplos, las diferencias entre BS con $b=2$ y BS con $b=3$ no son significativas.

En segundo lugar, se ha probado el heurístico h_{nl}^d con el resto de métodos utilizando *Support Vector Machines* SVM Vapnik [1995] como clasificador no lineal base con salida probabilística y con *Radial Basis Function* (RBF) como *kernel*. Los parámetros C de SVM y G de RBF Chang and Lin [2001] se optimizaron aplicando un *grid search* con valores de $C, G \in \{10^{-2}, \ldots, 10^{1}\}$ utilizando *Brier score* (3.1) como función de estimación con una validación cruzada de 2 particiones y 5 repeticiones. El resto de aspectos de configuración de experimentos es idéntico al llevado a cabo en la Sección 3.1.1. El heurístico h_l^d no se incluye en esta comparación debido a que no se puede calcular cuando el modelo está formado por clasificadores no lineales.

Tabla 3.6: Resultados de *subset 0/1 loss* utilizando como base un clasificador lineal para PCC (regresión logística). Los resultados que alcanzan o mejoran el óptimo se muestran en negrita

Conjuntos	h_l^d / h_{nl}^d UC / ϵ-A(.0)	ϵ-A(.25)	GS/BS(1) ϵ-A(.5)	BS(2)	BS(3)
bibtex	**81.92**	81.95	82.19	**81.88**	**81.92**
corel5k	**97.48**	98.62	98.90	98.30	98.04
emotions	**71.16**	71.82	72.83	72.16	71.32
enron	**83.14**	84.26	85.43	83.43	83.37
flags	**87.13**	87.16	**86.13**	88.21	**87.13**
image	**68.35**	**68.35**	69.75	**68.35**	**68.35**
mediamill*	**83.86**	84.58	85.80	84.10	**83.86**
medical	**30.37**	**30.37**	30.67	**30.37**	**30.37**
reuters	**22.73**	**22.70**	23.60	**22.69**	**22.73**
scene	**31.86**	**31.86**	33.28	31.90	**31.86**
slashdot	**51.80**	52.22	54.49	**51.77**	**51.80**
yeast	**76.95**	77.62	79.77	**76.83**	77.08

Tabla 3.7: Resultados de pruebas de rangos con signo de Wilcoxon utilizando como base un clasificador lineal para PCC (regresión logística)

Métodos óptimos	otros	p-value
UC/A*/ϵ-A(.0)	ϵ-A(.25)	0.0117
UC/A*/ϵ-A(.0)	GS/BS(1)/ϵ-A(.5)	0.0034
UC/A*/ϵ-A(.0)	BS(2)	0.1309
UC/A*/ϵ-A(.0)	BS(3)	0.1250

La Tabla 3.8 muestra los resultados para *subset 0/1 loss* de los diferentes métodos comparados. En la primera columna aparecen los métodos UC, ϵ-A con $\epsilon = .0$ y el algoritmo A* utilizando cualquier heurístico admisible; todos ellos proporcionan soluciones óptimas que obviamente coinciden. Como se puede apreciar, hay 4 casos de 12 para los cuales ϵ-A con $\epsilon = .25$ iguala las predicciones óptimas. Mientras que por otro lado, no existe ningún caso donde GS (o BS con $b = 1$ o ϵ-A con $\epsilon = .5$) alcance o mejore los resultados de los algoritmos que teóricamente obtienen soluciones óptimas. BS con $b = 2$ mejora los resultados óptimos en dos casos (*image* y *slashdot*) y los iguala en otros dos (*medical* y *scene*). Finalmente, BS con $b = 3$ obtiene resultados iguales en 8 casos y nunca es mejor.

Los resultados de las pruebas de rangos con signo de Wilcoxon (ver Tabla 3.9)

Tabla 3.8: Resultados de *subset 0/1 loss* utilizando como base para PCC un clasificador no lineal (SVM con kernel RBF). Los resultados que alcanzan o mejoran el óptimo se muestran en negrita

Conjuntos	h^d_{nl} / UC ϵ-A(.0)	ϵ-A(.25)	GS/BS(1) ϵ-A(.5)	BS(2)	BS(3)
bibtex	**81.20**	81.22	81.34	81.22	**81.20**
corel5k	**98.02**	98.48	98.54	98.22	98.10
emotions	**66.10**	**66.10**	69.79	66.77	**66.10**
enron	**84.20**	84.84	86.78	84.49	84.31
flags	**82.45**	83.47	83.45	82.97	**82.45**
image	**58.20**	58.45	60.20	**58.05**	**58.20**
mediamill*	**84.36**	**84.36**	85.02	84.48	84.40
medical	**28.73**	**28.73**	29.14	**28.73**	**28.73**
reuters	**21.98**	22.00	23.14	22.03	**21.98**
scene	**31.15**	**31.15**	32.40	**31.15**	**31.15**
slashdot	**51.27**	51.53	53.15	**51.24**	51.27
yeast	**77.24**	77.62	79.77	77.33	**77.24**

Tabla 3.9: Resultados de pruebas de rangos con signo de Wilcoxon utilizando como base para PCC un clasificador no lineal (SVM con kernel RBF)

Métodos óptimos	otros	p-value
UC/A*/ϵ-A(.0)	ϵ-A(.25)	0.0078
UC/A*/ϵ-A(.0)	GS/BS(1)/ϵ-A(.5)	0.0005
UC/A*/ϵ-A(.0)	BS(2)	0.0488
UC/A*/ϵ-A(.0)	BS(3)	0.2500

proporcionan en este caso las mismas conclusiones extraídas al usar clasificadores lineales como métodos base, pero los *p-values* son ligeramente menores. La diferencia sigue siendo significativa para ϵ-A con $\epsilon = .25$ al nivel de $\alpha = 0.05$ y para GS (o BS con $b{=}1$ o ϵ-A con $\epsilon{=}.5$) con $\alpha{=}0.01$.

Comparando los resultados obtenidos cuando se utilizan clasificadores lineales y no lineales como método base PCC, hay 8 conjuntos de datos de 12 en los que el uso de clasificadores no lineales como modelos de base mejoran los resultados de *subset 0/1 loss* respecto a la aplicación de clasificadores lineales. Por lo tanto, como se esperaba, esto significa que hay problemas que requieren modelos más complejos. Estos resultados apoyan la utilidad del heurístico h^d_{nl} propuesto en este trabajo porque es el único método existente en la literatura hasta el momento basado en el algoritmo A* capaz de realizar inferencia con PCC no lineal.

Tabla 3.10: Promedio de nodos explorados incluyendo el algoritmo A* con h^d_{nl} y h^d_l. El número entre paréntesis es el número de etiquetas del conjunto de datos

Conjuntos	h^1_{nl} h^1_l	h^2_{nl}	h^3_{nl}	h^2_l	h^3_l	h^∞_l	UC ϵ-A(.0)	ϵ-A(.25)	GS/BS(1) ϵ-A(.5)	BS(2)	BS(3)
bibtex(159)	283.3	277.7	272.4	278.0	273.2	215.9	289.3	184.0	160.0	319.0	477.0
Corel5k(374)	1456.9	1440.6	1425.4	1443.1	1431.6	1338.6	1474.2	517.1	375.0	749.0	1122.0
emotions(6)	9.1	8.0	7.4	8.3	7.8	7.7	10.7	10.8	7.0	13.0	18.0
enron(53)	110.3	106.1	102.2	106.5	103.3	75.5	114.8	77.3	54.0	107.0	159.0
flags(7)	16.4	10.2	8.7	12.7	10.4	8.8	22.6	16.3	8.0	15.0	21.0
image(5)	6.6	6.1	6.0	6.3	6.1	6.1	7.3	7.7	6.0	11.0	15.0
mediamill*(101)	188.2	184.8	181.5	185.6	183.7	178.9	191.8	142.4	102.0	203.0	303.0
medical(45)	46.6	46.6	46.5	46.6	46.6	46.4	46.6	46.6	46.0	91.0	135.0
reuters(7)	8.2	8.1	8.0	8.1	8.1	8.1	8.2	8.3	8.0	15.0	21.0
scene(6)	7.2	7.1	7.0	7.2	7.1	7.1	7.2	7.3	7.0	13.0	18.0
slashdot(22)	25.0	24.7	24.4	24.9	24.9	24.8	25.3	24.9	23.0	45.0	66.0
yeast(14)	23.6	21.4	20.0	22.8	22.4	21.0	26.1	26.0	15.0	29.0	42.0

3.5.2. Análisis computacional

En este caso, se restringió el estudio al uso de clasificadores lineales (regresión logística) como métodos base para poder incluir la familia de heurísticos h^d_l en la comparación. El resto de los métodos previamente existentes en la literatura también se incluyen para proporcionar una comparación más amplia.

3.5.2.1. Análisis de nodos expandidos

En este apartado se estudian los heurísticos h^d_{nl} y h^d_l en términos del número de nodos explorados.

La Tabla 3.10 muestra el número de nodos explorados por cada método. Tal como ya se mencionó en la Sección 3.1, GS (o ϵ-A con $\epsilon = .5$ o BS con $b = 1$) explora el menor número de nodos, ya que solo sigue un camino en el árbol, entonces explora tantos nodos como la cantidad de etiquetas del conjunto de datos (más uno si también se considera la raíz). Entre los métodos que no garantizan obtener predicciones óptimas, ϵ-A con $\epsilon = .25$ tiende a expandir menos nodos que BS con $b = 2$, pero a costa de ofrecer resultados peores en términos de *subset o/1 loss*. BS con $b = 3$ expande el mayor número de nodos, a pesar de obtener predicciones cerca del óptimo.

Entre los métodos que alcanzan soluciones óptimas, el algoritmo A* utilizando cualquiera de los heurísticos propuestos y para cualquier valor de $d > 0$ siempre explora una cantidad de nodos menor o igual a la que explora el algoritmo ϵ-A con $\epsilon = .0$, confirmando experimentalmente lo que teóricamente se había discutido y es que el algoritmo A* con cualquiera de los heurísticos es más dominante que ϵ-A con $\epsilon = .0$. Por otro lado, entre los heurísticos propuestos en este trabajo, h^d_{nl} siempre explora menos nodos que h^d_l para el mismo valor de d (conviene mencionar que para $d \in \{0, 1\}$ ambos heurísticos exploran los mismos caminos). La diferencia

Tabla 3.11: Promedio de tiempo de predicción en milisegundos. El número entre paréntesis es el número de etiquetas del conjunto de datos

Conjuntos	h^1_{nl} h^1_l	h^2_{nl}	h^3_{nl}	h^2_l	h^3_l	h^∞_l	UC ϵ-A(.0)	ϵ-A(.25)	GS/BS(1) ϵ-A(.5)	BS(2)	BS(3)
bibtex(159)	19.5	24.8	30.2	31.4	41.1	840.7	16.2	9.9	7.9	11.0	15.6
Corel5k(374)	168.4	190.8	229.4	257.8	333.3	23834.4	136.0	16.1	11.0	27.8	40.4
emotions(6)	0.6	0.7	0.7	0.8	0.9	1.1	0.6	0.6	0.4	0.5	0.6
enron(53)	8.5	14.5	18.4	14.1	18.5	108.4	7.2	3.0	1.9	3.9	5.5
flags(7)	1.1	0.9	0.9	1.4	1.5	1.5	1.2	0.7	0.4	0.5	0.7
image(5)	0.5	0.5	0.6	0.6	0.7	0.7	0.5	0.5	0.4	0.4	0.5
mediamill*(101)	14.4	18.9	23.3	24.2	32.6	484.7	11.5	5.5	3.7	7.2	10.5
medical(45)	3.3	4.6	5.7	5.7	7.7	49.5	2.7	2.7	2.7	3.3	4.6
reuters(7)	0.6	0.7	0.8	0.8	1.0	1.3	0.5	0.5	0.5	0.5	0.7
scene(6)	0.5	0.6	0.7	0.7	0.9	1.0	0.5	0.5	0.4	0.5	0.6
slashdot(22)	1.8	2.4	2.9	2.9	3.9	12.4	1.5	1.4	1.2	1.6	2.2
yeast(14)	1.6	2.0	2.3	2.6	3.4	7.5	1.5	1.0	0.6	1.0	1.4

en nodos explorados entre h^d_{nl} y h^d_l crece significativamente conforme aumenta la cantidad de etiquetas del conjunto de datos. Esto demuestra que h^d_{nl} es el heurístico más dominante. Incluso, es más dominante a medida que aumenta el nivel de profundidad d. Sorprendentemente, el número de nodos explorados por A* con h^d_{nl} es cercano al de GS en algunas situaciones. Sin embargo, a veces el número de nodos explorados por A* con h^d_{nl} para $d \geq 1$ es bastante similar al de ϵ-A con $\epsilon = .0$. La razón puede deberse a que las probabilidades proporcionadas por los clasificadores f_i toman valores cercanos a 0 o a 1, entonces, puede no haber tanta incertidumbre y la búsqueda puede estar altamente dirigida.

3.5.2.2. Análisis del tiempo de predicción

Tal como se mencionó anteriormente, teóricamente se espera que el tiempo de predicción sea mayor a medida que se exploran más nodos. En la Tabla 3.11 se puede observar que esto efectivamente ocurre para los algoritmos ϵ-A, UC, GS y BS. Sin embargo, no sucede para el algoritmo A* con los diferentes heurísticos propuestos, debido a que, aunque explore menos nodos que otros métodos, su tiempo computacional puede estar por encima de éstos a causa del tiempo empleado en el cálculo del heurístico. De hecho, el algoritmo A* utilizando cualquiera de los heurísticos propuestos es más lento que ϵ-A con $\epsilon = .0$, aunque este último explore muchos más nodos. En todo caso, el algoritmo A* con h^d_{nl} siempre es más rápido que cuando se usa h^d_l para el mismo valor de d, lo que implica que el coste computacional de h^d_{nl} es menor que el de h^d_l.

Analizando cada método por separado, como ya se mencionó en la Sección 3.1, el algoritmo $\epsilon-$A es más rápido a medida que ϵ aumenta, mientras que BS es más rápido cuando b disminuye. En cuanto al algoritmo A*, tanto utilizar h^d_{nl} como utilizar h^d_l es más rápido a medida que d disminuye. Finalmente, el algoritmo A* con h^1_{nl} o equivalentemente h^1_l, es decir para $d = 1$, puede considerarse una

buena alternativa para garantizar un rendimiento óptimo en términos *subset 0/1 loss* y conservar un equilibrio entre el número de nodos explorados y el tiempo computacional.

3.6. Comportamiento en presencia de ruido

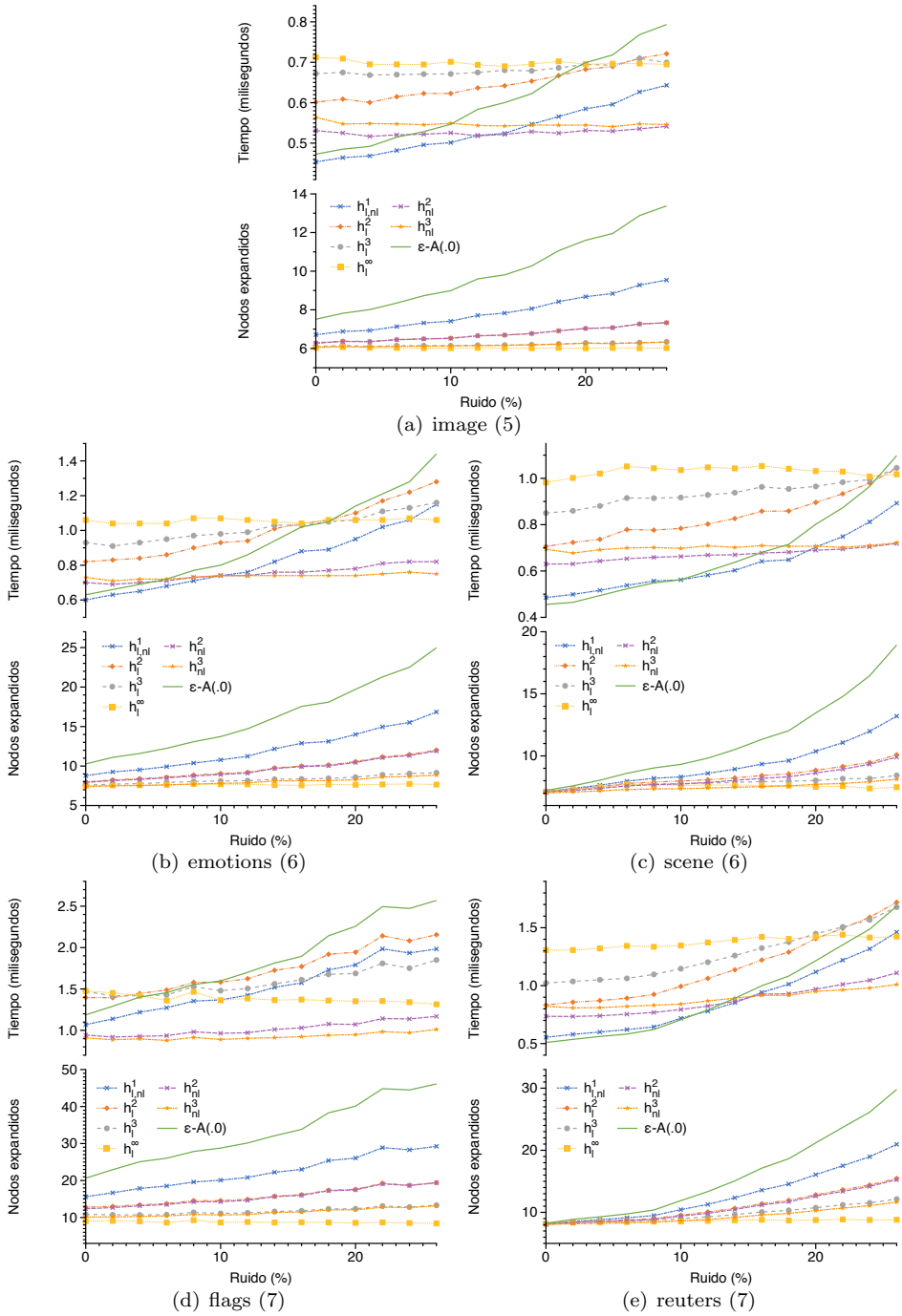
En apartados anteriores se ha hecho alusión en diferentes ocasiones a la existencia de mayor o menor incertidumbre a la hora de explorar el árbol de probabilidades, dando lugar a que la búsqueda sea más o menos dirigida. Esta conclusión surge a raíz de observar que el número de nodos explorados por el algoritmo A* es bastante bajo en comparación con el número de etiquetas o profundidad del árbol, lo que significa que el algoritmo A* realiza poco *backtracking* en el árbol. Este comportamiento hace pensar que las probabilidades proporcionadas por los modelos puedan no estar tan cerca de 0.5 o, aún más, puedan estar cerca de 0 ó 1. Esto da pie a analizar más en profundidad el potencial del uso del algoritmo A* con las familias de heurísticos propuestos. De esta manera, se pensó en añadir ruido haciendo que las probabilidades fueran más cercanas a 0.5, donde el heurístico más informado puede mostrar su potencial.

La inclusión del ruido se llevó a cabo en las etiquetas de los conjuntos de datos, cambiando un porcentaje de sus valores, pasando de ser relevantes a ser irrelevantes y viceversa. Esto significa que para un ejemplo se puede haber cambiado la relevancia de todas las etiquetas, ninguna o solo una parte de ellas.

El porcentaje de ruido incluido varía de 0 % a 26 % para conjuntos de datos con menos de 22 etiquetas. Sin embargo, los experimentos no terminaron en un tiempo prudencial aplicando todos los porcentajes de este rango para conjuntos de datos con más de 45 etiquetas. En particular, el porcentaje máximo considerado en este aspecto para las etiquetas de *medical* (45 etiquetas), *enron* (53 etiquetas), *mediamill* (101 etiquetas) y *bibtex* (159 etiquetas) fue de 18 %, 10 %, 6 % y 4 % respectivamente.

Las Figuras 3.8 y 3.9 muestran, respectivamente, para los conjuntos de datos con pocas etiquetas (de 5 a 7) y para conjuntos de datos con más etiquetas (de 14 a 159), el número de nodos expandidos y el tiempo computacional empleado por el algoritmo A* con el heurístico h_l^d para diferentes valores de d $(1, 2, 3, \infty)$, el algoritmo A* con el heurístico h_{nl}^d para d $(1, 2, 3)$ y el algoritmo $\epsilon-$A con $\epsilon=0$, para los diferentes porcentajes de ruido (debido a que h_l^1 y h_{nl}^1 son equivalentes se muestran juntas). Como se puede observar, únicamente se han tenido en cuenta en este estudio los métodos que garantizan predicciones óptimas. Obviamente, tanto el número de nodos explorados como el tiempo computacional debería aumentar en mayor o menor grado a medida que incrementa el porcentaje de ruido, debido a que aumenta la incertidumbre y por ende el problema de búsqueda se hace más complejo.

Como se puede ver en las gráficas, tanto el número de nodos explorados y, consecuentemente, el tiempo computacional, para el algoritmo $\epsilon-$A con $\epsilon = 0$



(a) image (5)

(b) emotions (6)

(c) scene (6)

(d) flags (7)

(e) reuters (7)

Figura 3.8: Número de nodos expandidos y tiempo computacional (en milisegundos) para el algoritmo A* con los heurísticos h_l^d y h_{nl}^d y el algoritmo $\epsilon-$A con $\epsilon=0$, para diferentes porcentajes de ruidos en los conjuntos de datos con pocas etiquetas (desde 5 hasta 7)

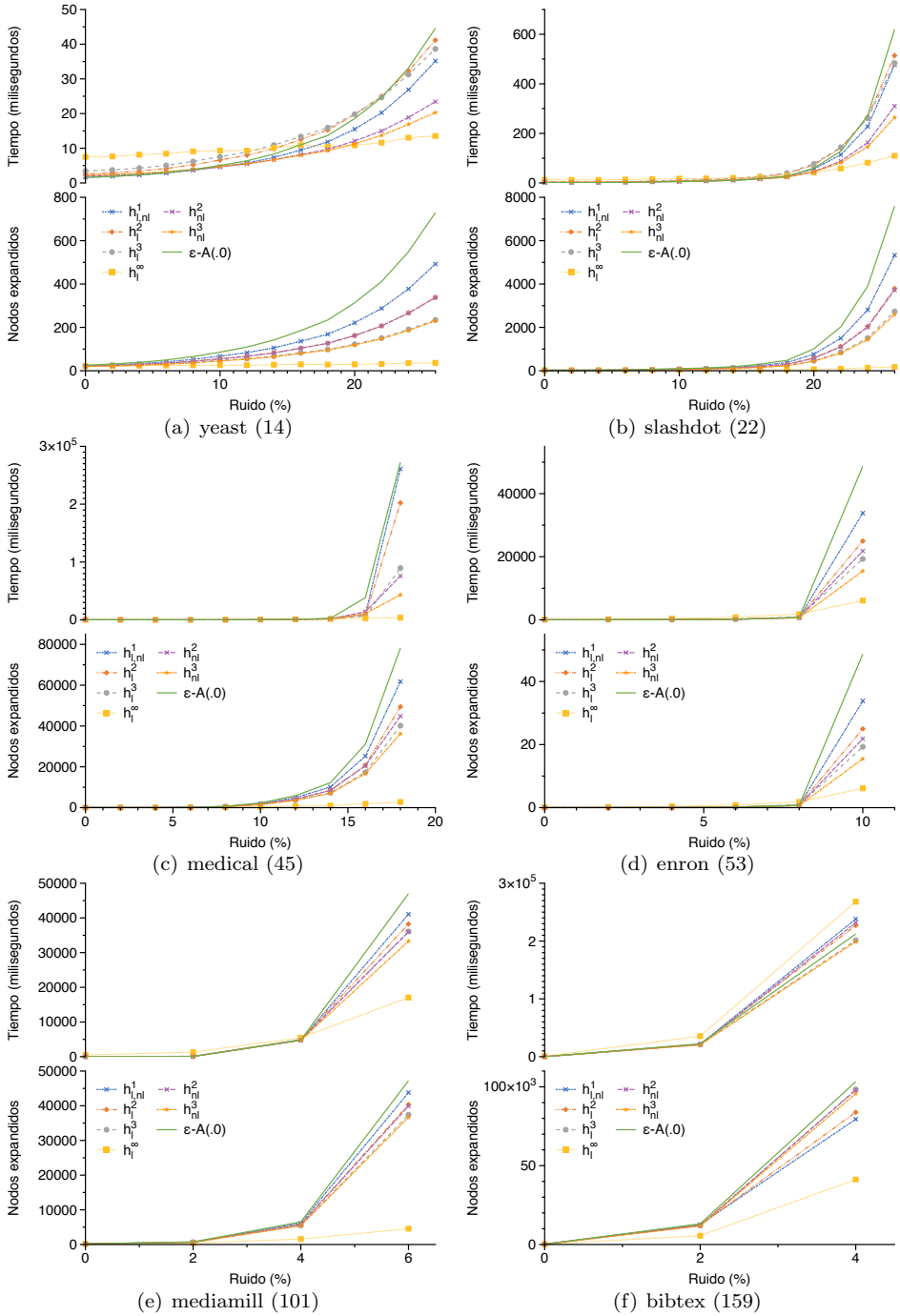


Figura 3.9: Número de nodos expandidos y tiempo computacional (en milisegundos) para el algoritmo A* con los heurísticos h_l^d y h_{nl}^d y el algoritmo $\epsilon-$A con $\epsilon=0$, para diferentes porcentajes de ruidos en los conjuntos de datos con mayor cantidad de etiquetas (desde 14 hasta 159)

siempre aumenta considerablemente conforme aumenta el porcentaje de ruido, ya que para este método existe una relación directamente proporcional entre el número de nodos explorados y el tiempo computacional. De hecho, para el porcentaje máximo de ruido siempre es $\epsilon-$A con $\epsilon = .0$ el método con mayor número de nodos explorados, excepto en *bibtex*. En general, a mayor cantidad de ruido el algoritmo $\epsilon-$A con $\epsilon = .0$ es el que tiene peor rendimiento en cuanto a número de nodos y tiempo computacional.

Con respecto al número de nodos explorados por el algoritmo A* utilizando ambas familias de heurísticos, éste disminuye a medida que d aumenta. De esta manera, el número de nodos expandidos utilizando el heurístico h_l^∞ se mantiene bastante constante a medida que el ruido aumenta en comparación con el uso del resto de heurísticos. De hecho, para el resto, el número de nodos aumenta rápidamente después de cierto porcentaje de ruido, especialmente para conjuntos de datos con más de 14 etiquetas. Comparando el número de nodos explorados por A* utilizando ambas familias de heurísticos h_l^d y h_{ln}^d, se puede ver que para el mismo valor de d exploran una cantidad de nodos muy similar.

En cuanto al tiempo de predicción, en la Tabla 3.11 se puede ver que, entre los métodos que garantizan obtener soluciones óptimas, los heurísticos h_l^1 y h_{nl}^1 y el algoritmo $\epsilon-$A con $\epsilon = .0$ tienen el mejor comportamiento en tiempo computacional. Sin embargo, a partir de las Figuras 3.8 y 3.9 se puede observar que esto solo ocurre cuando no hay presencia de ruido, debido a que los heurísticos h_l^d, h_{nl}^d con $d \geq 2$ tienen en general mejor comportamiento que los heurísticos h_l^1 y h_{nl}^1 y el algoritmo $\epsilon-$A con $\epsilon = 0$ cuando existe ruido. De hecho, el tiempo computacional de los heurísticos h_l^1 y h_{nl}^1 y el algoritmo $\epsilon-$A con $\epsilon = .0$ aumenta considerablemente a medida que aumenta el porcentaje de ruido, aunque es digno de mencionar que el tiempo computacional de los heurísticos h_l^1 y h_{nl}^1 aumenta en menor medida. En realidad, el tiempo computacional de los heuristicos h_l^d, h_{nl}^d con $d \geq 2$ no siempre aumenta directamente proporcional al porcentaje de ruido, sino que se mantiene casi constante, incluso para porcentajes de ruido bajos, mostrando así el potencial de las familias de heurísticos propuestas.

En las gráficas es bastante notable que el tiempo computacional empleado por h_{nl}^d es siempre menor que el tiempo computacional empleado por h_l^d para el mismo nivel de profundidad d. Por lo tanto, a diferencia de h_l^d, el tiempo empleado en calcular h_{nl}^d no es una desventaja, sino todo lo contrario. En general, se puede observar en las gráficas que el tiempo de predicción para h_l^d y para h_{ln}^d disminuye a medida que d aumenta. Por otro lado, en los conjuntos de datos con muchas etiquetas, h_l^∞ se convierte en la mejor opción, pues es el heurístico que menor tiempo computacional consume a mayor presencia de ruido. Sin embargo, en los conjuntos de datos pequeños h_{nl}^3 es el que tiene mejor rendimiento en cuanto a tiempo medio de predicción.

Capítulo 4

Conclusiones y trabajo futuro

4.1. Conclusiones

En este capítulo se presentan las conclusiones obtenidas con la realización del presente trabajo teniendo en cuenta los objetivos de investigación propuestos.

En primera instancia se realizó un análisis exhaustivo de los diferentes métodos de inferencia para PCC, estudiando sus estrategias de búsqueda, sus propiedades y su comportamiento en función de la variación de sus parámetros. De los métodos estudiados solo el algoritmo ϵ-A con $\epsilon = .0$ (o UC) puede garantizar la obtención de soluciones óptimas en términos de *subset 0/1*.

Otros métodos tienden a alcanzar soluciones óptimas cuando se asignan valores adecuados a sus parámetros. Sin embargo, no pueden garantizar la obtención de soluciones óptimas y además requieren un coste computacional mayor, explorando una cantidad elevada de nodos. En esta categoría entra el algoritmo BS que tiende a alcanzar una solución óptima para valores de $b \geq 10$. El algoritmo MC logra en muy pocos casos alcanzar soluciones óptimas incluso a costa de utilizar tamaños de muestra altos. Sin embargo, el algoritmo EMC sí que logra alcanzar soluciones óptimas, aunque a costa de generar un número elevado de muestras. Como era de esperar, el método GS es el algoritmo con peor rendimiento en términos de *subset 0/1 loss*, pues sus resultados son los más pobres. Pese a que algunos métodos lograron superar por pequeños márgenes los resultados óptimos, se podría decir que es algo circunstancial y que no se puede garantizar siempre, ya que simplemente puede estar ligado a cierto grado de imprecisión con la que los clasificadores estiman las probabilidades, debido principalmente a que los modelos se construyen a partir de un conjunto de entrenamiento que puede no reflejar fielmente la distribución subyacente de los datos del problema.

En cuanto al número de nodos explorados y el tiempo computacional existe

una relación directamente proporcional entre ambas magnitudes, de forma que al aumentar la primera, se incrementa la segunda. De esta manera, el método GS es el que menos nodos explora y por ende el más rápido, pero a costa de proporcionar resultados más pobres en términos de *subset 0/1 loss*. Los métodos BS, MC y EMC aumentan considerablemente el número de nodos explorados y el tiempo computacional en un intento por alcanzar una solución óptima, aunque esto no se garantice. En general, el algoritmo $\epsilon-$A con $\epsilon=0$ (o UC) es el que mejor comportamiento presenta, pues siempre garantiza alcanzar una solución óptima, explorando un considerable menor número de nodos y por ende consumiendo menor tiempo computacional que el resto de métodos, cuando logran alcanzar una solución óptima. Todo esto convierte al algoritmo ϵ-A con $\epsilon = 0$ (o UC) en la mejor opción para hacer inferencia en PCC, entre los métodos existentes en la literatura para dicho propósito.

En segunda instancia, esta investigación se centró en diseñar nuevos métodos alternativos para realizar inferencia en PCC en la clasificación multietiqueta, que al igual que el algoritmo $\epsilon-$A con $\epsilon=0$ (o UC) garantizasen siempre alcanzar una solución óptima y con bajo coste computacional. Para ello, se propuso el diseño de heurísticos admisibles sobre el algoritmo de búsqueda A*. Particularmente, se diseñaron dos familias de heurísticos admisibles. La primera familia de heurísticos se limitó a la utilización de clasificadores lineales como métodos base para PCC, con el objetivo de que lograsen ser lo más dominantes posible, y por tanto, explorar la menor cantidad de nodos posibles. Adicionalmente, se incluyó un parámetro d que permite limitar la profundidad de la búsqueda para mantener un equilibrio entre el número de nodos explorados y el tiempo de cálculo del heurístico, pues este último resultó ser muy elevado. Esta familia de heurísticos se denotó como h_l^d, donde el subíndice l hace referencia a que solo es válido para clasificadores lineales y el superíndice d hace referencia a la profundidad del subárbol que se explora para obtener la información heurística. Por otro lado, la segunda familia de heurísticos se centró en romper la limitación de la primera, cuyos heurísticos solo eran aplicables con clasificadores lineales. Esta nueva familia de heurísticos utiliza la estrategia de búsqueda exhaustiva pero con una profundidad limitada, con el fin de que su cómputo sea viable. Esta familia de heurísticos se denotó como h_{nl}^d, donde el subíndice nl indica que puede ser utilizada tanto con clasificadores no lineales y el superíndice d, al igual que en el caso anterior, indica la profundidad de la búsqueda. De nuevo, la inclusión de este parámetro permite conseguir un equilibrio entre el número de nodos explorados y el tiempo computacional de cálculo del heurístico.

En cuanto al tercer objetivo de esta investigación, que consistía en comparar los métodos de inferencia en PCC de la literatura con los propuestos en esta investigación, los experimentos lograron confirmar que efectivamente los heurísticos admisibles propuestos h_l^d y h_{nl}^d garantizan alcanzar una solución óptima en términos de *subset 0/1 loss*, tal como se esperaba al tratarse de heurísticos admisibles que se usan en combinación con el algoritmo de búsqueda A*. Los resultados de los experimentos también demostraron que el rendimiento en

términos de *subset 0/1* puede mejorar cuando se utilizan clasificadores no lineales en lugar de clasificadores lineales como métodos base. En cuanto al número de nodos explorados, se demostró teóricamente que el heurístico h_l^d es más dominante que el heurístico $h=1$ (o equivalentemente el algoritmo ϵ-A con $\epsilon=.0$), explorando, por tanto, igual o menor número de nodos que este último. A su vez, también se demostró teóricamente que el heurístico h_{nl}^d es más dominante que el heurístico h_l^d, explorando, consecuentemente, igual o menor cantidad de nodos que h_l^d, para el mismo valor de d (ambos heurísticos h_l^d y h_{nl}^d son más dominantes a medida que aumenta el valor de d). Sin embargo, en el caso de h_{nl}^d se recomienda utilizar valores menores o iguales a 3, puesto que para valores mayores el heurístico comienza a dejar de ser computacionalmente tratable. A pesar de que los heurísticos h_l^d y h_{nl}^d son más dominantes que el $\epsilon-$A con $\epsilon = 0$, este último logra tener mejor rendimiento general en cuanto a tiempo computacional, consumiendo normalmente menos tiempo que los heurísticos propuestos en este trabajo, debido a que el coste computacional de evaluar los heurísticos no parece compensar la reducción en el número de nodos explorados. Sin embargo, h_{nl}^d compensa en mayor grado el coste del cálculo del heurístico que h_l^d, puesto que para el mismo nivel de profundidad d logra consumir un menor tiempo computacional.

En última instancia y dando cumplimiento al cuarto y último objetivo de esta investigación, se añadió incertidumbre en forma de ruido a los conjuntos de datos de referencia, aumentado la complejidad de alcanzar una solución óptima y se comparó el comportamiento de los métodos que garantizan alcanzar soluciones óptimas, es decir, el algoritmo $\epsilon-$A con $\epsilon=0$ y el algoritmo A* con los heurísticos h_l^d y h_{nl}^d para diferentes valores de d. Los experimentos mostraron el potencial de los heurísticos propuestos en este trabajo de investigación, dado que A* con estos heurísticos tiene mejor comportamiento que el algoritmo ϵ-A con $\epsilon=0$ a medida que aumenta la incertidumbre en los conjuntos de datos. En este sentido, el tiempo computacional de ϵ-A con $\epsilon=0$ aumenta considerablemente a medida que aumenta la incertidumbre, mientras que el tiempo computacional de los heurísticos h_l^∞ y h_{nl}^3 es prácticamente constante (con tendencia a disminuir en algunos casos).

Como conclusión general, se han diseñado dos familias de heurísticos admisibles para el algoritmo A* con el fin de realizar inferencia en PCC que, además de garantizar que alcanzan soluciones óptimas, logran explorar menos nodos que los métodos existentes en la literatura. Si bien el tiempo de cálculo de los heurísticos no logra compensar el reducido número de nodos que explora ante conjuntos de datos donde la búsqueda de soluciones es más dirigida, A* con esta familia de heurísticos sí que demuestra su potencial ante la presencia de mayor incertidumbre en la búsqueda, situación que se suele presentar con frecuencia en problemas reales.

4.2. Trabajo futuro

En esta sección se detallan algunas propuestas de trabajo futuro que darán continuidad a esta investigación. Estas propuestas se han dividido en dos bloques

que corresponden a trabajos a corto o medio plazo y trabajos a largo plazo. En el primer bloque se incluyen trabajos que se están realizando en la actualidad o que por razones de tiempo no alcanzaron aún su fin para formar parte de esta memoria. En el segundo bloque se incluyen otros trabajos que podrían ser desarrollados una vez se concluyan los ya iniciados.

(i) Entre los trabajos que se encuentran en desarrollo y que podrán completarse a corto o medio plazo se encuentran:

   (a) Una propuesta de alternativa al algoritmo ϵ-A denominada *Improved ϵ-Approximate*. Esta propuesta consiste en un algoritmo alternativo basado en el algoritmo ϵ-A, que también garantiza alcanzar soluciones óptimas en términos de *subset 0/1 loss*, pero que explora un menor número de nodos. La idea consiste en establecer una cota para la máxima probabilidad que los nodos de un mismo nivel pueden tener, probabilidad que en cualquier caso será menor o igual que la unidad. Esta cota se puede utilizar para calcular un heurístico, haciendo que el algoritmo encuentre un camino más directo para llegar a la solución óptima. Los experimentos realizados hasta el momento demuestran que este nuevo algoritmo mejora el rendimiento del ϵ-A original tanto en número de nodos explorados como en tiempo computacional.

   (b) Un estudio sobre la dependencia incondicional entre las etiquetas. Estudios previos han demostrado que con la eliminación de ciertas etiquetas que tienen poca correlación se pueden mejorar las predicciones. Sin embargo, otros estudios realizados durante el transcurso de esta investigación demuestran que la eliminación de etiquetas tanto con mayor correlación como con menor correlación podría mejorar la predicción.

   (c) Aplicar y estudiar el uso de la clasificación multietiqueta sobre datos ordinales. Para ello se han desarrollado hasta la fecha algunas implementaciones de algoritmos basados en BR y CC y realizado experimentos preliminares sobre los mismos. Adicionalmente, se pretende desarrollar implementaciones de algoritmos para inferencia en PCC y realizar experimentos que estudien sus comportamientos cuando se aplican sobre datos ordinales.

(ii) A largo plazo y una vez concluidas las investigaciones anteriores, se podrían idear nuevos heurísticos admisibles que mejoren los resultados actuales en términos de coste computacional, debido a que la realización de este trabajo ha demostrado que la búsqueda heurística es una línea prometedora de investigación para la inferencia en PCC.

Capítulo 5

Compendio de publicaciones

Este capítulo contiene el compendio de las publicaciones fruto del presente trabajo de investigación. Estas publicaciones corresponden a tres artículos publicados en revista.

5.1. An overview of inference methods in probabilistic classifier chains for multilabel classification

En esta sección se incluye el artículo de revista:

- D. Mena, E. Montañés, J. R. Quevedo, and J. J. Coz, "An overview of inference methods in probabilistic classifier chains for multilabel classification," Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, vol. 6, no. 6, pp. 215–230, 2016.

Focus Article

An overview of inference methods in probabilistic classifier chains for multilabel classification

Deiner Mena,[1,2] Elena Montañés,[1] José Ramón Quevedo[1] and Juan José del Coz[1]*

This study presents a review of the recent advances in performing inference in probabilistic classifier chains for multilabel classification. The interest of performing such inference arises in an attempt of improving the performance of the approach based on greedy search (the well-known CC method) and simultaneously reducing the computational cost of an exhaustive search (the well-known PCC method). Unlike PCC and as CC, inference techniques do not explore all the possible solutions, but they increase the performance of CC, sometimes reaching the optimal solution in terms of subset 0/1 loss, as PCC does. The ε-approximate algorithm, the method based on a beam search and Monte Carlo sampling are those techniques. An exhaustive set of experiments over a wide range of datasets are performed to analyze not only to which extent these techniques tend to produce optimal solutions, otherwise also to study their computational cost, both in terms of solutions explored and execution time. Only ε-approximate algorithm with ε=.0 theoretically guarantees reaching an optimal solution in terms of subset 0/1 loss. However, the other algorithms provide solutions close to an optimal solution, despite the fact they do not guarantee to reach an optimal solution. The ε-approximate algorithm is the most promising to balance the performance in terms of subset 0/1 loss against the number of solutions explored and execution time. The value of ε determines a degree to which one prefers to guarantee to reach an optimal solution at the expense of increasing the computational cost.

How to cite this article:
WIREs Data Mining Knowl Discov 2016, 6:215–230. doi: 10.1002/widm.1185

INTRODUCTION

Multilabel classification[1] (MLC) is a machine learning problem in which models are able to assign a subset of (class) labels to each instance, unlike conventional (single-class) classification that involves predicting only a single class. MLC problems are ubiquitous and naturally occur, for instance, in assigning keywords to a paper, tags to resources in a social network, objects to images or emotional expressions to human faces.

In general, the problem of multilabel learning comes with two fundamental challenges. The first one bears on the computational complexity of the algorithms. A complex approach might not be applicable in practice when the number of labels is large. Therefore, the scalability of algorithms is a key issue in this field. The second problem is related to the own nature of multilabel data. Not only the number of labels is typically large, otherwise each instance also belongs to a variable-sized subset of labels simultaneously. Moreover, and perhaps even more important, the labels will normally not occur independently of each other; instead, there are statistical dependencies between them. From a learning and prediction point of view, these relationships constitute a

*Correspondence to: juanjo@uniovi.es

[1]Artificial Intelligence Center, University of Oviedo at Gijón, Gijón, Spain

[2]Dept. de Ingeniería en Telecomunicaciones e Informática, Universidad Tecnológica del Chocó, Chocó, Colombia

Conflict of interest: The authors have declared no conflicts of interest for this article.

promising source of information, in addition to the information coming from the mere description of the instances. Thus, it is hardly surprising that research on MLC has very much focused on the design of new methods that are able to detect—and benefit from—interdependencies among labels.

Several approaches have been proposed in the literature to cope with MLC. First, researchers tried to adapt and extend different state-of-the-art binary or multiclass classification algorithms, including methods using decision trees,[2] neural networks,[3] support vector machines,[4] naive Bayes,[5] conditional random fields,[6] and boosting.[7] Secondly, they further analyzed in depth the label dependence and attempted to design new approaches that exploit label correlations.[8] In this regard, two kinds of label dependence have been formally distinguished: conditional dependence[6,9–13] and unconditional dependence.[3,14,15] Also, pairwise relations,[3,4,7,16,17] relations in sets of different sizes,[12,18,19] or relations in the whole set of labels[10,14,15] have also been exploited.

Regarding conditional label dependence, the approach called probabilistic classifier chains (PCC) has aroused great interest among the multilabel community, because it offers the nice property of being able to estimate the conditional joint distribution of the labels. However, the original PCC algorithm[9] suffers from high computational cost, as it performs an exhaustive search (ES) as inference strategy to obtain optimal solutions in terms of a given loss function. Then, several efforts that use different searching and sampling strategies in order to overcome this drawback are being made just now. This includes uniform-cost (UC) search,[20] beam search (BS),[21,22] and Monte Carlo sampling (MS).[20,23,24] All of these algorithms successfully estimate the optimal solution reached by the original PCC,[9] at the same time that they reduce the computational cost in terms of both the number of candidate solutions explored and execution time. This article studies in depth the behavior and the properties of all these algorithms, comparing their strategies and establishing their differences and similarities, paying special attention to the meaning of their parameters and the effect of the different values they can take. The methods are experimentally compared over a wide range of multilabel datasets, concluding that even those that do not theoretically guarantee obtaining optimal solutions also reach good performance. However, the ϵ-approximate algorithm shows to be a promising alternative even for values of ϵ that do not guarantee reaching optimal solutions. For this algorithm, it happens that renouncing to reach optimal solutions leads to reduce the computational cost in terms of candidate solutions explored and execution time and viceversa.

The rest of the article is organized as follows. *PCCs in MLC* section formally describes multilabel framework and the principles of PCC. *Inference in PCCs* section discusses the properties and behavior of the different existing approaches for inference in PCC. Exhaustive experiments are shown and discussed in *Experiments* section. Finally, *Conclusions* section exposes some conclusions and includes new directions of future work.

PCCs IN MLC

This section formally describes the MLC task. Let $\mathcal{L} = \{\ell_1, \ell_2, \ldots, \ell_m\}$ be a finite and nonempty set of m labels and $S = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ a training set independently and randomly drawn according to an unknown probability distribution $\mathbf{P}(\mathbf{X}, \mathbf{Y})$ on $\mathcal{X} \times \mathcal{Y}$, where $\mathcal{X}$ and $\mathcal{Y}$ are the input and the output space, respectively. The former is the space of the instance description, whereas the latter is given by the power set $\mathcal{P}(\mathcal{L})$ of $\mathcal{L}$. To ease notation, we define y_i as a binary vector $y_i = (y_{i,1}, y_{i,2}, \ldots, y_{i,m})$ in which $y_{i,j} = 1$ indicates the presence (relevance) and $y_{i,j} = 0$ the absence (irrelevance) of ℓ_j in the labeling of x_i. Hence, y_i is the observation of a corresponding random vector $\mathbf{Y} = (\mathbf{Y_1}, \mathbf{Y_2}, \ldots, \mathbf{Y_m})$. Using this convention, the output space can also be defined as $\mathcal{Y} = \{0,1\}^m$. The goal in MLC is to induce from S a hypothesis $h : \mathcal{X} \to \mathcal{Y}$ that minimizes the risk in terms of certain loss function $L(\cdot)$ when it provides a vector of relevant labels $y = h(x) = (h_1(x), h_2(x), \ldots, h_m(x))$ for unlabeled query instances x. This risk can be defined as the expected loss over the joint distribution $\mathbf{P}(\mathbf{X}, \mathbf{Y})$, i.e.,

$$R_L(h) = \mathbb{E}_{\mathbf{X},\mathbf{Y}} L(\mathbf{Y}, h(\mathbf{X})), \tag{1}$$

therefore, denoting by $\mathbf{P}(y \mid x)$ the conditional distribution $\mathbf{Y} = y$ given $\mathbf{X} = x$, the so-called risk minimizer h^* can be expressed by

$$h^*(x) = \operatorname*{argmin}_h \sum_{y \in \mathcal{Y}} \mathbf{P}(y \mid x) L(y, h(x)). \tag{2}$$

Let us comment that the conditional distribution $\mathbf{P}(y \mid x)$ presents different properties which are crucial for optimizing different loss functions. At this respect, the strategy followed by a certain MLC algorithm for modeling label dependence determines the optimized loss function. Unfortunately, it is quite complex and confusing to find out the loss function optimized by several algorithms.

In regard to the loss functions, several performance measures have been taken for evaluating MLC. The most specific ones are the subset 0/1 loss and the Hamming loss, but there exist other measures that have been taken from other research fields, such as the F_1 measure or the *Jaccard* index. Here, we will focus on just the subset 0/1 loss because it is the measure that PCC is able to optimize. The subset 0/1 loss looks whether the predicted and relevant label subsets are equal or not. It is defined by

$$L_{S_{0/1}}(y, h(x)) = [[y \neq h(x)]], \tag{3}$$

in which the expression $[[p]]$ evaluates to 1 when the predicate, p, is true and to 0 otherwise. Notice that it suffices taking the mode of the entire joint conditional distribution for optimizing this loss function. Formally, the risk minimizer adopts the simplified following form

$$h^*(x) = \underset{y \in \mathcal{Y}}{\operatorname{argmax}} \mathbf{P}(y \mid x). \tag{4}$$

Among the methods that cope with MLC, the simplest straightforward approach is Binary Relevance (BR).[25] This method assumes independence among the labels and provides optimal prediction for subset 0/1 loss when such independence actually exists. Hence, it estimates $\mathbf{P}(y_j \mid x)$ for label ℓ_j just considering the description of the instances and using a probabilistic model $h_j : \mathcal{X} \to [0,1]$. The joint distribution of labels $\mathbf{Y} = (\mathbf{Y_1}, \mathbf{Y_2}, \ldots, \mathbf{Y_m})$ is computed using the product rule of probability assuming independence among labels, i.e., $\mathbf{P}(y \mid x) = \prod_{j=1}^{m} \mathbf{P}(y_j \mid x)$.

Concerning methods that take into account the possible dependence among labels, let us focus on those based on learning a chain of classifiers (as CC[12,26] or PCC[9]). First, these methods define an order of the label set. Then, following such order, they train a probabilistic binary classifier for each label ℓ_j to estimate $\mathbf{P}(y_j \mid x, y_1, \ldots, y_{j-1})$. Hence, the probabilistic model obtained for predicting label ℓ_j, denoted by h_j, is of the form

$$h_j : \mathcal{X} \times \{0,1\}^{j-1} \to [0,1]. \tag{5}$$

The training data for this classifier are the set $S_j = \{(\overline{x}_1, y_{1,j}), \ldots, (\overline{x}_n, y_{n,j})\}$ where $\overline{x}_i = (x, y_{i,1}, \ldots, y_{i,j-1})$, i.e., the features are x_i supplemented by the relevance of the labels $\ell_1, \ldots, \ell_{j-1}$ preceding ℓ_j in the chain and the category is the relevance of the label ℓ_j.

In the testing stage of the methods based on learning a chain of classifiers, the goal is to perform inference for each instance, which consists of estimating the risk minimizer for a given loss function over the estimated entire joint conditional distribution. The idea revolves around repeatedly applying the general product rule of probability to the joint distribution of the labels $\mathbf{Y} = (\mathbf{Y_1}, \mathbf{Y_2}, \ldots, \mathbf{Y_m})$, i.e., computing

$$\mathbf{P}(y \mid x) = \prod_{j=1}^{m} \mathbf{P}(y_j \mid x, y_1, \ldots, y_{j-1}) \tag{6}$$

Before analyzing this issue in the following section, notice that from a theoretical point of view, this expression holds for any order considered for the labels. But, in practice, these methods are label order dependent for several reasons. On the one hand, it is not possible to assure that the models obtained in the training stage perfectly estimate the joint conditional probability $P(y \mid x)$. On the other hand, predicted values instead of true values are successively taken in the testing stage. This is more serious if the highest errors occur at the beginning of the chain, as error predictions are successively propagated.[11,27,28] In any case, in this article we assume an order of the labels in the chain in order to better analyze all these methods in insolation without taking into account other factors. Hence, we do not include any study about which order can be the best.

Finally, let us now consider the task of performing different inference procedures as different manners of exploring a probability binary tree in order to facilitate the explanation and analysis of the inference approaches in next section. In such tree, a node of the $(j-1)$-th level is labeled by $(v_1, \ldots, v_{j-1})$ with $v_i \in \{0, 1\}$ for $i = 1, \ldots, j-1$. This node has two children respectively labeled as $(v_1, \ldots, v_{j-1}, 0)$ and $(v_1, \ldots, v_{j-1}, 1)$ with marginal joint conditional probability $\mathbf{P}(y_1 = v_1, \ldots, y_{j-1} = v_{j-1}, y_j = 0 \mid x)$ and $\mathbf{P}(y_1 = v_1, \ldots, y_{j-1} = v_{j-1}, y_j = 1 \mid x)$, respectively. The weights of the edges between both parent and children are respectively $\mathbf{P}(y_j = 0 \mid x, y_1 = v_1, \ldots, y_{j-1} = v_{j-1})$ and $\mathbf{P}(y_j = 1 \mid x, y_1 = v_1, \ldots, y_{j-1} = v_{j-1})$, which are respectively estimated by $1 - h_j(x, v_1, \ldots, v_{j-1})$ and $h_j(x, v_1, \ldots, v_{j-1})$. The marginal joint conditional probability of the children is computed using the product rule of probability. Then, $\mathbf{P}(y_1 = v_1, \ldots, y_{j-1} = v_{j-1}, y_j = 0 \mid x) = \mathbf{P}(y_j = 0 \mid x, y_1 = v_1, \ldots, y_{j-1} = v_{j-1}) \cdot \mathbf{P}(y_1 = v_1, \ldots, y_{j-1} = v_{j-1} \mid x)$ and $\mathbf{P}(y_1 = v_1, \ldots, y_{j-1} = v_{j-1}, y_j = 1 \mid x) = \mathbf{P}(y_j = 1 \mid x, y_1 = v_1, \ldots, y_{j-1} = v_{j-1}) \cdot$

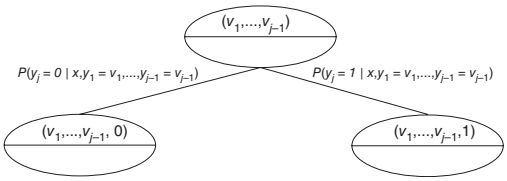


FIGURE 1 | A generic node and its children of the probability binary tree. The top part of each node contains the combination of labels and the bottom part includes the joint probability of such combination. The edges are labeled with the conditional probability.

$\mathbf{P}(y_1 = v_1, \ldots, y_{j-1} = v_{j-1} \mid x)$. The root node is labeled by the empty set. Figure 1 illustrates it.

INFERENCE IN PCCs

Several approaches have been proposed for inference in PCC. The method firstly proposed was that one based on greedy search (GS), being the integral part of the original CC method.[26] Its successor is the ES, called the PCC method.[9] The ε-approximate (ε-A) algorithm[20] is an UC search algorithm that can output optimal predictions in terms of subset 0/1 loss, reducing significantly the computational cost of ES. A more recent approach based on BS[21,22] presents good behavior both in terms of performance and computational cost. Finally, MS[20] is an appealing and simpler alternative[20,23] to overcome the high computational cost of ES. Before proceeding to cope with the specificity of all these inference methods, notice that the training phase is common to all of them, thus, the models h_j induced by the binary classifiers will be the same. So, in what follows we will focus just on the testing stage for a given unseen example x.

Greedy Search

At the testing stage, the GS strategy, originally called CC,[12,26] provides an output $\hat{y} = (\hat{y}_1, \ldots, \hat{y}_m)$ for a new unlabeled instance x by successively querying each classifier h_j that estimates the conditional probability $\mathbf{P}(y_j \mid x, y_1, \ldots, y_{j-1})$. This means to explore just one node in each level j. Given that only the two children of the explored node in level j are taken, their marginal joint conditional probabilities only differ in the marginal conditional probability term, $\mathbf{P}(y_j \mid x, y_1, \ldots, y_{j-1})$, because both children have the same parent. Thus, the path selected is the one of the child with the highest marginal conditional probability $\mathbf{P}(y_j \mid x, y_1, \ldots, y_{j-1})$. Notice that when $\hat{y}_j$ is estimated, h_j is applied, which needs both the feature vector x and the values for the labels from ℓ_1 to ℓ_{j-1}. In this regard, let us remind that $y_1, \ldots, y_{j-1}$ are not available in the testing stage, hence, the respective predictions $\hat{y}_1 = h_1(x)$, $\hat{y}_2 = h_2(x, \hat{y}_1)$, $\ldots$, $\hat{y}_{j-1} = h_{j-1}(x, \hat{y}_1, \hat{y}_2, \ldots, \hat{y}_{j-2})$ are taken instead. Thus, the prediction for an instance x is of the form $\hat{y} = (h_1(x), h_2(x, h_1(x)), h_3(x, h_1(x), h_2(x, h_1(x))), \ldots)$.

Figure 2(a) shows the path followed by an instance using this strategy. In this example, only the right node is explored at each level. The optimal solution is not reached, because the optimal solution is that which ends in the sixth leaf, whereas the method falls in the last leaf. The pseudocode of the GS method can be seen in Algorithm 1 (left) in which $v_R = \emptyset$ is the root node of the probability tree, and $(\Pi(lc(v)), \Pi(rc(v)))$ are the marginal conditional probabilities of the left and right child of a node v, respectively. For instance, if v is a node of j-th level, $\Pi(lc(v)) = \mathbf{P}(y_j = 0 \mid x, y_1 = v_1, \ldots, y_{j-1} = v_{j-1})$ and $\Pi(rc(v)) = \mathbf{P}(y_j = 1 \mid x, y_1 = v_1, \ldots, y_{j-1} = v_{j-1})$.

Algorithm 1 Greedy search (left) and Exhaustive search (right)

```
1: function GreedySearch
   Input: CC Model {h_j : j = 1,...,m}
   Output: v = (v_1,...,v_m) with v_j ∈ {0,1}
2:     v ← v_R    // the root of the probability tree
3:     while v is not a leaf do
4:         lc(v), rc(v) ← left and right child of v
5:         if Π(lc(v)) ≥ Π(rc(v)) then
6:             v ← lc(v)
7:         else
8:             v ← rc(v)
9:     return v
```

```
1: function ExhaustiveSearch
   Input: CC Model {h_j : j = 1,...,m}
   Output: v = (v_1,...,v_m) with v_j ∈ {0,1}
2:     Q ← {(v_R, Π̄(v_R) = 1)}    // {(root node, Π̄(root node))}
3:     maxP ← 0
4:     while Q ≠ ∅ do
5:         w ← pop an element in Q
6:         lc(w), rc(w) ← left and right child of w
7:         compute Π̄(lc(w)), Π̄(rc(w)) recursively from Π̄(w)
8:         if lc(w) and rc(w) are not leaves then
9:             Q ← Q ∪ {(lc(w), Π̄(lc(w))), (rc(w), Π̄(rc(w)))}
10:        else if max(Π̄(lc(w)), Π̄(rc(w))) > maxP then
11:            v ← argmax_{w'∈{lc(w),rc(w)}} (Π̄(w'))
12:            maxP ← max(Π̄(lc(w)), Π̄(rc(w)))
13:    return v
```

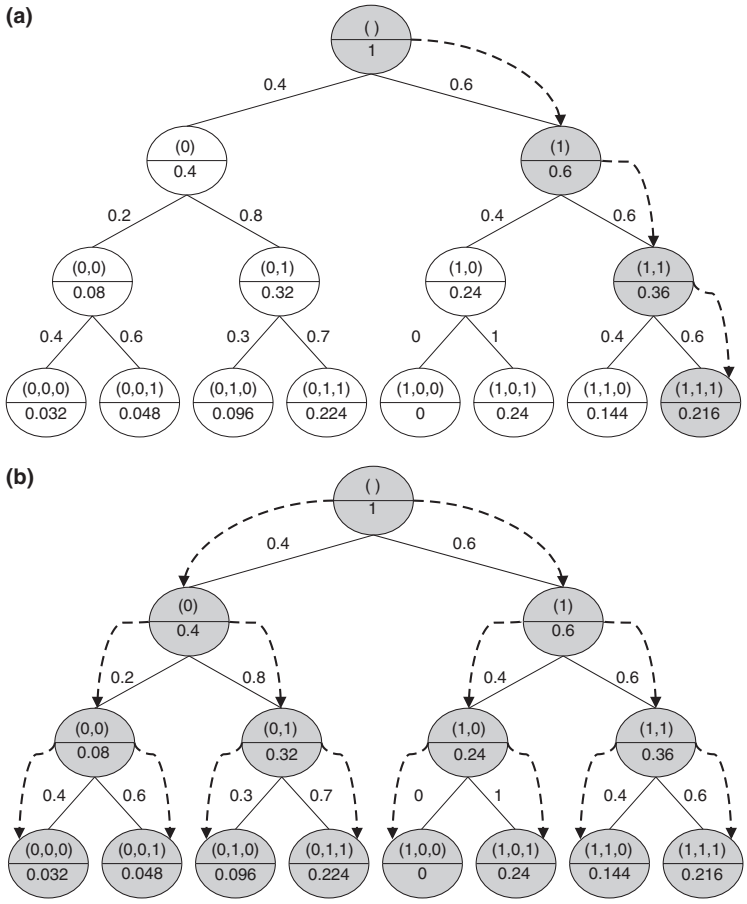



FIGURE 2 | An example of paths followed by an instance using Greedy Search (CC). The dotted arrows show the path followed by the algorithm. (a) Greedy Search (CC) and (b) Exhaustive Search (PCC).

Concerning the optimization of subset 0/1 loss, a rigorous analysis[20] establishes bounds for the performance of GS. For this purpose, the authors define the regret r of a classifier h for a given loss $L(\cdot,\cdot)$ as the difference between the risk of that classifier and the Bayes-optimal classifier h^B, i.e.,

$$r_L(h) = \mathbb{E}_{\mathbf{X},\mathbf{Y}} L(\mathbf{Y}, h(\mathbf{X})) - \mathbb{E}_{\mathbf{X},\mathbf{Y}} L\left(\mathbf{Y}, h^B(\mathbf{X})\right) \qquad (7)$$

In case of the subset 0/1 loss, one can just analyze the expectation over $\mathbf{Y}$ given x, because this loss is decomposable with regard to individual instances. Hence, the regret becomes

$$r_L(h) = \mathbb{E}_{\mathbf{Y}} L(\mathbf{Y}, h(x)) - \mathbb{E}_{\mathbf{Y}} L\left(\mathbf{Y}, h^B(x)\right) \qquad (8)$$

Then, assuming that perfect estimate of the joint conditional probability $\mathbf{P}(y \mid x)$ has been obtained, they establish an upper bound equal to $2^{-1} - 2^{-m}$ for the regret of the classifier h induced by the GS method for the subset 0/1 loss. This bound shows that this method offers quite poor performance for this loss, in other words, the method does not manage to provide a good estimation of the risk minimizer for this loss. However, the work concludes stating that GS tries to optimize the subset 0/1 loss rather than, e.g., the Hamming loss, because it is more suitable for estimating the mode of the joint conditional

distribution rather than the mode of the marginal conditional distribution.

Exhaustive Search

Unlike GS that explores only one label combination, ES analyzes all possible paths in the tree, estimating the entire joint conditional distribution $\mathbf{P}(y \mid x)$ for a new instance x. Hence, it provides Bayes optimal inference. Then, for each $h(x)$, it computes $\mathbf{P}(y \mid x)$ and $L(y, h(x))$ for all combination of labels $y = (y_1, y_2, \ldots, y_m)$ and outputs the combination $\hat{y} = (\hat{y}_1, \ldots, \hat{y}_m) = h^*(x)$ with minimum risk for the given loss $L(\cdot,\cdot)$. By doing so, it generally improves in terms of performance, as it perfectly estimates the risk minimizer, albeit at the cost of a higher computational cost, as it comes down to summing over an exponential (2^m) number of label combinations for each $h(x)$.

Figure 2(b) illustrates this approach; all paths are explored and the optimal solution is always reached. Algorithm 1 (right) shows a possible pseudocode for this method. In this case, $\overline{\Pi}(v) = P(y_1 = v_1, \ldots, y_j = v_j \mid x)$ is recursively obtained by $\overline{\Pi}(v) = \Pi(v) \cdot \overline{\Pi}(pa(v))$ in which v is a node of the j-th level and $pa(v)$ denotes the parent of node v.

ϵ-Approximate Algorithm

The ϵ-approximate (ϵ-A) algorithm[20] arises as an alternative to the high computational cost of ES and to the poor performance of CC. In terms of the probability tree defined above, it expands only the nodes whose marginal joint conditional probability exceeds the threshold $\epsilon = 2^{-k}$ with $1 \le k \le m$. This marginal joint conditional probability for a node in level j for an unlabeled x is

$$\mathbf{P}(y_1, \ldots, y_j \mid x) = \prod_{i=1}^{j} \mathbf{P}(y_i \mid x, y_1, \ldots, y_{i-1}), \qquad (9)$$

where $\mathbf{P}(y_i, \mid x, y_1, \ldots, y_{i-1})$ is estimated by $h_i(x, y_1, \ldots, y_{i-1})$.

The nodes are expanded in the order established by this probability, calculating the marginal joint conditional probability for their children. So, the algorithm does not follow a specific path, otherwise it changes from one path to another depending on the marginal joint conditional probabilities. At the end, two situations can be found: (1) the node expanded is a leaf or (2) there are not more nodes that exceed the threshold. If the former situation occurs, the prediction for the unlabeled instance x will be $\hat{y} = (\hat{y}_1, \ldots, \hat{y}_m)$ corresponding to the combination of the leaf reached (see Figure 3(a)). Conversely, if situation (2) takes place, then GS is applied to the nodes whose children do not exceed the threshold, and the prediction $y = (\hat{y}_1, \ldots, \hat{y}_m)$ for the unlabeled instance x in this case will be that with highest entire joint conditional probability $\mathbf{P}(y_1, \ldots, y_m \mid x)$ (see Figure 3(b) and (c)). Algorithm 2 entails the pseudocode of this method.

Algorithm 2 Pseudocode of the $\epsilon-$approximate algorithm

```
function ε-APPROXIMATE
Input: CC Model {h_j : j = 1, ..., m}, and ε ≥ 0
Output: v = (v_1, ..., v_m) with v_j ∈ {0, 1}
    an ordered list Q ← {(v_R, Π̄(v_R) = 1)}    // {(root node, Π̄(root node))}
    an ordered list K ← ∅    // list of non-survived parents
    repeat
        v ← pop the first element in Q
        if v is not a leaf then
            lc(v), rc(v) ← left and right child of v
            compute Π̄(lc(v)) and Π̄(rc(v)) recursively from Π̄(v)
            if Π̄(lc(v)) ≥ ε then
                insert (lc(v), Π̄(lc(v))) in list Q sorted according to Π̄
            if Π̄(rc(v)) ≥ ε then
                insert (rc(v), Π̄(rc(v))) in list Q sorted according to Π̄
            if lc(v) and rc(v) are not inserted in the list Q then
                insert (v, Π̄(v)) in list K sorted according to Π̄
    until v is a leaf or Q = ∅
    if v is not a leaf then
        ε ← 0
        while K ≠ ∅ do
            w ← pop first element in K and apply GreedySearch to obtain Π̄(w)
            if Π̄(w) ≥ ε then
                v ← w and ε ← Π̄(w)
    return v
```

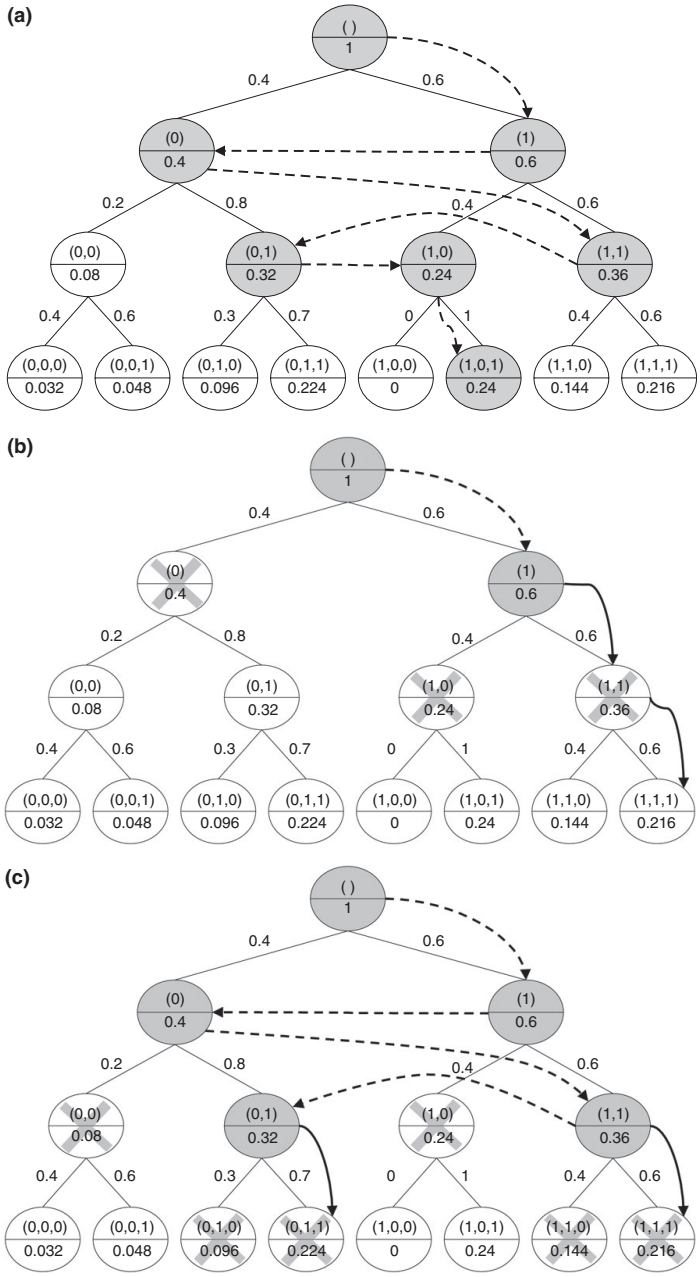



FIGURE 3 | Several examples of the paths followed by ε-A algorithm for different values of ε. The nodes with a cross are those that have a marginal joint conditional probability lower than ε and, hence, they are not explored anymore. The dotted arrows show the path followed by the algorithm. The solid arrows indicate the path followed by the algorithm when the marginal joint conditional probability does not exceed the value of ε, and, hence a GS is applied to this node from here to the bottom of the tree. (a) ε-A algorithm with ε=.0($k = m$), (b) ε-A algorithm with ε=.5 ($k = 1$), (c) ε-A algorithm with ε=.25($k = 2$).

The parameter ε plays an important role in the algorithm. The particular case of ε=.0 (or any value in the interval $[0, 2^{-m}]$, i.e.,, $k = m$) has special interest, because the algorithm performs a UC search that always finds the optimal solution. This is so because the marginal joint conditional probabilities for at least one leaf is guaranteed to be higher than ε, as any probability is positive (or null in the worst case). Even more, this leaf is an optimal solution, because the marginal joint conditional probabilities dismiss as it goes down on the tree. Figure 3a illustrates this situation. In this case, all nodes are candidates to be explored, because all of them will have a marginal joint conditional probability greater than ε. First, the right node of the first level is explored. Their children have a marginal joint conditional probability (0.24 and 0.36, respectively) lower than their uncle (0.4), so their uncle is explored before them. After that, the child positioned closer to the right of the first node is explored because it has the highest marginal joint conditional probability (0.36) among their brother and cousins. The following node to explore is its right cousin with a probability of 0.32, which is higher than their children (0.144 and 0.216). Then, its brother (0.24) is explored and finally its right nephew 0.24, which is already a leaf and as expected the leaf of the optimal solution.

Conversely, the method is looking to GS as ε grows, being the GS in case of ε=.5 (or equivalently $\varepsilon = 2^{-1}$, i.e.,, $k = 1$). This is so because in this case two situations are possible: (1) only one node has a marginal joint conditional probability greater than ε, in whose case the algorithm follows one path, or (2) none node has a marginal joint conditional probability greater than ε, in whose case a GS is applied from here to the bottom of the tree. Figure 3 (b) shows an example of this particular case. In the first level, only the right node has a marginal joint conditional probability that exceeds the value of ε. However, in the second level, none marginal joint conditional probability of the two nodes is greater than ε. In both cases, a step of GS algorithm is applied. Besides, the path followed leads to a non optimal solution.

Notice that this method provides an optimal prediction if the entire joint conditional probability of the corresponding label combination is greater than ε. The interpretation of the method for a generic value of $\varepsilon = 2^{-k}$ is that the method guarantees to reach a partial optimal solution at least until the k-th level on the tree. Even more, the solution remains optimal in levels below the k-th level if the highest marginal joint conditional probability remains higher than 2^{-k} in these levels. As a particular case, if this situation remains until reaching a leaf, then the algorithm obtains an optimal solution. Also notice that the partial combination of labels which is optimal at least until k-th level or which can be optimal until levels below the k-th level can be different from the one of the global optimal solution until such levels, as the global optimal solution also depends on what happens hereinafter. At this respect and from $(k + 1)$-th level, if the joint conditional probability of the optimal solution is greater than ε, then none node is discarded and hence this global optimal solution is reached. Conversely, if the entire joint conditional probability of the optimal solution is lower than ε, then some nodes will be discarded because their marginal joint conditional probability will fall below ε, and hence, it is guarantee that one of these nodes would have lead to the optimal solution. However, if this situation takes place, i.e., if a node is discarded, then the algorithm never reaches a leaf, and hence, GS is applied starting at each discarded node. Therefore, the optimal solution is reached if GS follows the optimal path, which is not guaranteed. Figure 3 (c) shows the particular case of ε=.25. In the figure, some nodes do not exceed the constraint of having their marginal joint conditional probability greater than ε. Even more, none leaf is reached without applying the GS strategy and the optimal solution is not reached.

Consequently, this algorithm estimates the risk minimizer for the subset 0/1 loss a greater or lesser extent depending on the value of ε. Moreover, a theoretical analysis of this estimation[20] allows to bound its goodness as a function of the number of iterations, which in turn depends on ε. Particularly, and in the same direction followed for the GS approach, this analysis establishes that this algorithm needs less than $\mathbf{O}(m2^k)$ iterations to find a prediction that allows to upper bound the regret $r_L(h)$ of the classifier h by $2^{-k} - 2^{-m}$ for the subset 0/1 loss and $k \leq m$, again under the assumption that a perfect estimate of the joint conditional probability $P(y \mid x)$ is obtained. As a consequence, if the probability distribution for which the joint mode has a probability mass bigger than 2^{-k}, then, the algorithm needs less than $m2^k$ iterations to find a prediction that corresponds to this mode. Let notice that the particular case of ε=.0 (or $k = m$) makes the bound becomes 0.

Beam Search

Beam Search (BS)[21,22] also explores more than one path in the probabilistic tree. This method includes a parameter b called beam width that limits the number of combinations of labels explored. The idea is to

explore b possible candidate sets of labels at each level of the tree. Hence, a certain number of the top levels are exhaustively explored depending on the value of b, particularly a total of $k^* - 1$ levels, being k^* the lowest integer such that $b < 2^{k^*}$. Then, only b possibilities are explored for each of the remaining levels. The combinations explored from the level k^* to the bottom are those with the highest marginal joint conditional probability seen thus far. This marginal joint conditional probability for a node of level j for an unlabeled instance x is the same as for the ε-A algorithm. Hence, such probability is given by Eq. (9). At the end, the algorithm outputs $\hat{y} = (\hat{y}_1, \ldots, \hat{y}_m)$ with the highest entire joint conditional probability $\mathbf{P}(y_1, \ldots, y_m | x)$.

BS differs from GS in that (1) BS explores more than one combination and also in (2) the probability taken for deciding a path to follow in the tree. Concerning (2), both take the marginal joint conditional probability $\mathbf{P}(y_1, \ldots, y_j | x)$, but in the case of GS, that is equivalent to take the marginal conditional probability $\mathbf{P}(y_j, | x, y_1, \ldots, y_{j-1})$ because the two nodes explored in each level of the tree have the same parent as explained before. However, in case of BS, the b nodes explored do not have to have the same parent (except for the root node) even in the case of $b = 2$.

In the case of $b = 1$, BS expands just one node in each level, that with highest marginal joint conditional probability that coincides with the highest marginal conditional probability, so BS when $b = 1$ follows just one path that coincides with the one followed by GS. Also, if $b = 2^m$, BS performs an ES. Hence, BS encapsules both GS and ES respectively by considering $b = 1$ and $b = 2^m$. This makes it possible to control the trade-off between computational cost and performance of the method by tuning b between 1 and 2^m. The number of nodes expanded by BS is bounded by $\mathbf{O}(bm)$.

As final remark, the fact that BS considers marginal joint conditional probabilities makes the method tend to estimate the risk minimizer for the subset 0/1 loss. In any case, it would be possible to include other loss functions into the search algorithm. At this respect, the authors who proposed BS for inference in PCC in MLC[21,22] do not include any theoretical analysis about the goodness of the estimation of the risk minimizer. However, they empirically show that taking certain values of b ($b < 15$), the risk minimizer provided by the method converges to the one obtained using ES.

Figure 4 shows an example of the paths explored by the BS algorithm when $b = 2$. Hence, all nodes of the first level are explored. From there, just two nodes are explored, those with highest marginal joint conditional probability among the children whose parents have been previously explored. At this respect, notice that the node with the optimal solution is not explored as its parent has not been previously explored. This example confirms that BS cannot guarantee to reach optimal solutions unless $b = 2^m$. The pseudocode of the algorithm is detailed in Algorithm 3.

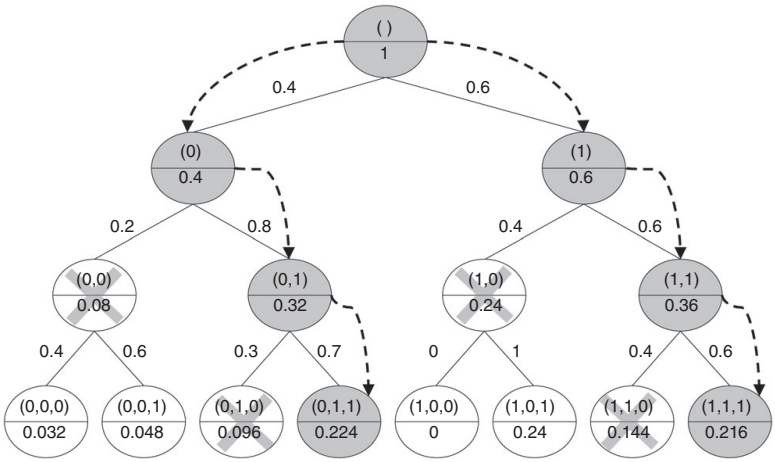


FIGURE 4 | An example of paths followed by an instance using Beam Search (BS) with $b = 2$. The cross out nodes with a cross mean that this node has not any of the highest marginal joint conditional probabilities and, hence, it is not explored anymore. The dotted arrows show the path followed by the algorithm.

Algorithm 3 Beam Search

function BEAMSEARCH
Input: CC Model $\{h_j : j = 1, ..., m\}$, and $b \geq 1$ // b is the beam width
Output: $v = (v_1, ..., v_m)$ with $v_j \in \{0, 1\}$
  $B^{(0)} \leftarrow \{(v_R, \overline{\Pi}(v_R) = 1)\}$ // {(root node, $\overline{\Pi}$(root node))}
  for $j = 1, \ldots, m$ **do**
    $B^{(j)} \leftarrow \emptyset$
    for $w \in B^{(j-1)}$ **do**
      $lc(w)$, $rc(w) \leftarrow$ left and right child of w
      compute $\overline{\Pi}(lc(w))$ and $\overline{\Pi}(rc(w))$ recursively from $\overline{\Pi}(w)$
      if $\overline{\Pi}(lc(w)) > \overline{\Pi}(last(B^j))$ **then**
        insert $(lc(w), \overline{\Pi}(lc(w)))$ in $B^{(j)}$ sorted according to $\overline{\Pi}$ and $B^{(j)} \leftarrow Top_b(B^{(j)})$
      if $\overline{\Pi}(rc(w)) > \overline{\Pi}(last(B^j))$ **then**
        insert $(rc(w), \overline{\Pi}(rc(w)))$ in $B^{(j)}$ sorted according to $\overline{\Pi}$ and $B^{(j)} \leftarrow Top_b(B^{(j)})$
  return $v \leftarrow Top(B^{(m)})$

Monte Carlo Sampling

Monte Carlo sampling (MS) is a technique based on repeating random sampling in order to obtain the distribution of an unknown probabilistic distribution. There are several ways of implementing a Monte Carlo method, but they tend to follow a particular pattern: (1) they define a domain of possible values, (2) they generate values randomly from a probability distribution over the domain, (3) they perform a deterministic computation on the values, and finally, (4) they aggregate the results.

Regarding PCC for MLC, two different Monte Carlo algorithms were recently proposed.[20,23] Both use the domain of the 0/1 vectors of dimension m. In both approaches, the random values are drawn using each classifier h_j, which estimates the probability $\mathbf{P}(y_j | x, y_1, \ldots, y_{j-1})$ of being the label ℓ_j relevant for a new unlabeled instance x. Hence, both algorithms take the previously obtained $\hat{y}_1^{(i)}, \ldots, \hat{y}_{j-1}^{(i)}$ in the chain, for predicting $\hat{y}_j^{(i)}$ in iteration i. Hence, the conditional probability $\mathbf{P}\left(y_j | x, \hat{y}_1^{(i)}, \ldots, \hat{y}_{j-1}^{(i)}\right)$ estimated by $h_j\left(x, \hat{y}_1^{(i)}, \ldots, \hat{y}_{j-1}^{(i)}\right)$ is calculated when the random process takes place for the label ℓ_j. The difference between Monte Carlo approaches and GS, e.g., is that, in the latter the prediction $\hat{y}_j$ for ℓ_j is directly obtained from the evaluation of h_j, whereas in the former such prediction is obtained after a random process drawn using the distribution induced by h_j. This difference makes it possible to repeat the process a certain number of times, producing different predictions, although some of them may be repeated several times. Algorithm 4 shows the pseudocode of both approaches. The key element is the process to draw observations of the conditional distribution (line 4 in the pseudocode). Figure 5 depicts an example of one observation. In each node v, a random number is generated to decide whether the next label is relevant or not. If the random number is lower or equal to $\Pi(lc(v))$ then the left child is selected and the next label is irrelevant; otherwise the label is relevant. The process moves down in the tree according to this rule until a leaf is reached, obtaining a new observation. Notice that this implies that the number of nodes expanded by Monte Carlo approaches is always mq, being q the number of observations of the sample.

Algorithm 4 Pseudocode of Monte Carlo Sampling (left) and Efficient Monte Carlo Sampling (right)

function MONTECARLOSAMPLING
Input: CC Model $\{h_j : j = 1, ..., m\}$, and q // q sample size
Output: $v = (v_1, ..., v_m)$ with $v_j \in \{0, 1\}$
  $Q = \emptyset$
  for $t = 1, \ldots, q$ **do**
    draw w according to Π
    insert w in Q
  return $v \leftarrow$ mode of Q

function EFFICIENTMONTECARLOSAMPLING
Input: CC Model $\{h_j : j = 1, ..., m\}$, and q // q sample size
Output: $v = (v_1, ..., v_m)$ with $v_j \in \{0, 1\}$
  $\overline{\Pi}(u_0) \leftarrow 0$
  for $t = 1, \ldots, q$ **do**
    draw w according to Π
    if $\overline{\Pi}(w) > \overline{\Pi}(u_{t-1})$ **then**
      $u_t \leftarrow w$
    else
      $u_t \leftarrow u_{t-1}$
  return $v \leftarrow u_q$

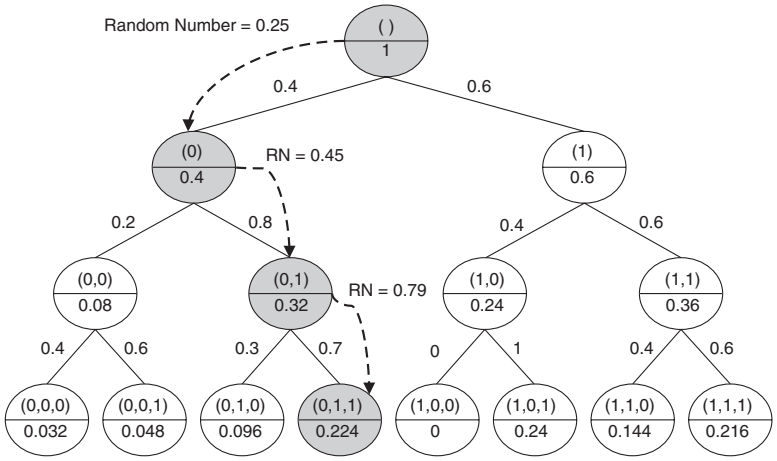


FIGURE 5 | An example of one observation drawn using Monte Carlo sampling. In each node v, a biased coin is flipped to decide whether the label is relevant or not. The probability of tails and heads are given, respectively, by the weights $\pi(lc(v))$ and $\pi(rc(v))$ of the left and right child of a node v. If the random number is lower or equal than $\pi(lc(v))$ then the left child is selected and the label is irrelevant; the right child is selected otherwise. One observation of the conditional distribution is obtained when a leaf is reached.

Finally, different aggregation approaches of the predictions obtained through the iteration procedure can be performed depending on the target loss to be optimized. The risk minimizer in this case is estimated over the subset of $\mathcal{Y}$ formed by the predictions obtained in the iterations instead of over the whole $\mathcal{Y}$. Hence, the estimation can be poorer than in case of ES, and, hence the optimal solution is not guarantee to be reached. However, the fact of using the distribution induced by h_j in the random process guarantees that the aggregated final prediction $\hat{y} = (\hat{y}_1, \ldots, \hat{y}_m)$ of these Monte Carlo methods converges to the risk minimizer when the sample drawn is large enough and when an adequate aggregation procedure according to the loss to be minimized is taken. It is in this point where the two Monte Carlo approaches differ. The first one proposed (we will refer to it as MC)[20] takes the most frequent combination of $\hat{y} = (\hat{y}_1, \ldots, \hat{y}_m)$, i.e., the mode. This approach not only has been used for performing inference in classifier chains, otherwise it was also used for performing inference in classifier trellisers.[29] Conversely, the most recent proposal (we will refer to it as EMC, efficient MC)[23] takes the combination with the highest joint conditional probability. Their authors tag this approach as efficient, because this aggregation procedure allows the method to converge faster. Besides, it is not necessary to store all the combinations, consuming less memory. In any case, both tend to optimize the subset 0/1 loss, despite it is not possible to guarantee to reach it.

EXPERIMENTS

The experiments were performed over several benchmark multilabel datasets whose main properties are shown in Table 1. As can be seen, there are significant differences in the number of attributes, instances, and labels. The cardinality—number of labels per instance—varies from 1.07 to 4.27. Concerning the number of labels, there are some datasets with just 5, 6, or 7 labels, whereas others have more than a hundred, even one of them has almost four hundred labels. The approaches for inference in PCC compared were those discussed along the paper,

TABLE 1 | Properties of the Datasets

Datasets	Instances	Attributes	Labels	Cardinality
bibtex	7395	1836	159	2.40
corel5k	5000	499	374	3.52
emotions	593	72	6	1.87
enron	1702	1001	53	3.38
flags	194	19	7	3.39
image	2000	135	5	1.24
mediamill*	5000	120	101	4.27
medical	978	1449	45	1.25
reuters	7119	243	7	1.24
scene	2407	294	6	1.07
slashdot	3782	1079	22	1.18
yeast	2417	103	14	4.24

except ES. No experiment was carried out with the ES method due to its computational cost. Hence, the methods compared were GS, ε-A algorithm for different values of ε (.0,.25,.5), BS for different values of the beam width b (1, 2, 3, 10) and the MS approaches, MC and EMC, with samples of different size (10, 50, 100), as suggested in Ref 23. Let us remember that the ε-A algorithm with ε=.5 is equivalent to GS and to BS with b = 1.

The results are presented in terms of the example-based subset 0/1 loss estimated by means of a 10-fold cross-validation. The base learner employed to obtain the binary classifiers that compose all these multilabel models was *logistic regression*[30] with probabilistic output. The regularization parameter C was established for each *individual* binary classifier performing a grid search over the values $C \in \{10^{-3}, 10^{-2}, \ldots, 10^{3}\}$ optimizing the Brier loss estimated by means of a balanced twofold cross validation repeated five times. The Brier loss[31] is a proper score that measures the accuracy of probabilistic predictions. The expression is $\frac{1}{n}\sum_{i=1}^{n}\left(\hat{p}_i - a_i\right)^2$, where for an instance i, p_i is the predicted probability that label i is relevant, and a_i is the actual label value (0 or 1).

Tables 2–4 respectively show the subset 0/1 loss, the number of nodes explored, and the averaged computational time (in seconds) per test instance for the different methods compared. The number of nodes explored and the computational time for the Monte Carlo approaches should be the same as those of GS multiplied by the size of the sample drawn. Maybe slight differences can occur in time due to initialization or other implementation issues.

Before discussing the results of the tables, let us remember that only ε-A algorithm with ε=.0 provides Bayes optimal inference, like ES does. This means that it always predicts the label combination with the highest joint conditional probability. Despite other methods may predict other label combination with lower joint conditional probability for some examples, they obtain better subset 0/1 scores in some few cases. This fact may be due to several reasons, mainly: (1) the relatively small size of the testing sets, and (2) the models h_j obtained to estimate the joint conditional probability $P(y \mid x)$ do not return true estimations usually. Theoretically, under perfect conditions (large test sets and perfect models), ε-A with ε=.0 would obtain the best scores.

Then, looking at Table 2 and as it is theoretically expected, the performance of ε-A algorithm decreases as the value of ε increases. As commented before, some exceptions can occur. In this case, it happens for flag and reuters datasets.

Concerning BS method, the performance increases as b increases (as theoretically expected), although some exceptions appear in bibtex, reuters, slashdot and yeast, due to the same reasons discussed above. Values from 1 to 3 were taken into account, because the performance gets steady when b = 3 for most of the datasets. Even more, it reaches steadiness when b = 3 for the half of the datasets and when b = 2 for both image and medical. In any case, a value equal to 10 was also considered to show some cases in which the predictions of the algorithm

TABLE 2 | Subset 0/1 Loss for the Different Methods

Datasets	ES ε-A (.0)	ε-A (.25)	GS BS(1) ε-A(.5)	BS(2)	BS(3)	BS (10)	MC (10)	MC (50)	MC (100)	EMC (10)	EMC (50)	EMC (100)
bibtex	81.92	81.95	82.19	**81.88**	**81.92**	**81.92**	**84.02**	82.85	82.46	82.73	82.22	82.11
corel5k	97.48	98.62	98.90	98.30	98.04	**97.48**	**99.64**	98.98	98.26	98.72	97.70	**97.42**
emotions	71.16	71.82	72.83	72.16	71.32	**71.16**	**80.77**	73.68	73.84	72.83	72.33	72.50
enron	83.14	84.26	85.43	83.43	83.37	**83.14**	**92.95**	85.90	84.61	87.43	83.61	83.26
flags	87.13	87.16	**86.13**	88.21	**87.13**	**87.13**	**96.39**	91.21	89.24	91.29	90.71	90.18
image	68.35	**68.35**	69.75	**68.35**	**68.35**	**68.35**	**69.20**	**65.25**	**62.65**	**64.95**	**65.70**	**62.95**
mediamill*	83.86	84.58	85.80	84.10	**83.86**	**83.86**	**90.90**	85.88	84.88	85.34	**83.70**	**83.40**
medical	30.37	**30.37**	**30.67**	**30.37**	**30.37**	**30.37**	**31.19**	**30.16**	**30.67**	**30.16**	**30.16**	**30.16**
reuters	22.73	**22.70**	23.60	**22.69**	22.73	**22.73**	**25.37**	23.18	22.83	22.97	22.83	22.83
scene	31.86	**31.86**	33.28	31.90	**31.86**	**31.86**	**33.53**	**30.28**	**29.70**	**29.24**	**29.24**	**29.29**
slashdot	51.80	52.22	54.49	**51.77**	**51.80**	**51.80**	**56.45**	52.96	52.70	52.22	52.35	52.35
yeast	76.95	77.62	79.77	**76.83**	**77.08**	**76.95**	**85.52**	79.93	79.35	79.06	77.53	77.62

Those scores that are equal or better than optimal predictions reached by ε-A and ES are shown in bold.

TABLE 3 | Number of Explored Nodes for the Different Methods

Datasets	ε-A(.0)	ε-A(.25)	GS BS(1) ε-A(.5)	BS(2)	BS(3)	BS(10)	MC/EMC(10)	MC/EMC(50)	MC/EMC(100)
bibtex	289.3	184.0	160.0	319.0	477.0	1575.0	1590.0	7950.0	15900.0
corel5k	1474.2	517.1	375.0	749.0	1122.0	3725.0	3740.0	18700.0	37400.0
emotions	10.7	10.8	7.0	13.0	18.0	45.0	60.0	300.0	600.0
enron	114.8	77.3	54.0	107.0	159.0	515.0	530.0	2650.0	5300.0
flags	22.6	16.3	8.0	15.0	21.0	55.0	70.0	350.0	700.0
image	7.3	7.7	6.0	11.0	15.0	35.0	50.0	250.0	500.0
mediamill*	191.8	142.4	102.0	203.0	303.0	995.0	1010.0	5050.0	10100.0
medical	46.6	46.6	46.0	91.0	135.0	435.0	450.0	2250.0	4500.0
reuters	8.2	8.3	8.0	15.0	21.0	55.0	70.0	350.0	700.0
scene	7.2	7.3	7.0	13.0	18.0	45.0	60.0	300.0	600.0
slashdot	25.3	24.9	23.0	45.0	66.0	205.0	220.0	1100.0	2200.0
yeast	26.1	26.0	15.0	29.0	42.0	125.0	140.0	700.0	1400.0

TABLE 4 | Average Prediction Time (in Seconds) Per Example for All Methods

Datasets	ES ε-A(.0)	ε-A (.25)	GS BS (1) ε-A (.5)	BS(2)	BS(3)	BS(10)	MC (10)	MC (50)	MC (100)	EMC (10)	EMC (50)	EMC (100)
bibtex	0.0162	0.0099	0.0079	0.0110	0.0156	0.0507	0.3220	1.6099	3.2256	0.3207	1.6021	3.2065
corel5k	0.1360	0.0161	0.0110	0.0278	0.0404	0.1361	0.7607	3.8076	7.6294	0.7563	3.7815	7.5781
emotions	0.0006	0.0006	0.0004	0.0005	0.0006	0.0013	0.0120	0.0590	0.1177	0.0117	0.0585	0.1172
enron	0.0072	0.0030	0.0019	0.0039	0.0055	0.0172	0.1046	0.5227	1.0457	0.1039	0.5190	1.0388
flags	0.0012	0.0007	0.0004	0.0005	0.0007	0.0016	0.0139	0.0691	0.1381	0.0136	0.0680	0.1360
image	0.0005	0.0005	0.0004	0.0004	0.0005	0.0009	0.0100	0.0491	0.0978	0.0098	0.0485	0.0970
mediamill*	0.0115	0.0055	0.0037	0.0072	0.0105	0.0333	0.1992	0.9963	1.9937	0.1984	0.9910	1.9821
medical	0.0027	0.0027	0.0027	0.0033	0.0046	0.0144	0.0886	0.4422	0.8857	0.0881	0.4404	0.8823
reuters	0.0005	0.0005	0.0005	0.0005	0.0007	0.0016	0.0138	0.0684	0.1373	0.0137	0.0682	0.1368
scene	0.0005	0.0005	0.0004	0.0005	0.0006	0.0013	0.0119	0.0587	0.1172	0.0118	0.0584	0.1171
slashdot	0.0015	0.0014	0.0012	0.0016	0.0022	0.0063	0.0433	0.2159	0.4317	0.0429	0.2149	0.4296
yeast	0.0015	0.0010	0.0006	0.0010	0.0014	0.0040	0.0277	0.1380	0.2759	0.0274	0.1369	0.2737

converge to those of ε-A with ε=.0 or ES. Notice that this is at the cost of exploring much more solutions.

Both Monte Carlo approaches improve their performance as the size of the sample increases. However, there are some exceptions. In this sense, it happens that MC reaches slightly better subset 0/1 loss for a sample size of 50 than for a sample size of 100 in case of emotions and medical. Unfortunately, EMC has more exceptions, but this is quite logical, because EMC is more sensitive to the size of the sample. This is so because it takes the maximum joint conditional probability, which is more likely to change as the sample size increases. In case of MC, it would be necessary to enlarge enough the sample size to make the mode change. Comparing both approaches, it is clear that EMC converges faster than MC, because the subset 0/1 of EMC is quite lower than that of MC for the same size of the sample drawn, especially in case of size equal to 10. Even more, MC with a sample of size equal to 50 is hardly

TABLE 5 | Subset 0/1 Loss (With the Standard Deviation) for the Compared Methods Considering a Sample of Different Label Orders

Datasets	ε-A(.0)	ε-A(.25)	ε-A(.5)	BS(2)	MC(10)	EMC(10)
image (24)	**64.44 ± 2.62**	65.02 ± 2.45	66.69 ± 1.93	64.61 ± 2.72	71.05 ± 1.97	66.03 ± 2.69
emotions (144)	**71.55 ± 1.06**	71.71 ± 1.00	73.16 ± 1.17	71.75 ± 1.02	78.81 ± 1.38	72.36 ± 1.07
scene (144)	32.09 ± 1.26	32.11 ± 1.27	34.35 ± 1.68	32.09 ± 1.26	35.63 ± 1.50	**32.00 ± 1.30**
reuters (500)	**22.78 ± 0.29**	**22.78 ± 0.29**	23.69 ± 0.25	**22.78 ± 0.29**	25.29 ± 0.33	22.87 ± 0.28

The number of label orders are in parentheses and represent the 20% of the possible label orders, except in the case of reuters dataset which represent the 10%.

able to reach EMC with a sample of size equal to 10. Also, EMC with a sample size of 50 outperforms all MC for most of the datasets. Just flags and image are exceptions. EMC is an appealing approach to perform inference in PCC, however, sometimes, even with a large enough sample, EMC is able to reach the precision of ε –A with ε=.0 in some datasets.

With regard to the number of nodes explored (see Table 3), GS (equivalent to ε-A algorithm with ε=.5 and to BS(1)) is the method which explores the smallest number of nodes, as it only goes over one path in the tree. In fact, such number corresponds to the number of labels plus one, because the root of the tree is considered as an explored node. It follows the ε-A algorithm with ε=.25, because the BS(b) with b from 2 rapidly increases the number of nodes explored. However, let us remember that none of those methods guarantee to reach an optimal solution. Then, focusing on the method that theoretically reaches the optimum (the ε-A algorithm with ε=.0), it occurs that this particular case of the ε-A algorithm explores the greatest amount of nodes among the same algorithm with other values for ε, especially as the number of labels grows. However, the ε-A algorithm clearly outperforms the BS technique in number of nodes explored, even for ε=.0 and for low values of the beam width b. Regarding EMC and MC, the number of nodes considerably increases as the size of the sample drawn increases. Hence, they require to explore much more nodes to be closer to the optimal, despite sometimes their performance could be better.

The computational time is expected to be higher as more nodes are explored, and looking at Table 4 one can confirm that this is what indeed happens. At this respect, ε-A algorithm is actually quite faster than BS technique and MS. In the same way, considering each method separately and varying their parameters is shown that ε-A algorithm is faster as ε increases, whereas BS is faster as the beam width b dismisses. However, this is not surprising, because increasing the value of ε or dismissing the value of the beam width b means to explore less nodes of the tree. Analogously, the time needed for the Monte Carlo approaches enlarge as the size of the sample drawn grows.

Additional experiments were performed in order to analyze the possible influence of taking different label orders. Table 5 contains the average of the subset 0/1 scores for a set of random label orders. Particularly, it was taken 20% of the possible label orders for the datasets with five or six labels (24 and 144 different label orders, respectively) and 10% for reuters dataset that has seven labels (500 different label orders). The results confirm that ε-A algorithm obtains the best scores. The only exception is the scene dataset in which EMC performs slightly better.

As a conclusion, the ε-A algorithm can be considered the best alternative if one desires to guarantee good performance in terms of subset 0/1, taking care of not exceeding the number of nodes explored and the execution time, even in the case of taking ε=.0 for which an optimal solution is guaranteed to reach.

CONCLUSIONS

This study analyzes the methods that have been proposed so far for performing inference in probabilistic classifiers chains for MLC. The ε-approximate algorithm with ε=.0 is theoretically shown to reach an optimal solution in terms of subset 0/1 loss, unlike other approaches that only estimate it, like the method based on a BS, MS or even the same algorithm with $\varepsilon > 0$. Besides, it offers good behavior both in number of nodes explored and in computational time. Even, the ε-approximate algorithm with values of ε greater than .0 offers good behavior, because it considerably reduces both the number of nodes explored and execution time with regard to either beam search or MS, keeping similar performance in terms of subset 0/1. Although beam search and MS seem worse to optimize subset 0/1 loss, they offer some interesting benefits. First, these methods can be adapted to optimize other loss functions. Additionally, MS succeeds in discovering efficiently good label orders.

ACKNOWLEDGMENTS

This research has been supported by the Spanish Ministerio de Economía y Competitividad (grants TIN2011-23558 and TIN2015-65069).

REFERENCES

1. Gibaja E, Ventura S. Multi-label learning: a review of the state of the art and ongoing research. *Wiley Interdiscip Rev Data Mining Knowl Discov* 2014, 4:411–444.
2. Clare A, King RD. Knowledge discovery in multi-label phenotype data. In: *European Conference on Data Mining and Knowledge Discovery (2001)*, Freiburg, Germany, 2001, 42–53.
3. Zhang M-L, Zhou Z-H. Multilabel neural networks with applications to functional genomics and text categorization. *IEEE Trans Knowl Data Eng* 2006, 18:1338–1351.
4. Elisseeff A, Weston J. A Kernel method for multi-labelled classification. In: *Advances in Neural Information Processing Systems (NIPS 2001)* Vancouver, Canada. 2001, 681–687.
5. McCallum AK. Multi-label text classification with a mixture model trained by EM. In: *AAAI 99 Workshop on Text Learning*, Orlando, Florida, 1999.
6. Ghamrawi N, McCallum A. Collective multi-label classification. In: *ACM International Conference on Information and Knowledge Management*, Bremen, Germany, 2005, 195–200. New York, NY, USA: ACM.
7. Schapire RE, Singer Y. Boostexter: a boosting-based system for text categorization. *Mach Learn* 2000, 39:135–168.
8. Dembczyński K, Waegeman W, Cheng W, Hüllermeier E. On label dependence and loss minimization in multi-label classification. *Mach Learn* 2012, 88:5–45.
9. Dembczyński K, Cheng W, Hüllermeier E. Bayes optimal multilabel classification via probabilistic classifier chains. In: *ICML (2010)*, Haifa, Israel, 2010, 279–286.
10. Montañés E, Quevedo JR, del Coz JJ. Aggregating independent and dependent models to learn multi-label classifiers. In: *ECML/PKDD'11—Volume Part II*, Athens, Greece, 2011, 484–500. Berlin Heidelberg: Springer-Verlag.
11. Montañés E, Senge R, Barranquero J, Quevedo JR, del Coz JJ, Hüllermeier E. Dependent binary relevance models for multi-label classification. *Pattern Recogn* 2014, 47:1494–1508.
12. Read J, Pfahringer B, Holmes G, Frank E. Classifier chains for multi-label classification. *Mach Learn* 2011, 85:333–359.
13. Tsoumakas G, Katakis I, Vlahavas I. Mining multi-label data. In: *Data Mining and Knowledge Discovery Handbook*. New York, US: Springer; 2010, 667–685.
14. Cheng W, Hüllermeier E. Combining instance-based learning and logistic regression for multilabel classification. *Mach Learn* 2009, 76:211–225.
15. Godbole S, Sarawagi S. Discriminative methods for multi-labeled classification. In: *Pacific-Asia Conference on Knowledge Discovery and Data Mining (2004)*, Sydney, Australia, 2004, 22–30.
16. Fürnkranz J, Hüllermeier E, Loza Menca E, Brinker K. Multilabel classification via calibrated label ranking. *Mach Learn* 2008, 73:133–153.
17. Qi GJ, Hua XS, Rui Y, Tang J, Mei T, Zhang HJ. Correlative multi-label video annotation. In: *Proceedings of the International Conference on Multimedia*, Augsburg, Germany, 2007, 17–26. New York: ACM.
18. Read J, Pfahringer B, Holmes G. Multi-label classification using ensembles of pruned sets. In: *IEEE International Conference on Data Mining*, Pisa, Italy, 2008, 995–1000.
19. Tsoumakas G, Vlahavas I. Random k-labelsets: an ensemble method for multilabel classification. In: *ECML/PKDD'07*, LNCS, Warsaw, Poland, 2007, 406–417. Berlin Heidelberg: Springer.
20. Dembczynski K, Waegeman W, Hüllermeier E. An analysis of chaining in multi-label classification. In: Raedt LD, Bessière C, Dubois D, Doherty P, Frasconi P, Heintz F, Lucas PJF, eds. *ECAI: Frontiers in Artificial Intelligence and Applications*, Montpellier, France, vol. 242. Amsterdam, Netherlands: IOS Press; 2012, 294–299.
21. Kumar A, Vembu S, Menon AK, Elkan C. Learning and inference in probabilistic classifier chains with beam search. In: *ECML/PKDD (2012)*, Bristol, UK, 2012, 665–680.
22. Kumar A, Vembu S, Menon AK, Elkan C. Beam search algorithms for multilabel learning. *Mach Learn* 2013, 92:65–89.
23. Read J, Martino L, Luengo D. Efficient monte carlo methods for multi-dimensional learning with classifier chains. *Pattern Recogn* 2014, 47:1535–1546.
24. Read J, Martino L, Olmos PM, Luengo D. Scalable multi-output label prediction: from classifier chains to classifier trellises. *Pattern Recogn* 2015, 48:2096–2109.

25. Luaces Ó, Dez J, Barranquero J, del Coz JJ, Bahamonde A. Binary relevance efficacy for multilabel classification. *Prog Artif Intell* 2012, 4:303–313.
26. Read J, Pfahringer B, Holmes G, Frank E. Classifier chains for multi-label classification. In: *ECML/PKDD'09*, LNCS, Bled, Slovenia, 2009, 254–269. Berlin Heidelberg: Springer.
27. Senge R, del Coz JJ, Hüllermeier E. On the problem of error propagation in classifier chains for multi-label classification. In: *Conference of the German Classification Society on Data Analysis, Machine Learning and Knowledge Discovery (2012)*, Hildesheim, Germany, 2012.
28. Senge R, del Coz JJ, Hüllermeier E. Rectifying classifier chains for multi-label classification. In: *LWA 2013: Lernen, Wissen & Adaptivität, Workshop Proceedings Bamberg*, Bamberg, Germany, 2013, 151–158.
29. Read J, Martino L, Olmos PM, Luengo D. Scalable multi-output label prediction: from classifier chains to classifier trellises. *Pattern Recogn* 2015, 48:2096–2109.
30. Lin C-J, Weng RC, Keerthi SS. Trust region Newton method for logistic regression. *J Mach Learn Res* 2008, 9:627–650.
31. Brier GW. Verification of forecasts expressed in terms of probability. *Mon Weather Rev* 1950, 78:1–3.

5.2. A family of admissible heuristics for A* to perform inference in probabilistic classifier chains

En esta sección se incluye el artículo de revista:

- D. Mena, E. Montañés, J. R. Quevedo, and J. J. del Coz, "A family of admissible heuristics for A* to perform inference in probabilistic classifier chains," Machine Learning, vol. 106, no. 1, pp. 143–169, 2017.

Mach Learn
DOI 10.1007/s10994-016-5593-5

A family of admissible heuristics for A* to perform inference in probabilistic classifier chains

Deiner Mena[1,2] · **Elena Montañés**[1] · **José Ramón Quevedo**[1] · **Juan José del Coz**[1]

Received: 14 September 2015 / Accepted: 15 September 2016

Abstract Probabilistic classifier chains have recently gained interest in multi-label classification, due to their ability to optimally estimate the joint probability of a set of labels. The main hindrance is the excessive computational cost of performing inference in the prediction stage. This pitfall has opened the door to propose efficient inference alternatives that avoid exploring all the possible solutions. The ϵ-approximate algorithm, beam search and Monte Carlo sampling are appropriate techniques, but only ϵ-approximate algorithm with $\epsilon = 0$ theoretically guarantees reaching an optimal solution in terms of subset 0/1 loss. This paper offers another alternative based on heuristic search that keeps such optimality. It consists of applying the A* algorithm providing an admissible heuristic able to explore fewer nodes than the ϵ-approximate algorithm with $\epsilon = 0$. A preliminary study has already coped with this goal, but at the expense of the high computational time of evaluating the heuristic and only for linear models. In this paper, we propose a family of heuristics defined by a parameter that controls the trade-off between the number of nodes explored and the cost of computing the heuristic. Besides, a certain value of the parameter provides a method that is also suitable for the non-linear case. The experiments reported over several benchmark datasets show that the number of nodes explored remains quite steady for different values of the parameter, although the time considerably increases for high values. Hence, low values of the parameter

Editor: Eyke Hüllermeier.

✉ Elena Montañés
elena@aic.uniovi.es

Deiner Mena
deiner@aic.uniovi.es; deiner.mena@utch.edu.co

José Ramón Quevedo
quevedo@aic.uniovi.es

Juan José del Coz
juanjo@aic.uniovi.es

[1] Artificial Intelligence Center, University of Oviedo, Campus de Viesques, s/n, 33204 Gijón, Asturias, Spain

[2] Cra. 22 No 18B-10 B/ Nicolás Medrano - Ciudadela Universitaria, Quibdó - Chocó, Colombia

Published online: 26 September 2016

give heuristics that theoretically guarantee exploring fewer nodes than the ϵ-approximate algorithm with $\epsilon = 0$ and show competitive computational time. Finally, the results exhibit the good behavior of the A* algorithm using these heuristics in complex situations such as the presence of noise.

Keywords Multi-label classification · Probabilistic classifier chains · Inference · A* · Admissible heuristics

1 Introduction

Multi-label classification (MLC) is a machine learning problem in which models are sought that assign a subset of (classes) labels to each instance, unlike conventional (single-class) classification that involves predicting only a single class. Multi-label classification problems are ubiquitous and naturally occur, for instance, in assigning keywords to a paper, tags to resources in a social network, objects to images or emotional expressions to human faces.

In general, the problem of multi-label learning comes with two fundamental challenges. The first refers to the computational complexity of the algorithms. If the number of labels is large, then a complex approach might not be applicable in practice. Therefore, the scalability of algorithms is a key issue in this field. The second problem is related to the 'own nature' of multi-label data. Not only is the number of labels typically large, but each instance also belongs to a variable-sized subset of labels simultaneously. Moreover, and perhaps even more importantly, the labels will normally not occur independently of each other; instead, there are statistical dependencies between them. From a learning and prediction point of view, these relationships constitute a promising source of information, in addition to that coming from the mere description of the instances. Thus, it is hardly surprising that research on MLC has very much focused on the design of new methods that are able to detect—and benefit from—interdependencies among labels.

Several approaches have been proposed in the literature to cope with MLC. Firstly, researchers tried to adapt and extend different state-of-the-art binary or multi-class classification algorithms (Elisseeff and Weston 2005; McCallum 1999; Zhang and Zhou 2007). Secondly, they further analyzed in depth the label dependence and attempt to design new approaches that exploit label correlations (Dembczyński et al. 2012). In this regard, two kinds of label dependence have been formally distinguished, namely, conditional dependence (Dembczyński et al. 2010; Montañés 2011, 2014; Read et al. 2011) and marginal (unconditional) dependence (Cheng and Hüllermeier 2009). Also, pairwise relations (Elisseeff and Weston 2005), relations in sets of different sizes (Read et al. 2011; Tsoumakas and Vlahavas 2007), or relations in the whole set of labels (Cheng and Hüllermeier 2009; Montañés 2011) have also been exploited.

Regarding conditional label dependence, the approach called probabilistic classifier chains (PCC) has aroused great interest among the multi-label community, since it offers the excellent property of being able to estimate the conditional joint distribution of the labels. However, the original PCC algorithm (Dembczyński et al. 2010) suffers from high computational cost, since it performs an exhaustive search as inference strategy to obtain optimal solutions in terms of a given loss function. Several efforts that use different searching and sampling strategies in order to overcome this drawback are being made currently. This includes uniform-cost search (Dembczynski et al. 2012), beam search (Kumar et al. 2012, 2013) and Monte Carlo sampling (Dembczynski et al. 2012; Read et al. 2014). All of these algorithms successfully

estimate an optimal solution reached by the original PCC (Dembczyński et al. 2010), at the same time that they reduce the computational cost in terms of both the number of candidate solutions explored and execution time. The main contribution of this paper is to propose an alternative based on an heuristic search strategy. In particular, the proposal consists of obtaining admissible heuristics for the well-known A* algorithm (Hart et al. 1968). In this respect, we have already started to fill this gap in the literature with a recent published preliminary work (Mena et al. 2015), concluding that the proposal guarantees, not only optimal predictions in terms of subset 0/1 loss, but also that it explores fewer nodes than all previous methods that also provide optimal predictions. Unfortunately, and after studying in depth this heuristic, two main drawbacks can be stated: (i) it could only be defined for linear models and (ii) its computation is moderately high. In this direction, the goal of this paper is twofold. On the one hand, this work goes further by defining a family of heuristics through a parameter that controls the trade-off between the number of nodes explored and the cost of computing the heuristic. Besides, a special value of the parameter leads to an heuristic suitable for non-linear models. On the other hand, this work also studies and analyzes situations in which the computation of the heuristic compensates the whole computational cost, showing a steady behavior with regard to other algorithms. All these methods are analyzed and experimentally compared over a wide range of multi-label datasets.

The rest of the paper is organized as follows. Section 2 formally describes the multi-label framework and the principles of PCC. Section 3 describes and discusses the properties and behavior of the different state-of-the-art approaches for inference in PCCs. Section 4 details the heuristic search framework and defines admissible heuristics for the A* algorithm. Exhaustive experiments are shown and discussed in Sect. 5. Finally, Sect. 6 offers some conclusions and includes new directions for future work.

2 Probabilistic classifier chains in multi-label classification

This section formally describes the MLC task and the PCC methods.

2.1 Formal settings of multi-label classification and loss functions

Let be $\mathcal{L} = \{\ell_1, \ell_2, \ldots, \ell_m\}$ a finite and non-empty set of m labels and $S = \{(\boldsymbol{x}_1, \boldsymbol{y}_1), \ldots, (\boldsymbol{x}_n, \boldsymbol{y}_n)\}$ a training set independently and randomly drawn according to an unknown probability distribution $\mathbf{P}(\mathbf{X}, \mathbf{Y})$ on $\mathcal{X} \times \mathcal{Y}$, where $\mathcal{X}$ and $\mathcal{Y}$ are the input and the output space, respectively. The former is the space of the instance description, whereas the latter is given by the power set $\mathcal{P}(\mathcal{L})$ of $\mathcal{L}$. To ease notation, we define $\boldsymbol{y}_i$ as a binary vector $\boldsymbol{y}_i = (y_{i,1}, y_{i,2}, \ldots, y_{i,m})$ in which $y_{i,j} = 1$ indicates the presence (relevance) and $y_{i,j} = 0$ the absence (irrelevance) of ℓ_j in the labeling of $\boldsymbol{x}_i$. Hence, $\boldsymbol{y}_i$ is the realization of a corresponding random vector $\mathbf{Y} = (\mathbf{Y_1}, \mathbf{Y_2}, \ldots, \mathbf{Y_m})$. Using this convention, the output space can also be defined as $\mathcal{Y} = \{0, 1\}^m$. The goal in MLC is to induce from S a hypothesis $\boldsymbol{f} : \mathcal{X} \longrightarrow \mathcal{Y}$ that minimizes the risk in terms of certain loss function $L(\cdot)$ when it provides a vector of relevant labels $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x}) = (f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_m(\boldsymbol{x}))$ for unlabeled query instances $\boldsymbol{x}$. This risk can be defined as the expected loss over the joint distribution $\mathbf{P}(\mathbf{X}, \mathbf{Y})$, that is,

$$R_L(\boldsymbol{f}) = \mathbb{E}_{\mathbf{X},\mathbf{Y}} L(\mathbf{Y}, \boldsymbol{f}(\mathbf{X})). \tag{1}$$

therefore, denoted by $\mathbf{P}(\boldsymbol{y} \mid \boldsymbol{x})$ the conditional distribution $\mathbf{Y} = \boldsymbol{y}$ given $\mathbf{X} = \boldsymbol{x}$, then the so-called risk minimizer $\boldsymbol{f}^*$ can be expressed by

$$f^*(\boldsymbol{x}) = \arg\min_{f} \sum_{\boldsymbol{y}\in\mathcal{Y}} \mathbf{P}(\boldsymbol{y} \mid \boldsymbol{x}) L(\boldsymbol{y}, f(\boldsymbol{x})). \tag{2}$$

Let us comment that the conditional distribution $\mathbf{P}(\boldsymbol{y} \mid \boldsymbol{x})$ presents different properties which are crucial for optimizing different loss functions. In this respect, the strategy followed by a certain MLC algorithm for modeling label dependence determines the loss function that is optimized. But, unfortunately, for most of the algorithms it is quite complex and confusing to discover the loss function they attempt to optimize.

With regard to the loss functions, several performance measures have been taken for evaluating MLC. The most specific ones are the subset 0/1 loss and the Hamming loss, but there exist other measures that have been taken from other research fields, as such F_1 or the $Jaccard$ index. Here, we will focus on just the subset 0/1 loss, since it is the measure PCCs are able to optimize. The subset 0/1 loss checks if the predicted and relevant label subsets are equal or not and is defined by[1]

$$L_{S_{0/1}}(\boldsymbol{y}, f(\boldsymbol{x})) = [\![\boldsymbol{y} \neq f(\boldsymbol{x})]\!]. \tag{3}$$

In the case of this evaluation measure, it is sufficient to take the mode of the entire joint conditional distribution for optimizing this loss. Formally, the risk minimizer adopts the following simplified form

$$f^*(\boldsymbol{x}) = \arg\max_{\boldsymbol{y}\in\mathcal{Y}} \mathbf{P}(\boldsymbol{y} \mid \boldsymbol{x}). \tag{4}$$

2.2 Probabilistic classifier chains

PCCs (Dembczyński et al. 2010) [such as CC (Read et al. 2009, 2011)] are based on learning a chain of classifiers. These methods take an order of the label set and train a probabilistic binary classifier for estimating $\mathbf{P}(y_j \mid \boldsymbol{x}, y_1, \ldots, y_{j-1})$ for each label ℓ_j following this order. Hence, the probabilistic model obtained for predicting label ℓ_j, denoted by f_j is of the form

$$f_j : \mathcal{X} \times \{0, 1\}^{j-1} \longrightarrow [0, 1]. \tag{5}$$

The training data for this classifier is the set $S_j = \{(\overline{\boldsymbol{x}}_1, y_{1,j}), \ldots, (\overline{\boldsymbol{x}}_n, y_{n,j})\}$ where $\overline{\boldsymbol{x}}_i = (\boldsymbol{x}_i, y_{i,1}, \ldots, y_{i,j-1})$, that is, the features are $\boldsymbol{x}_i$ supplemented by the relevance of the labels $\ell_1, \ldots, \ell_{j-1}$ preceding ℓ_j in the chain and the category is the relevance of the label ℓ_j.

In the testing stage of the methods based on learning a chain of classifiers, the goal is to perform inference for each instance, which consists of estimating the risk minimizer for a given loss function over the estimated entire joint conditional distribution. The idea revolves around repeatedly applying the general product rule of probability to the joint distribution of the labels $\mathbf{Y} = (\mathbf{Y_1}, \mathbf{Y_2}, \ldots, \mathbf{Y_m})$, that is, computing

$$\mathbf{P}(\boldsymbol{y} \mid \boldsymbol{x}) = \prod_{j=1}^{m} \mathbf{P}(y_j \mid \boldsymbol{x}, y_1, \ldots, y_{j-1}). \tag{6}$$

Before analyzing this issue in the next section, note that from a theoretical point of view, this expression holds for any order considered for the labels. But, in practice, these methods are label order dependent for several reasons. On the one hand, it is not possible to assure that the models obtained in the training stage perfectly estimate the joint conditional probability $P(\boldsymbol{y} \mid \boldsymbol{x})$. On the other hand, predicted values instead of true values are successively taken in

[1] For a predicate p, the expression $[\![p]\!]$ evaluates to 1 if p is true and 0 otherwise.

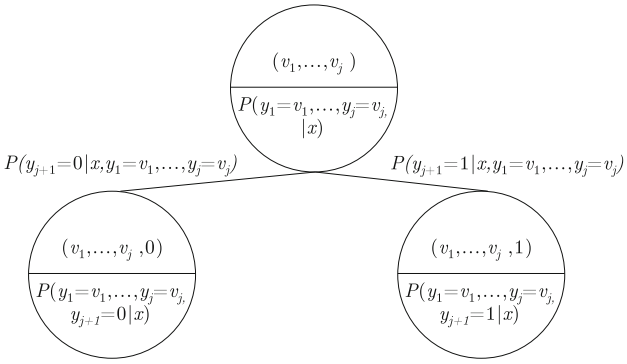


Fig. 1 A generic node and its children of the probability binary tree. The *top part* of each node contains the combination of labels and the *bottom part* includes the joint probability of such a combination. The *edges* are labeled with the conditional probability

the testing stage. This is more serious if the highest errors occur at the beginning of the chain, since error predictions are successively propagated (Montañés 2014; Senge et al. 2012a, b). In any case, in this paper we assume the order of the labels in the chain to be given, since the goal is just to analyze the performance of the methods, without taking into account the effect of different orders. Hence, we do not include any study about which order can be the best.

Before going into the detailed description of the state-of-the-art of the inference approaches and of our proposal, we note that the training phase is common to all of them, thus, the models f_j induced by the binary classifiers will be the same. So, in what follows we will focus just on the testing stage.

3 Inference in probabilistic classifier chains

First of all, let us consider the task of performing different inference procedures as different manners of exploring a probability binary tree in order to facilitate the explanation and analysis of the inference approaches in the next section. In such a tree, the k-th node of level $j < m$ with $k \leq 2^j$ is labeled by $y_j^k = (v_1, v_2, \ldots, v_j)$ with $v_i \in \{0, 1\}$ for $i = 1, \ldots, j$. This node has two children respectively labeled as $y_{j+1}^{2k-1} = (v_1, v_2, \ldots, v_j, 0)$ and $y_{j+1}^{2k} = (v_1, v_2, \ldots, v_j, 1)$ and with marginal joint conditional probability $\mathbf{P}(y_1 = v_1, \ldots, y_j = v_j, y_{j+1} = 0 \,|\, \boldsymbol{x})$ and $\mathbf{P}(y_1 = v_1, \ldots, y_j = v_j, y_{j+1} = 1 \,|\, \boldsymbol{x})$. The weights of the edges between both parent and children are respectively $\mathbf{P}(y_{j+1} = 0 \,|\, \boldsymbol{x}, y_1 = v_1, \ldots, y_j = v_j)$ and $\mathbf{P}(y_{j+1} = 1 \,|\, \boldsymbol{x}, y_1 = v_1, \ldots, y_j = v_j)$, which are respectively estimated by $1 - f_{j+1}(\boldsymbol{x}, v_1, \ldots, v_j)$ and $f_{j+1}(\boldsymbol{x}, v_1, \ldots, v_j)$. The marginal joint conditional probability of the children is computed by the product rule of probability. Then, $\mathbf{P}(y_1 = v_1, \ldots, y_j = v_j, y_{j+1} = 0 \,|\, \boldsymbol{x}) = \mathbf{P}(y_{j+1} = 0 \,|\, \boldsymbol{x}, y_1 = v_1, \ldots, y_j = v_j) \cdot \mathbf{P}(y_1 = v_1, \ldots, y_j = v_j \,|\, \boldsymbol{x})$ and $\mathbf{P}(y_1 = v_1, \ldots, y_j = v_j, y_{j+1} = 1 \,|\, \boldsymbol{x}) = \mathbf{P}(y_{j+1} = 1 \,|\, \boldsymbol{x}, y_1 = v_1, \ldots, y_j = v_j) \cdot \mathbf{P}(y_1 = v_1, \ldots, y_j = v_j \,|\, \boldsymbol{x})$. The root node is labeled by the empty set. Figure 1 illustrates this.

Several approaches have been proposed for inference in PCCs. The method the first proposed is the one based on greedy search (GS), being the integral part of the original CC method (Read et al. 2009). Its successor is the exhaustive search (ES), called the PCC method (Dembczyński et al. 2010). The ϵ-approximate (ϵ-A) algorithm (Dembczynski et al. 2012) is a

uniform-cost (UC) search algorithm that can output optimal predictions in terms of subset 0/1 loss and also reduces significantly the computational cost of ES. A more recent approach based on beam search (Kumar et al. 2012, 2013) (BS) presents good behavior both in terms of performance and computational cost. Finally, Monte Carlo sampling (Dembczynski et al. 2012) is an appealing and simpler alternative (Dembczynski et al. 2012; Read et al. 2014) to overcome the high computational cost of ES.

This section copes with the particularities of these inference methods, except the Monte Carlo sampling approaches. We have already studied Monte Carlo approaches (Dembczynski et al. 2012; Read et al. 2014) and compared them with the ϵ-A algorithm and BS techniques (Mena et al. 2015). The conclusions of that work were that, although they are well suited for minimization of other losses, e.g., Hamming loss or example-based F-measure, (i) they need to explore many more nodes to be closer to the optimal, despite the fact that their performance could sometimes be better, (ii) they enlarge as the size of the sample drawn grows and (iii) they are quite slow even for low values of the sample. Hence, we do not consider them as competitive methods in the present work, although they could sometimes be appealing.

3.1 Greedy search

At the testing stage, the GS strategy, originally called CC (Read et al. 2009, 2011), provides an output $\hat{\boldsymbol{y}} = (\hat{y}_1, \ldots, \hat{y}_m)$ for a new unlabeled instance $\boldsymbol{x}$ by successively querying each classifier f_j that estimates the conditional probability $\mathbf{P}(y_j \mid \boldsymbol{x}, y_1, \ldots, y_{j-1})$. This means exploring just one node in each level j. Given that only the two children of the explored node in level j are taken, their marginal joint conditional probability only differs in the marginal conditional probability $\mathbf{P}(y_j \mid \boldsymbol{x}, y_1, \ldots, y_{j-1})$, since both children have the same parent. Thus, the path selected is that of the child with the highest marginal conditional probability $\mathbf{P}(y_j \mid \boldsymbol{x}, y_1, \ldots, y_{j-1})$ and the prediction for an instance $\boldsymbol{x}$ is of the form

$$\hat{\boldsymbol{y}} = (\, f_1(\boldsymbol{x}),\ f_2(\boldsymbol{x}, f_1(\boldsymbol{x})),\ f_3(\boldsymbol{x}, f_1(\boldsymbol{x}), f_2(\boldsymbol{x}, f_1(\boldsymbol{x}))), \ldots). \tag{7}$$

Figure 2a shows the path followed by an instance using this strategy. In this example, only the right node is explored in each level. The optimal solution is not reached, since the optimal solution is that which ends in the sixth leaf, whereas the method falls in the last leaf.

Concerning the optimization of subset 0/1 loss, a rigorous analysis (Dembczynski et al. 2012) establishes bounds for the performance of the GS, showing the poor performance of the method for this loss, although it tends to optimize it.

3.2 Exhaustive search

Unlike GS that explores only a single label combination, ES estimates the entire joint conditional distribution $\mathbf{P}(\cdot \mid \boldsymbol{x})$ for a new unlabeled instance $\boldsymbol{x}$, since it provides a Bayes optimal inference. Hence, it explores all possible paths in the tree. Then, for each $h(\boldsymbol{x})$, it computes $\mathbf{P}(\boldsymbol{y} \mid \boldsymbol{x})$ and $L(\boldsymbol{y}, \boldsymbol{f}(\boldsymbol{x}))$ for all combination of labels $\boldsymbol{y} = (y_1, y_2, \ldots, y_m)$ and outputs the combination $\hat{\boldsymbol{y}} = (\hat{y}_1, \ldots, \hat{y}_m) = \boldsymbol{f}^*(\boldsymbol{x})$ with minimum risk for the given loss $L(\cdot, \cdot)$. By doing so, it generally improves in terms of performance, since it perfectly estimates the risk minimizer, albeit at the cost of a higher computational cost, as it comes down to summing over an exponential (2^m) number of label combinations for each $h(\boldsymbol{x})$.

Figure 2b illustrates this approach where, by exploring all paths, the optimal solution is always reached.

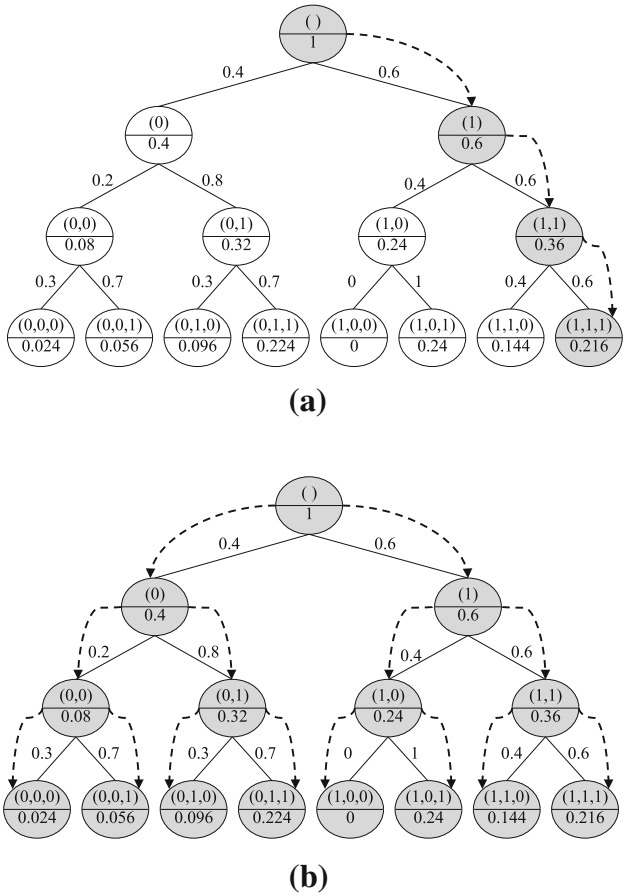


Fig. 2 An example of paths followed by an instance using **a** Greedy Search (CC) and **b** exhaustive Search (PCC). The *dotted arrows* show the path followed by the algorithm

3.3 ϵ-approximate algorithm

The ϵ-Approximate (ϵ-A) algorithm (Dembczynski et al. 2012) arises as an alternative to the high computational cost of ES and to the poor performance of CC. In terms of the probability tree defined above, it expands only the nodes whose marginal joint conditional probability exceeds the threshold $\epsilon = 2^{-k}$ with $1 \leq k \leq m$ (notice that $\epsilon = 0$ and all values of ϵ between 0 and 2^{-m} are in fact the same case of $\epsilon = 2^{-m}$). This marginal joint conditional probability for a node in level j, which deals with the label ℓ_j and for an unlabeled instance $\boldsymbol{x}$ is

$$\mathbf{P}(y_1, \ldots, y_j \,|\, \boldsymbol{x}) = \prod_{i=1}^{j} \mathbf{P}(y_i \,|\, \boldsymbol{x}, y_1, \ldots, y_{i-1}), \tag{8}$$

where $\mathbf{P}(y_i, | \, \boldsymbol{x}, y_1, \ldots, y_{i-1})$ is estimated by $f_i(\boldsymbol{x}, y_1, \ldots, y_{i-1})$.

The nodes are expanded in the order established by this probability, calculating the marginal joint conditional probability for their children. So, the algorithm does not follow a specific path, otherwise it changes from one path to another depending on the marginal joint

conditional probabilities. In the end, two situations can be found: (i) the node expanded is a leaf or (ii) there are no more nodes that exceed the threshold. If the former situation happens, the prediction for the unlabeled instance $\boldsymbol{x}$ will be $\hat{\boldsymbol{y}} = (\hat{y}_1, \ldots, \hat{y}_m)$ corresponding to the combination of the leaf reached (see Fig. 3a). Conversely, if situation (ii) takes place, then GS is applied to the nodes whose children do not exceed the threshold, and the prediction $\hat{\boldsymbol{y}} = (\hat{y}_1, \ldots, \hat{y}_m)$ for the unlabeled instance $\boldsymbol{x}$ in this case will be that with the highest entire joint conditional probability $\mathbf{P}(y_1, \ldots, y_m \mid \boldsymbol{x})$ (see Fig. 3b, c).

The parameter ϵ plays an important role in the algorithm. The particular case of $\epsilon = 0$ (or any value in the interval $[0, 2^{-m}]$, that is, $k = m$) is of special interest, since the algorithm performs a UC that always finds the optimal solution. Figure 3a illustrates this situation.

Conversely, the method is looking to GS as ϵ grows, being the GS in the case of $\epsilon = 0.5$ (or equivalently $\epsilon = 2^{-1}$, that is, $k = 1$). This is so because in this case two situations are possible: (i) only one node has a marginal joint conditional probability greater than ϵ, in which case the algorithm follows one path, or (ii) no nodes have a marginal joint conditional probability greater than ϵ, in which case a GS is applied from here to the bottom of the tree. Figure 3b shows an example of this particular case.

Notice that this method provides an optimal prediction if the entire joint conditional probability of the corresponding label combination is greater than ϵ. The interpretation of the method for a generic value of $\epsilon = 2^{-k}$ is that the method guarantees reaching a partial optimal solution at least until the $k - th$ level on the tree. Figure 3c shows the particular case of $\epsilon = 0.25$.

Consequently, this algorithm estimates the risk minimizer for the subset 0/1 loss to a greater or lesser extend, depending on the value of ϵ. Moreover, a theoretical analysis of this estimation (Dembczynski et al. 2012) allows bounding its goodness as a function of the number of iterations, which in turn depends on ϵ.

3.4 Beam search

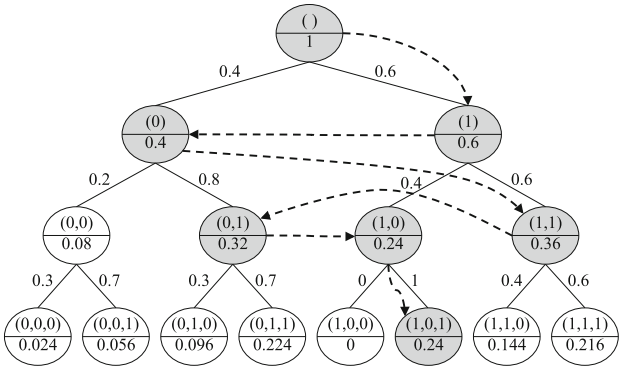
Beam Search (BS) (Kumar et al. 2012, 2013) also explores more than one path in the probabilistic tree. This method includes a parameter b called the beam width that limits the number of combinations of labels explored. The idea is to explore b possible candidate sets of labels at each level of the tree. Hence, depending on such a value, a certain number of the top levels are exhaustively explored, particularly a total of $k^* - 1$ levels, k^* being the lowest integer such that $b < 2^{k^*}$. Then, only b number of possibilities are explored for each of the remainder levels. The combinations explored from the level k^* to the bottom are those with the highest marginal joint conditional probability seen thus far. This marginal joint conditional probability for a node of level j for an unlabeled instance $\boldsymbol{x}$ is the same as for the ϵ-A algorithm. Hence, such a probability is

$$\mathbf{P}(y_1, \ldots, y_j \mid \boldsymbol{x}) = \prod_{i=1}^{j} \mathbf{P}(y_i \mid \boldsymbol{x}, y_1, \ldots, y_{i-1}), \tag{9}$$

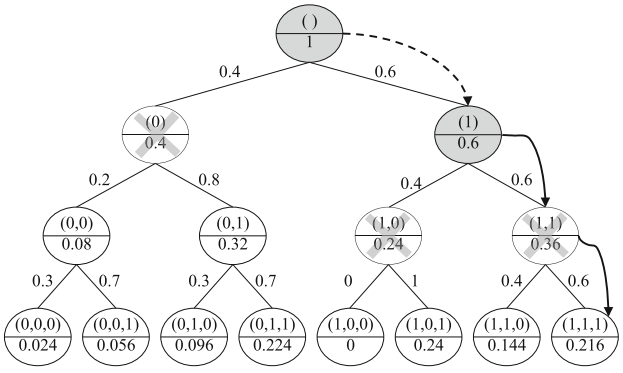
where $\mathbf{P}(y_i, \mid \boldsymbol{x}, y_1, \ldots, y_{i-1})$ is estimated by $f_i(\boldsymbol{x}, y_1, \ldots, y_{i-1})$.

In the end, the algorithm outputs $\hat{\boldsymbol{y}} = (\hat{y}_1, \ldots, \hat{y}_m)$ with the highest entire joint conditional probability $\mathbf{P}(y_1, \ldots, y_m \mid \boldsymbol{x})$.

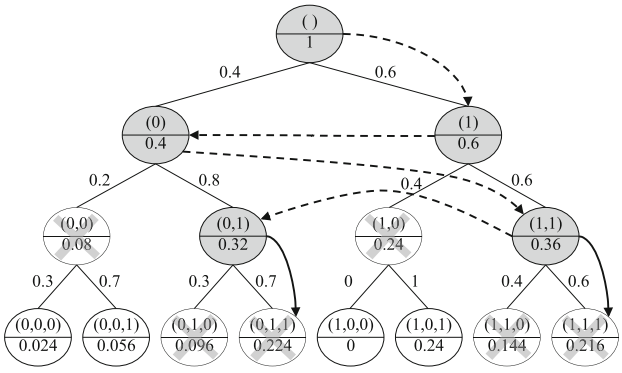
BS differs from GS in that (i) BS explores more than one combination whereas GS just explores one and also in (ii) the probability taken for deciding a path to follow in the tree is different from one method to the other. Concerning (ii), both take the marginal joint conditional probability $\mathbf{P}(y_1, \ldots, y_j | \boldsymbol{x})$, but in the case of GS, that is equivalent to taking



(a)



(b)



(c)

Fig. 3 Several examples of the paths followed by the ϵ-A algorithm for different values of ϵ. The *nodes with a cross* are those that have a marginal joint conditional probability lower than ϵ and, hence, they are not explored any more. The *dotted arrows* show the path followed by the algorithm. The *solid arrows* indicate the path followed by the algorithm when the marginal joint conditional probability does not exceed the value of ϵ, and, hence a GS is applied to this node from here to the *bottom* of the tree. **a** ϵ-A algorithm with $\epsilon = 0$ ($k = m$), **b** ϵ-A algorithm with $\epsilon = 0.5$ ($k = 1$) and **c** ϵ-A algorithm with $\epsilon = 0.25$ ($k = 2$)

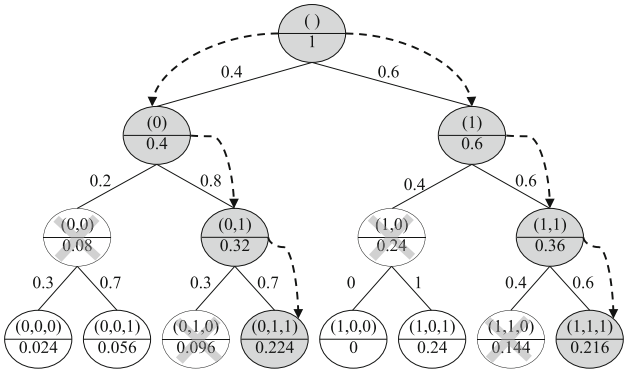


Fig. 4 An example of paths followed by an instance using Beam Search (BS) with $b = 2$. The *nodes with a cross* mean that this node has none of the highest marginal joint conditional probabilities and, hence, it is not explored any more. The *dotted arrows* show the path followed by the algorithm

the marginal conditional probability $\mathbf{P}(y_j, | \boldsymbol{x}, y_1, \ldots, y_{j-1})$ since the two nodes explored in each level of the tree have the same parent, as explained before. However, in the case of BS, the b nodes explored do not have to have the same parent, even in the case of $b = 2$. Of course, if $b > 2$, this is impossible to happen in a binary tree.

Let consider the case of $b = 1$. In this case, BS expands just one node in each level, the one with the highest marginal joint conditional probability that coincides with the highest marginal conditional probability, so BS when $b = 1$ follows just one path that coincides with the one followed by GS. Also, if $b = 2^m$, BS performs an ES. Hence, BS encapsulates both GS and ES respectively considering $b = 1$ and $b = 2^m$. This makes it possible to control the trade-off between computational cost and performance of the method by tuning b between 1 and 2^m.

As a final remark, the fact that BS considers marginal joint conditional probabilities makes the method tend to estimate the risk minimizer for the subset 0/1 loss. Kumar et al. (2012) and Kumar et al. (2013) do not include any theoretical analysis about the goodness of the estimation of the risk minimizer, but they empirically show that by taking certain values of b ($b < 15$), the risk minimizer provided by the method converges to the one obtained using ES.

Figure 4 shows an example of the paths explored by the BS algorithm when $b = 2$.

4 A* algorithm for inference in PCC

The algorithm A* is the most widely-known form of best-first search (Pearl 1984), in which the *best* node at each iteration, according to an evaluation function e, is expanded. The particularity of the A* algorithm is that e can be seen as a function E of the other two functions g and h, $e(k) = E(g(k), h(k))$, where $g(k)$ evaluates the cost of reaching node k from the root and $h(k)$ evaluates the cost of reaching a solution (a leaf) from k. Hence, $e(k)$ evaluates the total cost of reaching a solution from the root through k. In general, it is possible to obtain the exact value of the known information (g), but the unknown information must be estimated through an heuristic (h). To obtain an optimal solution, h must not overestimate the actual cost of reaching a solution, that is, it must be an admissible heuristic. These kinds of heuristics are optimistic, because they estimate that the cost of obtaining a solution is less

than it actually is. Also, the A* algorithm is optimally efficient for any heuristic, because no other optimal algorithm using the same heuristic guarantees to expand fewer nodes than A*.

4.1 Building an admissible heuristic

In order to adapt A* for inference in PCC, we must take into account that we have probabilities instead of costs. So, (1) A* must select the node with the highest estimated probability, (2) h must not underestimate the probability from the node to a leaf, that is, h must be an admissible heuristic[2] and (3) E must be the product function, $e = g \cdot h$. Considering all these aspects, e will provide an estimation of the entire joint conditional probability $\mathbf{P}(y_1, \ldots, y_m \mid \boldsymbol{x})$ for optimizing subset 0/1 loss. In order to derive g and h, let us say that the product rule of probability to the joint distribution of the labels $\mathbf{Y} = (\mathbf{Y_1}, \mathbf{Y_2}, \ldots, \mathbf{Y_m})$ can be rewritten as

$$\begin{aligned}\mathbf{P}(y_1, \ldots, y_m \mid \boldsymbol{x}) = \mathbf{P}(y_1, \ldots, y_j \mid \boldsymbol{x}) \\ \times \mathbf{P}(y_{j+1}, \ldots, y_m \mid \boldsymbol{x}, y_1, \ldots, y_j).\end{aligned} \tag{10}$$

Hence, for a node at level j, let us consider g to be the marginal joint conditional probability $\mathbf{P}(y_1, \ldots, y_j \mid \boldsymbol{x})$ and h an heuristic that does not underestimate the marginal joint conditional probability $\mathbf{P}(y_{j+1}, \ldots, y_m \mid \boldsymbol{x}, y_1, \ldots, y_j)$. Let us remember that the values of $y_1, \ldots, y_j$ are known at level j.

Before discussing the heuristic h proposed here, let us notice that g is the same marginal joint conditional probability that both the ϵ-A algorithm and the BS calculate to select the nodes to be expanded. Even more, ϵ-A with $\epsilon = 0$ is not only equivalent to the UC search, but also to A* with the constant heuristic $h = 1$. This heuristic is admissible too, since no probability is greater than 1. But, on the other hand, it is also the worst admissible heuristic, since any other admissible heuristic will dominate it[3] and consequently the A* algorithm using such an heuristic will never expand more nodes than the A* algorithm using $h = 1$.

Let us go now to discuss the heuristic proposed in this paper. Since our heuristic must not underestimate the marginal joint conditional probability $\mathbf{P}(y_{j+1}, \ldots, y_m \mid \boldsymbol{x}, y_1, \ldots, y_j)$ in order to be admissible, it is quite straightforward to pick the maximum value of such probability for obtaining an optimal heuristic h^*:

$$\begin{aligned}h^* = &\max_{(y_{j+1}, \ldots, y_m) \in \{0,1\}^{m-j}} \mathbf{P}(y_{j+1}, \ldots, y_m \mid \boldsymbol{x}, y_1, \ldots, y_j) \\ = &\max_{\substack{(y_{j+1}, \ldots, y_m) \\ \in \{0,1\}^{m-j}}} \prod_{i=j+1}^{m} \mathbf{P}(y_i \mid \boldsymbol{x}, y_1, \ldots, y_j, y_{j+1}, \ldots, y_{i-1}).\end{aligned} \tag{11}$$

However, obtaining such a maximum is, in fact, applying an ES over the set of labels $\mathcal{L} = \{\ell_{j+1}, \ldots, \ell_m\}$. Hence, this optimal heuristic is not computationally applicable. So, we need to obtain a tractable heuristic in exchange for renouncing such optimality. This leads to design an heuristic $\hat{h}$ also admissible, but less dominant than h^* ($h^* \prec \hat{h}$). For this purpose, let us consider $(\overline{y}_{j+1}, \ldots, \overline{y}_m)$ the values that define h^*, that is

$$h^* = \prod_{i=j+1}^{m} \mathbf{P}(y_i \mid \boldsymbol{x}, y_1, \ldots, y_j, \overline{y}_{j+1}, \ldots, \overline{y}_{i-1}), \tag{12}$$

[2] An heuristic h is admissible for our search problem if and only if it satisfies that $h^*(k) \leq h(k)$ for any node k, where $h^*(k)$ is the highest (and unknown) probability from the node k to a leaf.

[3] In our case, an heuristic h_1 dominates another heuristic h_2 (denoted by $h_1 \prec h_2$) if and only if it satisfies that $h_1(k) \leq h_2(k)$ for any node k.

and let us notice that values $(\overline{y}_{j+1}, \ldots, \overline{y}_m)$ which maximize the product do not have to maximize each individual term, then

$$\begin{aligned}&\mathbf{P}(y_i \mid \boldsymbol{x}, y_1, \ldots, y_j, \overline{y}_{j+1}, \ldots, \overline{y}_{i-1})\\ &\leq \max_{\substack{(y_{j+1},\ldots,y_{i-1})\\ \in\{0,1\}^{i-1-j}}} \mathbf{P}(y_i \mid \boldsymbol{x}, y_1, \ldots, y_j, y_{j+1}, \ldots, y_{i-1}).\end{aligned} \tag{13}$$

Hence, by defining $\hat{h}$ as

$$\hat{h} = \prod_{i=j+1}^{m} \max_{\substack{(y_{j+1},\ldots,y_{i-1})\\ \in\{0,1\}^{i-1-j}}} \mathbf{P}(y_i \mid \boldsymbol{x}, y_1, \ldots, y_j, y_{j+1}, \ldots, y_{i-1}). \tag{14}$$

it is easy to deduce that $\hat{h}$ is admissible and less dominant than h^* ($h^* \prec \hat{h}$). But again, $\hat{h}$ is not computationally applicable in general. However, this is not the case if we restrict ourselves to the case of linear models, for instance using *logistic regression*. Remember that $\mathbf{P}(y_i \mid \boldsymbol{x}, y_1, \ldots, y_{i-1})$ is estimated through the model $f_i(\boldsymbol{x}, y_1, \ldots, y_{i-1})$ using a sigmoid function to transform the output of $f_i(\boldsymbol{x}, y_1, \ldots, y_{i-1})$ into a probability. Particularly,

$$\mathbf{P}(y_i = 1 \mid \boldsymbol{x}, y_1, \ldots, y_{i-1}) = \frac{1}{1 + \exp^{-f_i(\boldsymbol{x}, y_1, \ldots, y_{i-1})}} \tag{15}$$

$$\mathbf{P}(y_i = 0 \mid \boldsymbol{x}, y_1, \ldots, y_{i-1}) = 1 - \frac{1}{1 + \exp^{-f_i(\boldsymbol{x}, y_1, \ldots, y_{i-1})}}. \tag{16}$$

In order to obtain the maximum value between $\mathbf{P}(y_i = 1 \mid \boldsymbol{x}, y_1, \ldots, y_{i-1})$ and $\mathbf{P}(y_i = 0 \mid \boldsymbol{x}, y_1, \ldots, y_{i-1})$, we need first to obtain the maximum value of both terms on their own. In this direction and according to the above expressions of both probabilities, $\mathbf{P}(y_i = 1 \mid \boldsymbol{x}, y_1, \ldots, y_{i-1})$ will be maximum if $f_i(\boldsymbol{x}, y_1, \ldots, y_{i-1})$ is the maximum and analogously $\mathbf{P}(y_i = 0 \mid \boldsymbol{x}, y_1, \ldots, y_{i-1})$ will be maximum when $f_i(\boldsymbol{x}, y_1, \ldots, y_{i-1})$ is the minimum. Hence, let us now focus on obtaining the maximum and the minimum of $f_i(\boldsymbol{x}, y_1, \ldots, y_{i-1})$.

From now on, let us consider that f_i is a linear model, then f_i adopts the following form

$$f_i(\boldsymbol{x}, y_1, \ldots, y_{i-1}) = \langle \boldsymbol{w}_x^i, \boldsymbol{x} \rangle + \langle \boldsymbol{w}_y^i, (y_1, \ldots, y_{i-1}) \rangle + \beta^i, \tag{17}$$

that splitting the second term in the known part (from ℓ_1 to ℓ_j) and unknown part (from ℓ_{j+1} to ℓ_i) it leads to

$$f_i(\boldsymbol{x}, y_1, \ldots, y_j, y_{j+1}, \ldots, y_{i-1}) = \langle \boldsymbol{w}_x^i, \boldsymbol{x} \rangle + \sum_{k=1}^{j} w_{y,k}^i y_k + \sum_{k=j+1}^{i-1} w_{y,k}^i y_k + \beta^i. \tag{18}$$

Since $\boldsymbol{x}$ is given and $y_1, \ldots, y_j$ are fixed, the second summation contains the variables for which the maximum and the minimum must be obtained. Let C_i be the constant part of f_i with regard obtaining the maximum and the minimum, that is,

$$C_i(\boldsymbol{x}, y_1, \ldots, y_j) = \langle \boldsymbol{w}_x^i, \boldsymbol{x} \rangle + \sum_{k=1}^{j} w_{y,k}^i y_k + \beta^i. \tag{19}$$

Then f_i can be rewritten as

$$f_i(\boldsymbol{x}, y_1, \ldots, y_j, y_{j+1}, \ldots, y_{i-1}) = C_i(\boldsymbol{x}, y_1, \ldots, y_j) + \sum_{k=j+1}^{i-1} w_{y,k}^i y_k. \tag{20}$$

Consequently, the function whose maximum must be obtained is

$$\sum_{k=j+1}^{i-1} w^i_{y,k} y_k = w^i_{y,j+1} y_{j+1} + \ldots + w^i_{y,i-1} y_{i-1}. \tag{21}$$

Let us now denote by $K^+_{i,j}$ and $K^-_{i,j}$ the positive and negative indexes of the coefficients $w^i_{y,k}$ with $j+1 \leq k \leq i-1$, that is,

$$\begin{aligned} K^+_{i,j} &= \{k \mid j+1 \leq k \leq i-1,\ w^i_{y,k} \geq 0\} \\ K^-_{i,j} &= \{k \mid j+1 \leq k \leq i-1,\ w^i_{y,k} < 0\}. \end{aligned} \tag{22}$$

Hence, (21) is maximum when y_k for $j+1 \leq k \leq i-1$ are

$$y_k = \begin{cases} 1 & \text{if } k \in K^+_{i,j} \\ 0 & \text{if } k \in K^-_{i,j} \end{cases} \tag{23}$$

and (21) is minimum when y_k for $j+1 \leq k \leq i-1$ are

$$y_k = \begin{cases} 1 & \text{if } k \in K^-_{i,j} \\ 0 & \text{if } k \in K^+_{i,j} \end{cases} \tag{24}$$

Hence,

(i) Let $y^{i,1}_{j+1}, \ldots, y^{i,1}_{i-1} \in \{0,1\}$ be the values which maximize (21), that is, the values that maximize $f_i(\boldsymbol{x}, y_1, \ldots, y_j, y_{j+1}, \ldots, y_{i-1})$ and hence, the values which makes $\mathbf{P}(y_i = 1 \mid \boldsymbol{x}, y_1, \ldots, y_j, y_{j+1}, \ldots, y_{i-1})$ be maximum and,

(ii) Let $y^{i,0}_{j+1}, \ldots, y^{i,0}_{i-1} \in \{0,1\}$ be the values that minimize (21), that is, the values that minimize $f_i(\boldsymbol{x}, y_1, \ldots, y_j, y_{j+1}, \ldots, y_{i-1})$ and hence, the values which makes $\mathbf{P}(y_i = 0 \mid \boldsymbol{x}, y_1, \ldots, y_j, y_{j+1}, \ldots, y_{i-1})$ be maximum.

Hence, $\max_{\substack{(y_{j+1},\ldots,y_{i-1}) \\ \in \{0,1\}^{i-1-j}}} \mathbf{P}(y_i \mid \boldsymbol{x}, y_1, \ldots, y_j, y_{j+1}, \ldots, y_{i-1})$ will be

$$\max_{v \in \{0,1\}} \{\mathbf{P}(y_i = v \mid \boldsymbol{x}, y_1, \ldots, y_j, y^{i,v}_{j+1}, \ldots, y^{i,v}_{i-1})\}. \tag{25}$$

Notice that $y^{i,1}_k = 1 - y^{i,0}_k$ for $j+1 \leq k \leq i-1$ and that according to the above definition of the sigmoid function, if $f_i(\boldsymbol{x}, y_1, \ldots, y_j, y^{i,1}_{j+1}, \ldots, y^{i,1}_{i-1}) \geq -f_i(\boldsymbol{x}, y_1, \ldots, y_j, y^{i,0}_{j+1}, \ldots, y^{i,0}_{i-1})$, then the maximum of $\mathbf{P}(y_i = v \mid \boldsymbol{x}, y_1, \ldots, y_j, y^{i,v}_{j+1}, \ldots, y^{i,v}_{i-1})$ is reached when $v = 1$ and otherwise when $v = 0$.

Therefore, the final expression for the heuristic will be

$$\hat{h} = \prod_{i=j+1}^{m} \max_{v \in \{0,1\}} \{\mathbf{P}(y_i = v \mid \boldsymbol{x}, y_1, \ldots, y_j, y^{i,v}_{j+1}, \ldots, y^{i,v}_{i-1})\}. \tag{26}$$

Remember that h^* and $\hat{h}$ only differ on the values of $y_{j+1}, \ldots, y_{i-1}$. In the former, the same values are common for all the factors of the product, whereas in the latter these values depend on each term i of the product, and hence, they can be different, which makes $\hat{h}$ not be an optimal heuristic. On the other hand, the cost of computing $\hat{h}$ is of polynomial order, unlike h^* which is of exponential order. Figure 5 exemplifies $\hat{h}$. Let us focus on the root node of the subtree in Fig. 5. The marginal joint conditional probability until this node is $g = 0.6$. For computing $\hat{h}$, first we evaluate the maximum marginal conditional probabilities

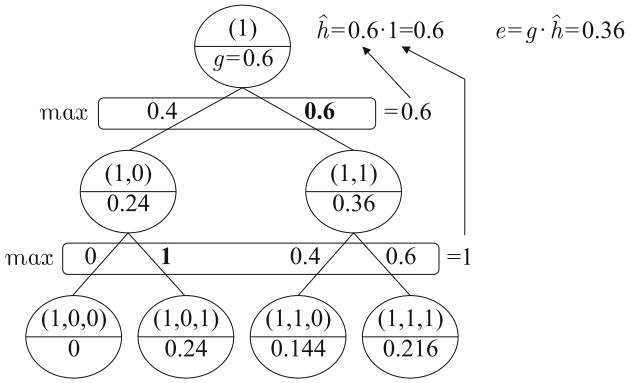


Fig. 5 An example of the computation of heuristic $\hat{h}$

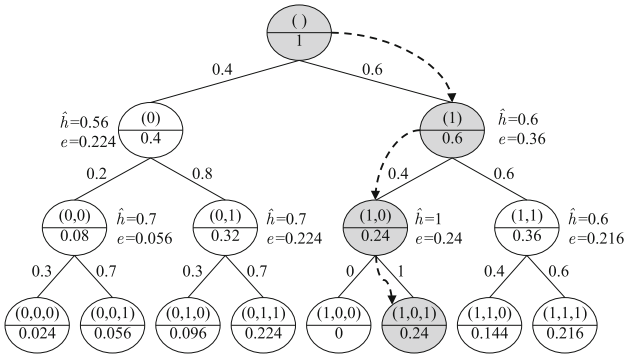


Fig. 6 An example of A* using the heuristic $\hat{h}$. The *dotted arrows* show the path followed by the algorithm. The values of g are provided inside each node

$P(y_2 \mid \boldsymbol{x}, y_1 = 1)$ provided by $f_2(\boldsymbol{x}, y_1)$ and $P(y_3 \mid \boldsymbol{x}, y_1 = 1, y_2)$ provided by $f_3(\boldsymbol{x}, y_1, y_2)$ which respectively are 0.6 and 1.0. Then we carry out their product by applying the product rule of the probability to estimate the marginal joint conditional probability from that node to a leaf. Notice that the maximum at each level does not correspond to the same branch of the tree. In other words, the maximum in the first level corresponds to $y_2 = 1$, whereas the maximum in the second level is obtained by $P(y_3 \mid \boldsymbol{x}, y_1 = 1, y_2 = 0)$ when $y_3 = 1$ fixing $y_2 = 0$. This is what makes the heuristic $\hat{h}$ not to be the optimal h^*.

Figure 6 shows the path followed by A* using $\hat{h}$. It reaches the optimal leaf as is theoretically expected. Comparing this graph with the one in Fig. 3a that illustrates ϵ-A with $\epsilon = 0$, and taking into account the properties of the heuristics related to the dominance, ϵ-A, $\epsilon = 0$ (equivalent to A* using $h = 1$ or UC search) explores more nodes than A* using $\hat{h}$.

As a final remark, the A* algorithm using $\hat{h}$ perfectly estimates the risk minimizer for the subset 0/1 loss, as both ϵ-A with $\epsilon = 0$ and ES do it. Even more, A* with $\hat{h}$ expands equal or fewer nodes, since $\hat{h}$ is more dominant than the heuristic $h = 1$ ($\hat{h} \prec h = 1$). Obviously, computing $\hat{h}$ is more costly than computing $h = 1$ or applying just a UC search. The question is then if this additional computing time compensates the theoretical guarantee it has of expanding fewer nodes.

4.2 A general admissible heuristic for trading off the number of nodes explored and its computing time

The previous section pointed out that both $\hat{h}$ and $h = 1$ are admissible heuristics. Besides, using the former theoretically guarantees that the A* algorithm explores fewer nodes than using the latter. But the cost of computing $\hat{h}$ could be high in comparison to just considering a constant heuristic $h = 1$. This trade-off sheds light on including a parameter d for limiting the depth of the heuristic. Hence, for a node of level j and a value of d with $0 \leq d \leq m - j$, only the terms $\mathbf{P}(y_i \mid \boldsymbol{x}, y_1, \ldots, y_j, y^i_{j+1}, \ldots, y^i_{i-1})$ of $\hat{h}$ are evaluated using f_i for nodes from level $j+1$ to level $j+k$ whereas these terms are estimated by the constant 1 for nodes from level $j+k+1$ to level m, that is,

$$\hat{h}^d = \prod_{i=j+1}^{j+d} \mathbf{P}(y_i \mid \boldsymbol{x}, y_1, \ldots, y_j, y^i_{j+1}, \ldots, y^i_{i-1}) \cdot \prod_{i=j+d+1}^{m} 1, \tag{27}$$

or equivalently

$$\hat{h}^d = \prod_{i=j+1}^{j+d} \mathbf{P}(y_i \mid \boldsymbol{x}, y_1, \ldots, y_j, y^i_{j+1}, \ldots, y^i_{i-1}). \tag{28}$$

It is clear that $\hat{h}$ is more dominant than $\hat{h}^d$ ($\hat{h} \prec \hat{h}^d$) and it continues being admissible. In turn, $\hat{h}^d$ is more dominant than $h = 1$ ($\hat{h}^d \prec h = 1$). Hence, it is expected that using the heuristic $\hat{h}^d$ with $0 \leq d \leq m - j$ and $1 \leq j \leq m - 1$, the A* algorithm explores more number or equal number of nodes than $\hat{h}$, but less number or equal number of nodes than $h = 1$. In fact, $\hat{h}^d$ encapsulates both heuristics, since taking the extreme values of d leads to them. On the one hand, taking the maximum value of d leads to $\hat{h}^{m-j}$ with $1 \leq j \leq m-1$ (we will denote this heuristic as $\hat{h}^\infty$), which is in fact the heuristic $\hat{h}$ detailed in Sect. 4.1. On the other hand, taking the minimum value of d leads to $\hat{h}^0$ which is indeed the heuristic $h = 1$. In general, $\hat{h}^{d_1}$ is more dominant than $\hat{h}^{d_2}$ if $d_1 > d_2$. However, the computational time of obtaining the values of $\hat{h}^d$ increases as d increases. Hence, tuning d adequately one can obtain a balance between the number of nodes explored and the computational time employed to evaluate the heuristic.

The case of $d = 1$ has especial interest, since the heuristic is also valid for non-linear models f_j. The form of the heuristic is

$$\hat{h}^1 = \prod_{i=j+1}^{j+1} \mathbf{P}(y_i \mid \boldsymbol{x}, y_1, \ldots, y_j, y^i_{j+1}, \ldots, y^i_{i-1}) \cdot \prod_{i=j+2}^{m} 1, \tag{29}$$

that simplifying leads to

$$\hat{h}^1 = \mathbf{P}(y_{j+1} \mid \boldsymbol{x}, y_1, \ldots, y_j). \tag{30}$$

As seen, this heuristic $\hat{h}^1$ means just evaluating the model of the node $f_j(\boldsymbol{x}, y_1, \ldots, y_j)$, hence without making any restriction on the linearity of f_j.

Figure 7 shows an example of the path followed by the A* algorithm when $\hat{h}^1$ is taken as an heuristic. As seen, the A* algorithm explores more nodes using this heuristic than using $\hat{h}$, but the computational time of evaluating the heuristic is considerably reduced, as we will show later on in the experiments.

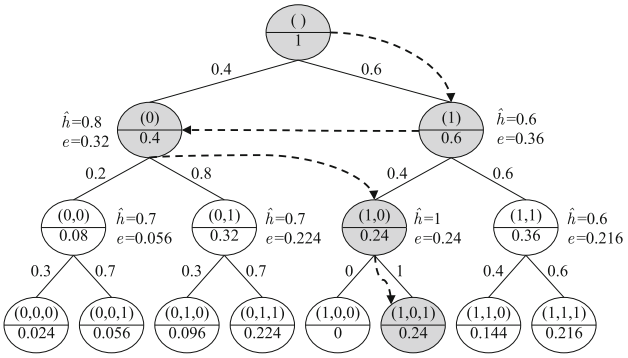


Fig. 7 An example of A* using the heuristic $\hat{h}^1$. The *dotted arrows* show the path followed by the algorithm. The values of g are provided inside each node

4.3 Implementation details

Here we significantly extend the results of A* using $\hat{h}$ presented in the preliminary work (Mena et al. 2015). However, they present differences because we have carried out an improvement in the implementation of the algorithms in order to become more efficient. In fact, the time results reported there in comparison with the number of nodes explored was the key to make us reconsider the implementation of the algorithms. Algorithm 1 shows the pseudocode of A* using $\hat{h}^d$.

Particularly, the differences between this version and that of the preliminary work (Mena et al. 2015) are that now:

1. The models f_i (parameters $\boldsymbol{w}_x^i$, $\boldsymbol{w}_y^i$ and $\boldsymbol{\beta}^i$) are ordered according to the label order of the chain.
2. All repeated computations are calculated just once and stored:
   (a) The evaluation of $\langle \boldsymbol{w}_x^i, \boldsymbol{x} \rangle$ for each linear model f_i, which is common for all nodes corresponding to f_i, is computed once and saved in variable WX (line 2).
   (b) Sets K^+ and K^-, that is the best values of y_k for the unknown part of the heuristic, are computed only once before starting the main loop of A* (lines 3–8).
3. Open set, Q, stores tuples of four elements: label combination, level, e and g. Q is stored in a resizable vector whose positions are reused. The left child is stored in the position of its parent (line 21). Notice that parent information is not required to obtain the final solution, since Q contains all the required information for each node.
4. The design of Q allows us to optimize the function Max, the operation to obtain the best node to be expanded. The function Max only needs to evaluate the first elements $(1 \ldots last)$ of Q (line 13), because the rest of the vector is unused.
5. Algorithm 1 contains several auxiliary functions to make the pseudocode shorter and more easily understood. However, in the actual program all these functions were coded inline, making the code faster because all the calls to functions with large parameters, like the function *Heuristic*, are avoided.

4.4 Complexity analysis

Despite the theoretical optimality properties of the A* algorithm, it might not be useful in some problems because it may explore an exponential number of nodes with regard to

Algorithm 1 Pseudocode of the implementation of A* algorithm using $\hat{h}^d$

function A*
Input: $\boldsymbol{x}$, $[\mathbf{W}, \boldsymbol{\beta}]$ a CC Linear Model, m, d, $BlockSize$
Output: Label combination with highest probability for $\boldsymbol{x}$
  $WX \leftarrow \mathbf{W}_x * \boldsymbol{x}$ // Computes $\langle \boldsymbol{w}_x, \boldsymbol{x} \rangle$ for all labels
  $K^+ \leftarrow AllocMemory(m)$
  $K^- \leftarrow AllocMemory(m)$
  for $label = [2 : m]$ **do** // Starts at 2nd level, no label attributes in the 1st model
    $K^+[label] \leftarrow [\![\mathbf{W}_y[label] >= 0]\!]$
    $K^-[label] \leftarrow [\![\mathbf{W}_y[label] < 0]\!]$
  end for
  $Q \leftarrow AllocMemory(BlockSize)$ // tuples $\{Labels, Level, e, g\}$
  $Q[1] \leftarrow \{[\,], 0, 1, 1\}$ // root node, empty label set, level 0, e and $g = 1$
  $last \leftarrow 1$ // last element used in Q
  while true **do**
    $[Best, Position] \leftarrow Max(Q, last)$
    if $Best.Level = m$ **then**
      return $Best.Labels$ // Leaf node
    end if
    $level \leftarrow Best.Level + 1$
    $P \leftarrow 1/(1 + exp(-(WX[level] + \beta[level] + \langle \boldsymbol{w}_y[level], Best.Labels \rangle)))$
    // Left child
    $\hat{h}^d \leftarrow Heuristic(d, level, [Best.Labels\ 0], K^+, K^-, WX, \mathbf{W}_y, \boldsymbol{x})$ // Eq(28)
    $Q[Position] \leftarrow \{[Best.Labels\ 0], level, Best.g * (1 - P) * \hat{h}^d, Best.g * (1 - P)\}$
    // Right child
    $last \leftarrow last + 1$
    if $last > Q.size$ **then**
      $Q \leftarrow resize(Q, BlockSize)$
    end if
    $\hat{h}^d \leftarrow Heuristic(d, level, [Best.Labels\ 1], K^+, K^-, WX, \mathbf{W}_y, \boldsymbol{x})$ // Eq(28)
    $Q[last] \leftarrow \{[Best.Labels\ 1], level, Best.g * P * \hat{h}^d, Best.g * P\}$
  end while
end function

the depth of the tree [see Pearl (1984) and Russell and Norvig (2003)]. Only under certain conditions over the heuristic taken, is it possible to obtain an algorithm with a theoretical complexity less than exponential. This is the case if one is able to prove that the heuristic h satisfies the following condition for any level j with regard to the optimal heuristic h^* (Russell and Norvig 2003)

$$|h^*(j) - h(j)| \leq \mathcal{O}(\log h^*(j)). \tag{31}$$

In general, it occurs that that error is at least linear with regard to the path cost, hence leading to exponential growth. However, by taking care of designing good heuristics, one can obtain huge profit in relation to not providing any kind of heuristic information. That is the case with our heuristic $\hat{h}^d$ with regard to the heuristic $h = 1$, which does not provide any kind of heuristic information, in spite of being an improvement of the ES.

On what follows, let us focus on the heuristic $\hat{h}^\infty$, since it is the most dominant heuristic among the heuristics proposed. According to the results where the noise is increasing (see Figs. 9, 10 of Sect. 5.3), it is not clear that $\hat{h}^\infty$ is exponential at all as the rest of the algorithms clearly show, at least for the datasets taken. This behavior of $\hat{h}^\infty$ sheds light on performing a deeper analysis over other situations. In this sense, let us consider a theoretical case when the difference between $\hat{h}^\infty$ and h^* is maximum. For instance, this situation occurs (see Fig. 8) when the first level has probability 0 in the left branch and 1 in the right branch, all the

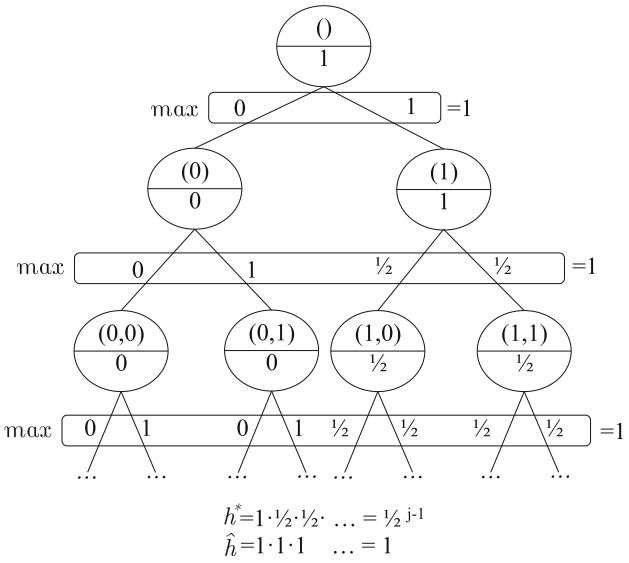


Fig. 8 An example of worst cases for heuristic $\hat{h}$

probabilities of the left subtree are 0 (for the left branches) and 1 (for the right branches) and all the probabilities of the right subtree are 1/2 (for all the branches). In this case, the error can be bounded as follows for any level j

$$|h^*(j) - \hat{h}^{\infty}(j)| \leq |1/2^{j-1} - 1|. \tag{32}$$

This bound tends to 1 as the level j grows, hence, in this case, it is not possible to guarantee that the theoretical complexity is less than exponential. However, these extreme cases are hardly likely to occur. Let us remember that the probabilities are evaluated from the models, where the description $\boldsymbol{x}$ of the examples plays an important role. This description is equal for obtaining the probabilities of all the nodes of the same level, hence, such probabilities will probably not differ so much among them as in the extreme case considered.

5 Experiments

This section deals with the experiments carried out with all the approaches. Before discussing the experimental results in Sects. 5.2 and 5.3, let us describe in Sect. 5.1 the common settings of all experiments: datasets, learning algorithms and parameter selection procedures. Finally, Sect. 5.3 reports the computational results of the methods for more complex problems, such as those that include noise.

5.1 Settings

The experiments were performed over several benchmark multi-label datasets whose main properties are shown in Table 1. As can be seen, there are significant differences in the number of attributes, instances and labels. The cardinality—number of labels per instance—varies from 1.07 to 4.27. Concerning the number of labels, there are some datasets with just 5, 6 or 7 labels, whereas others have more than 100, one of them even has almost 400 labels.

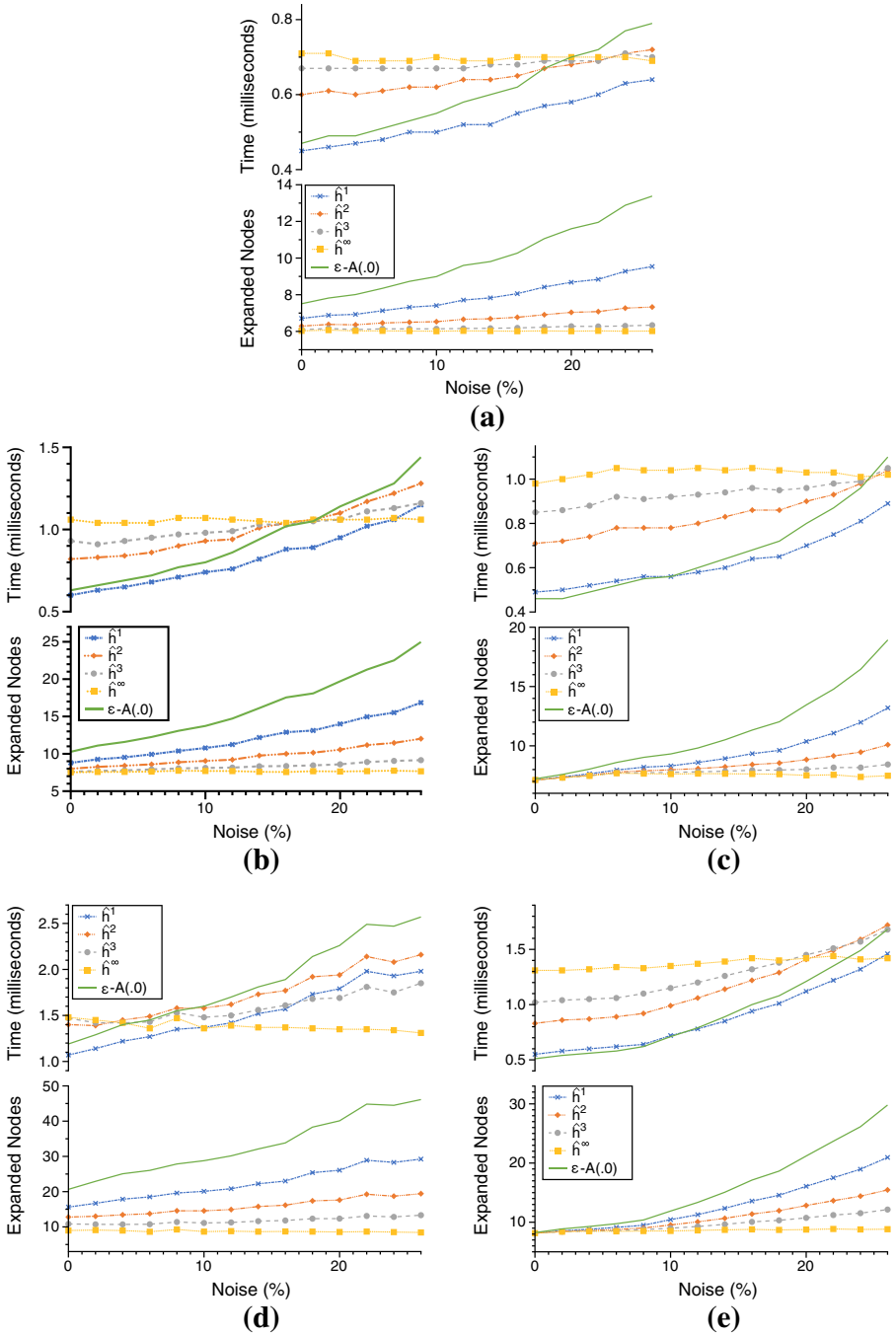


Fig. 9 Number of nodes expanded and computational time employed (ms) for the algorithm A* with the heuristic $\hat{h}$ for different values of the parameter d $(1, 2, 3, \infty)$ and for the ϵ-A with $\epsilon = 0$, for different percentages of noise and for datasets with few labels (from 5 to 7). **a** Image (5), **b** emotions (6), **c** scene (6), **d** flags (7) and **e** reuters (7)

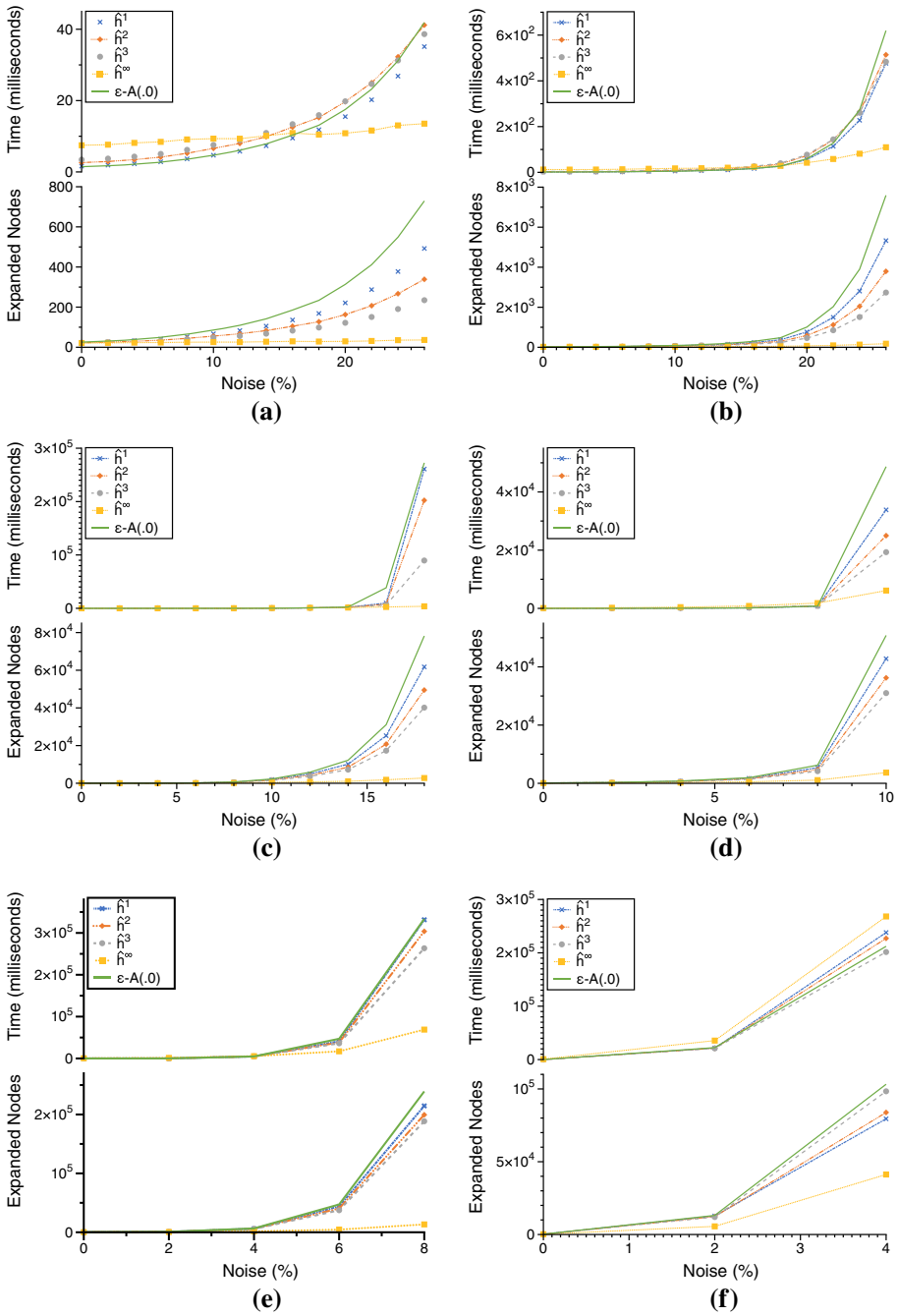


Fig. 10 Number of nodes expanded and computational time employed (ms) for the algorithm A* with the heuristic $\hat{h}$ for different values of the parameter d $(1, 2, 3, \infty)$ and for the ϵ-A with $\epsilon = 0$, for different percentages of noise and for datasets with more labels (from 14 to 159). **a** Yeast (14), **b** slashdot (22), **c** medical (45), **d** enron (53), **e** mediamill (101) and **f** bibtex (159)

Table 1 Properties of the datasets

Datasets	Instances	Attributes	Labels	Cardinality
Bibtex	7395	1836	159	2.40
Corel5k	5000	499	374	3.52
Emotions	593	72	6	1.87
Enron	1702	1001	53	3.38
Flags	194	19	7	3.39
Image	2000	135	5	1.24
Mediamill*	5000	120	101	4.27
Medical	978	1449	45	1.25
Reuters	7119	243	7	1.24
Scene	2407	294	6	1.07
Slashdot	3782	1079	22	1.18
Yeast	2417	103	14	4.24

The approaches for inference in PCC compared with our proposal were those discussed throughout the paper, except for the ES. No experiment was carried out with the ES method due to its computational cost. Hence, the methods compared with the A* algorithm with the heuristic $\hat{h}$ for different values of the parameter d $(1, 2, 3, \infty)$ were GS, ϵ-A algorithm for different values of ϵ $(.0, .25, .5)$ and BS for different values of beam width b $(1, 2, 3, 10)$. Let us remember that the ϵ-A algorithm with $\epsilon = 0.5$ is equivalent to GS and to BS with $b = 1$.

The results will be presented in terms of the example-based subset 0/1 loss estimated by means of a 10-fold cross-validation.

The base learner employed to obtain the binary classifiers that compose all these multi-label models was *logistic regression* (Lin et al. 2008) with probabilistic output. The regularization parameter C was established for each *individual* binary classifier performing a grid search over the values $C \in \{10^{-3}, 10^{-2}, \ldots, 10^{3}\}$ optimizing the brier loss estimated by means of a balanced 2-fold cross validation repeated 5 times. The brier loss (Brier 1950) is a proper score that measures the accuracy of probabilistic predictions, as *logistic regression* does. The expression is as follows

$$\frac{1}{n}\sum_{i=1}^{n}(\hat{p}_i - a_i)^2, \tag{33}$$

where for an instance i, p_i is the predicted probability of a certain label and a_i is the actual value of the label (0 or 1).

5.2 Results over benchmark datasets

Tables 2, 3 and 4 respectively show the subset 0/1 loss, the number of nodes explored and the computational time (in seconds) averaged per test instances for the different methods compared.

Before discussing the results of the tables, let us remember that only the A* algorithm using $\hat{h}^d$ (and consequently also the ϵ-A with $\epsilon = 0$, since it is the A* algorithm with $h = 1$ or $\hat{h}^0$) provides a Bayes optimal inference, as the ES does. This means that they always predict

Table 2 Subset 0/1 loss for the different methods

Datasets	$\hat{h}$ (*) ϵ-A(.0)	ϵ-A(.25)	GS/BS(1) ϵ-A(.5)	BS(2)	BS(3)	BS(10)
Bibtex	**81.92**	81.95	82.19	**81.88**	**81.92**	**81.92**
Corel5k	**97.48**	98.62	98.90	98.30	98.04	**97.48**
Emotions	**71.16**	71.82	72.83	72.16	71.32	**71.16**
Enron	**83.14**	84.26	85.43	83.43	83.37	**83.14**
Flags	**87.13**	87.16	**86.13**	88.21	**87.13**	**87.13**
Image	**68.35**	**68.35**	69.75	**68.35**	**68.35**	**68.35**
Mediamill*	**83.86**	84.58	85.80	84.10	**83.86**	**83.86**
Medical	**30.37**	**30.37**	30.67	**30.37**	**30.37**	**30.37**
Reuters	**22.73**	**22.70**	23.60	**22.69**	**22.73**	**22.73**
Scene	**31.86**	**31.86**	33.28	31.90	**31.86**	**31.86**
Slashdot	**51.80**	52.22	54.49	**51.77**	**51.80**	**51.80**
Yeast	**76.95**	77.62	79.77	**76.83**	77.08	**76.95**

Those scores that are equal to or better than optimal predictions reached by ϵ-A and ES are shown in bold

the label combination with the highest joint conditional probability. Despite the fact that other methods may predict other label combinations with lower joint conditional probability for some examples, in a few cases they obtain better subset 0/1 scores. This fact is due to several reasons, mainly (a) the relatively small size of testing sets, and (b) that the models f_j obtained to estimate the joint conditional probability $P(\boldsymbol{y} \mid \boldsymbol{x})$ do not usually return true estimations. Theoretically, under perfect conditions (large test sets and perfect models), the A* algorithm using $\hat{h}^d$ would obtain the best scores. In general, the performance of the ϵ-A algorithm decreases as the value of ϵ increases and the performance of the BS method increases as b increases. The BS method reaches stability for low values of the beam b, it even converges to the performance of the A* algorithm but at the cost of exploring many more nodes.

With regard to the number of nodes explored (see Table 3), GS (equivalent to ϵ-A algorithm with $\epsilon = 0.5$ and to BS(1)) is the method which explores the least number of nodes, since it only goes over one path in the tree. In fact, such a number corresponds to the number of labels (plus one if the root is considered as an explored node). It follows the A* algorithm using our heuristic $\hat{h}^d$ for any value of d, although it is exceeded by the ϵ-A algorithm with $\epsilon > 0$ and by BS with $b > 1$ in some cases. However, let us remember that neither GS nor the ϵ-A algorithm with $\epsilon > 0$ nor BS with any value of the beam b guarantee reaching an optimal solution as the A* algorithm using heuristic $\hat{h}^d$ does. So, it is not surprising that they explore fewer nodes. What it is actually surprising is that for most of the cases they explore more nodes even without reaching an optimal solution.

Let us now focus on the methods that theoretically reach the optimum (the A* algorithm with the heuristic $\hat{h}^d$ or the ϵ-A algorithm with $\epsilon = 0$). As it has theoretically been shown, the A* algorithm with the heuristic $\hat{h}^\infty$ explores the least number of nodes, followed by the same algorithm but with the heuristic $\hat{h}^3$, then with the heuristic $\hat{h}^2$, after that with the heuristic $\hat{h}^1$ and finally the ϵ-A algorithm with $\epsilon = 0$.

The computational time is expected to be higher as more nodes are explored, but looking at Table 4 one can see that this does not happen at all. This is true for GS, the ϵ-A algorithm and the BS technique, but the A* algorithm obtains higher computational time in spite of exploring considerably fewer nodes. The reason for that is the time spent in computing the

Table 3 Number of explored nodes for the different methods

Datasets	$\hat{h}^1$	$\hat{h}^2$	$\hat{h}^3$	$\hat{h}^\infty$	$\epsilon-$A(.0)	$\epsilon-$A(.25)	GS/BS(1) $\epsilon-$A(.5)	BS(2)	BS(3)	BS(10)
Bibtex	283.31	277.97	273.23	215.91	289.27	184.00	160.00	319.00	477.00	1575.00
Corel5k	1456.92	1443.08	1431.57	1338.62	1474.17	517.11	375.00	749.00	1122.00	3725.00
Emotions	9.12	8.28	7.80	7.68	10.67	10.78	7.00	13.00	18.00	45.00
Enron	110.32	106.50	103.28	75.51	114.81	77.29	54.00	107.00	159.00	515.00
Flags	16.45	12.69	10.43	8.79	22.56	16.33	8.00	15.00	21.00	55.00
Image	6.64	6.28	6.14	6.11	7.33	7.66	6.00	11.00	15.00	35.00
Mediamill*	188.18	185.64	183.68	178.89	191.76	142.37	102.00	203.00	303.00	995.00
Medical	46.61	46.58	46.55	46.40	46.64	46.65	46.00	91.00	135.00	435.00
Reuters	8.15	8.14	8.13	8.13	8.24	8.26	8.00	15.00	21.00	55.00
Scene	7.16	7.15	7.15	7.15	7.25	7.25	7.00	13.00	18.00	45.00
Slashdot	24.98	24.88	24.86	24.84	25.29	24.89	23.00	45.00	66.00	205.00
Yeast	23.60	22.80	22.38	21.01	26.09	26.02	15.00	29.00	42.00	125.00

Table 4 Average prediction time (in seconds) per example for all methods

Datasets	$\hat{h}^1$	$\hat{h}^2$	$\hat{h}^3$	$\hat{h}^\infty$	$\epsilon-$A(.0)	$\epsilon-$A(.25)	GS/BS(1) $\epsilon-$A(.5)	BS(2)	BS(3)	BS(10)
Bibtex	0.0195	0.0314	0.0411	0.8407	0.0162	0.0099	0.0079	0.0110	0.0156	0.0507
Corel5k	0.1684	0.2578	0.3333	23.8344	0.1360	0.0161	0.0110	0.0278	0.0404	0.1361
Emotions	0.0006	0.0008	0.0009	0.0011	0.0006	0.0006	0.0004	0.0005	0.0006	0.0013
Enron	0.0085	0.0141	0.0185	0.1084	0.0072	0.0030	0.0019	0.0039	0.0055	0.0172
Flags	0.0011	0.0014	0.0015	0.0015	0.0012	0.0007	0.0004	0.0005	0.0007	0.0016
Image	0.0005	0.0006	0.0007	0.0007	0.0005	0.0005	0.0004	0.0004	0.0005	0.0009
Mediamill*	0.0144	0.0242	0.0326	0.4847	0.0115	0.0055	0.0037	0.0072	0.0105	0.0333
Medical	0.0033	0.0057	0.0077	0.0495	0.0027	0.0027	0.0027	0.0033	0.0046	0.0144
Reuters	0.0006	0.0008	0.0010	0.0013	0.0005	0.0005	0.0005	0.0005	0.0007	0.0016
Scene	0.0005	0.0007	0.0009	0.0010	0.0005	0.0005	0.0004	0.0005	0.0006	0.0013
Slashdot	0.0018	0.0029	0.0039	0.0124	0.0015	0.0014	0.0012	0.0016	0.0022	0.0063
Yeast	0.0016	0.0026	0.0034	0.0075	0.0015	0.0010	0.0006	0.0010	0.0014	0.0040

heuristic. In this respect, the A* algorithm is faster as the parameter d diminishes, although this implies exploring more nodes. Hence, considering the methods that reach the optimum, the ϵ-A approximation algorithm with $\epsilon = 0$ is the fastest method, followed by A* using the heuristic $\hat{h}^1$, then $\hat{h}^2$ and so on until ending with the heuristic $\hat{h}^\infty$.

As a conclusion, the A* algorithm with the heuristic h^1 can be considered a good alternative for guaranteeing optimal performance in terms of subset 0/1 and balancing the trade-off between the number of nodes explored and the computational time. Besides, it is also applicable for non-linear models.

5.3 Results with noise

We have also performed experiments including noise in the datasets in order to analyze more in depth the power of the A* algorithm for inference in PCCs. This study arises from the fact that the number of nodes explored by A* is quite low in comparison with the number of labels (depth of the tree). This is so even using the heuristic $h = 1$ (ϵ-A with $\epsilon = 0$) which does not provide information as other heuristics do, for instance, $\hat{h}$. This means that the algorithm A* performs few backtracking in the tree, hence, the probabilities provided by the models may be not so close from 0.5 or, even more, they may be close to 0 or 1. In this way, by adding noise we expect the probabilities to be closer to 0.5, making the problem more complex where more informative heuristics can show their strength.

Certain grades of noise in terms of percentage were added to the datasets. No kinds of noise were included in the description of the examples, that is, in the $\boldsymbol{x}$ part, otherwise, the noise was only introduced in the labels of the examples, that is, in the $\boldsymbol{y}$ part. In this sense, a percentage of the values of the labels was swapped from being relevant to become irrelevant and vice versa for the whole dataset. This means that for a given example the relevance of either all their labels or only some of them or even none of them was changed. The percentage of noise included ranges from 0 to 25 % for datasets with fewer than 22 labels. However, the experiments did not finish in a prudential time by using all the percentages of this range for datasets with more than 45 labels. In particular, the maximum percentage considered in this respect for medical (45 labels), enron (53 labels), mediamill (101 labels) and bibtex (159 labels) respectively was 18, 10, 8 and 4 %.

Figures 9 and 10, respectively show for datasets with few labels (from 5 to 7) and for datasets with more labels (from 14 to 159), the number of nodes expanded and the computational time employed by the A* algorithm with the heuristic $\hat{h}$ for different values of the parameter d (1, 2, 3, ∞) and by the ϵ-A with $\epsilon = 0$, all of them for different percentages of noise.

Obviously, both the number of nodes explored and the computational time increases as the percentage of noise increases. Regarding the number of nodes explored, it decreases as d increases as theoretically expected. In this respect, the number of nodes expanded using the heuristic $\hat{h}^\infty$ keeps quite steady as the noise increases in comparison to using the rest of the heuristics. In fact, by using these other heuristics, the number of nodes rockets from a certain percentage of noise. This is so in general, but it especially happens for datasets with more than 14 labels.

Concerning the computational time and for datasets with few labels, A* using $\hat{h}^d$ from $d > 1$ is steadier than using $\hat{h}^1$ or ϵ-A with $\epsilon = 0$ as the noise increases. In fact, these two last approaches are the best options, $\hat{h}^1$ being clearly better than ϵ-A with $\epsilon = 0$. However, for high percentages of noise the other options begin to show their strength. Only flags among the datasets with few labels presents a different behavior. In this case, the behavior of the computational time is similar to that of the datasets with more labels, which coincides with

the behavior of the number of nodes. So, the computational time increases as d decreases and $\hat{h}^{\infty}$ becomes clearly the best option. In any case, we are measuring the computational time in milliseconds, so the worth of the heuristics becomes crucial for datasets with more than 22 labels.

6 Conclusions

This paper proposes a family of admissible heuristics for the A* algorithm for performing inference in PCCs. In this way, providing admissible heuristics leads obtaining optimal predictions in terms of subset 0/1 loss and exploring fewer nodes than previous approaches that also produce optimal predictions. The family of heuristics ($\hat{h}^d$) is defined by a parameter d which determines the depth in the tree for evaluating the heuristic and, hence, controls the trade-off between the number of nodes explored and the computational time of the algorithm due to the evaluation of the heuristic. In this manner, the algorithm A*($\hat{h}^d$) explores fewer nodes but it expends more time in reaching a solution as d increases. So far, only uniform-cost search (or ϵ-A($\epsilon = .0$), which in turn is the particular case of the A*($\hat{h}^0$)) has been shown to provide optimal subset 0/1 loss, unlike other approaches such as GS, ϵ-A($\epsilon > 0$) or BS that only estimate it. The algorithm A*($\hat{h}^d$) with $d > 1$ is limited to linear models, but this is not a major drawback because it is quite common to employ linear models. This is usually the case for MLC problems with many labels in which the examples of some classes may be scarce, since using non-linear models in such problems can lead to overfitting. Only the particular case of $d = 1$ is also suitable for non-linear models, since it just involves evaluating the models from all the known values of the labels.

In the experiments performed over the benchmark datasets, the algorithm A*($\hat{h}^1$) or uniform-cost search (or ϵ-A($\epsilon = .0$) or A*($\hat{h}^0$)) are better choices than A*($\hat{h}^d$) for $d > 1$, since the computational cost of evaluating the heuristic does not seem to compensate for the reduction in nodes explored. In this respect, adding noise to the benchmark datasets, and, hence, obtaining more complex problems, allows us to show the strength of A*($\hat{h}^d$) for $d > 1$. In this sense, the behavior of A*($\hat{h}^{\infty}$) is quite steady as the percentage of noise increases, unlike A*($\hat{h}^d$) for the rest of the values of d, including $d = 0$ (ϵ-A($\epsilon = .0$)). Furthermore, only for small datasets in a number of labels, $\hat{h}^1$ becomes the best alternative.

In our opinion, the heuristic search is a promising line of research for inference in PCCs. As future work, new admissible heuristics can be devised, keeping in mind that we should look for a trade-off between their computational complexity and the quality of the estimations.

Acknowledgments This research has been supported by the Spanish Ministerio de Economía y Competitividad (Grants TIN2011-23558, TIN2015-65069).

References

Brier, G. W. (1950). Verification of forecasts expressed in terms of probability. *Monthly Weather Review*, *78*(1), 1–3.

Cheng, W., & Hüllermeier, E. (2009). Combining instance-based learning and logistic regression for multilabel classification. *Machine Learning*, *76*(2–3), 211–225.

Dembczyński, K., Cheng, W., & Hüllermeier, E. (2010). Bayes optimal multilabel classification via probabilistic classifier chains. *ICML*, *2010*, 279–286.

Dembczyński, K., Waegeman, W., Cheng, W., & Hüllermeier, E. (2012). On label dependence and loss minimization in multi-label classification. *Machine Learning*, *88*(1–2), 5–45.

Dembczynski, K., Waegeman, W., & Hllermeier, E. (2012). An analysis of chaining in multi-label classification. In L. D. Raedt, C. Bessire, D. Dubois, P. Doherty, P. Frasconi, F. Heintz, & P. J. F. Lucas (Eds.), *ECAI, frontiers in artificial intelligence and applications* (Vol. 242, pp. 294–299). Amsterdam: IOS Press.

Elisseeff, A., & Weston, J. (2005). A kernel method for multi-labelled classification. In: *ACM conference on research and development in information retrieval* (pp. 274–281).

Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, *4*(2), 100–107.

Kumar, A., Vembu, S., Menon, A. K., & Elkan, C. (2012). Learning and inference in probabilistic classifier chains with beam search. *ECML/PKDD*, *2012*, 665–680.

Kumar, A., Vembu, S., Menon, A. K., & Elkan, C. (2013). Beam search algorithms for multilabel learning. *Machine Learning*, *92*(1), 65–89.

Lin, C. J., Weng, R. C., & Keerthi, S. S. (2008). Trust region Newton method for logistic regression. *Journal of Machine Learning Research*, *9*(Apr), 627–650.

McCallum, A. K. (1999). Multi-label text classification with a mixture model trained by EM. In: *AAAI 99 workshop on text learning*.

Mena, D., Montañés, E., Quevedo, J. R., & del Coz, J. J. (2015). *An overview of inference methods in probabilistic classifier chains for multi-label classification*. Technical Report: Artificial Intelligence Center, University of Oviedo, Spain, Campus de Viesques, Gijón.

Mena, D., Montañés, E., Quevedo, J. R., & del Coz, J. J. (2015). Using a* for inference in probabilistic classifier chains. In: *IJCAI 2015, Proceedings of the 20th international joint conference on artificial intelligence* (pp. 3707–3713), Buenos Aires, Argentina, July 25–31.

Montañés, E., Quevedo, J. R., & del Coz, J. J. (2011). Aggregating independent and dependent models to learn multi-label classifiers. In: *ECML/PKDD'11—Volume Part II* (pp. 484–500). Springer-Verlag.

Montañés, E., Senge, R., Barranquero, J., Quevedo, J. R., del Coz, J. J., & Hüllermeier, E. (2014). Dependent binary relevance models for multi-label classification. *Pattern Recognition*, *47*(3), 1494–1508. Handwriting Recognition and other PR Applications

Pearl, J. (1984). *Heuristics: Intelligent search strategies for computer problem solving*. Boston, MA: Addison-Wesley Longman Publishing Co. Inc.

Read, J., Martino, L., & Luengo, D. (2014). Efficient monte carlo methods for multi-dimensional learning with classifier chains. *Pattern Recognition*, *47*(3), 1535–1546.

Read, J., Pfahringer, B., Holmes, G., & Frank, E. (2009). Classifier chains for multi-label classification. In: *ECML/PKDD'09, LNCS* (pp. 254–269). Springer

Read, J., Pfahringer, B., Holmes, G., & Frank, E. (2011). Classifier chains for multi-label classification. *Machine Learning*, *85*(3), 333–359.

Russell, S. J., & Norvig, P. (2003). *Artificial intelligence: A modern approach* (2nd ed.). New York: Pearson Education.

Senge, R., Barranquero, J., del Coz, J. J., & Hüllermeier, E. (2012a). *Rectifying classifier chains for multi-label classification*. Technical Report.

Senge, R., del Coz, J. J., & Hüllermeier, E. (2012b). On the problem of error propagation in classifier chains for multi-label classification. In: *Conference of the German classification society on data analysis, machine learning and knowledge discovery*.

Tsoumakas, G., & Vlahavas, I. (2007). Random k-labelsets: An ensemble method for multilabel classification. In: *ECML/PKDD'07, LNCS* (pp. 406–417). Springer.

Zhang, M. L., & Zhou, Z. H. (2007). Ml-knn: A lazy learning approach to multi-label learning. *Pattern Recognition*, *40*(7), 2038–2048.

5.3. A heuristic in A* for inference in nonlinear Probabilistic Classifier Chains

En esta sección se incluye el artículo de revista:

- D. Mena, E. Montañés, J. R. Quevedo, and J. J. Del Coz, "A heuristic in A* for inference in nonlinear Probabilistic Classifier Chains," Knowledge-Based Systems, 2017.

Knowledge-Based Systems 000 (2017) 1–13

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

A heuristic in A* for inference in nonlinear Probabilistic Classifier Chains

Deiner Mena [a,b,1,2], José Ramón Quevedo [a,1], Elena Montañés [a,1,*], Juan José del Coz [a,1]

[a] Artificial Intelligence Center, University of Oviedo at Gijón, Asturias 33204, Spain
[b] Dept. de Ingeniería en Telecomunicaciones e Informática, Universidad Tecnológica del Chocó, Quibdó - Chocó, Colombia

ARTICLE INFO

Article history:
Received 31 October 2016
Revised 29 January 2017
Accepted 18 March 2017
Available online xxx

Keywords:
Multilabel
Classifier chains
Inference
Heuristic search

ABSTRACT

Probabilistic Classifier Chains (PCC) is a very interesting method to cope with multi-label classification, since it is able to obtain the entire joint probability distribution of the labels. However, such probability distribution is obtained at the expense of a high computational cost. Several efforts have been made to overcome this pitfall, proposing different inference methods for estimating the probability distribution. Beam search and the ϵ- approximate algorithms are two methods of this kind. A more recently approach is based on the A* algorithm with an admissible heuristic, but it is limited to be used just for linear classifiers as base methods for PCC. This paper goes in that direction presenting an alternative admissible heuristic for the A* algorithm with two promising advantages in comparison to the above-mentioned heuristic, namely, i) it is more dominant for the same depth and, hence, it explores fewer nodes and ii) it is suitable for nonlinear classifiers. Additionally, the paper proposes an efficient implementation for the computation of the heuristic that reduces the number of models that must be evaluated by half. The experiments show, as theoretically expected, that this new algorithm reaches Bayes-optimal predictions in terms of subset 0/1 loss and explores fewer nodes than other state-of-the-art methods that also provide optimal predictions. In spite of exploring fewer nodes, this new algorithm is not as fast as the ϵ-approximate algorithm with $\epsilon = 0$ when the search for an optimal solution is highly directed. However, it shows its strengths when the datasets present more uncertainty, making faster predictions than other state-of-the-art approaches.

1. Introduction

In Multi-label classification (MLC), a subset from a predefined set of labels may be assigned to an instance. Several real problems fall into this kind of learning task, for instance, tag assignment to resources in social networks, object detection in pictures or medical diagnosis.

Researchers in this field coincide in arguing that one interesting quality of multi-label learning is the dependence that may be concealed among labels. In fact, this dependence has been susceptible of being exploited by algorithms already available in the literature [1–7]. In the last years, Probabilistic Classifier Chains (PCC) have been gained interest due to its promising property of obtaining the entire joint probability. In particular, PCC methods exploit the conditional dependence, widely discussed in the literature [8], instead of the marginal or unconditional dependence.

The high computational cost of the original method [9], which performs an exhaustive search (ES), is the reason why researchers have been get down to work for designing inference algorithms able to estimate such entire joint conditional probability with an non-prohibitive computational cost. One of the forefathers of these inference methods is the ϵ-approximate algorithm (ϵ-A) [10], which uses uniform-cost search to obtain optimal Bayes predictions in terms of subset 0/1 loss considerably reducing the computational cost. Another alternative also available in the literature is beam search (BS) [11], whose reduction in computational time is remarkable, in spite of not guaranteeing optimal predictions in terms of subset 0/1 loss. Also Monte Carlo approaches [10,12] take the classifiers to draw samples of label combination susceptible of being an optimal prediction. Even more recently, the A* algorithm has also been proposed [13,14], which includes an admissible heuristic that guarantees reaching an optimal solution in terms of subset 0/1 loss. Despite authors include a depth parameter in order to control the number of nodes explored in regard to the

* Corresponding author at.
E-mail addresses: deiner.mena@utch.edu.co (D. Mena), quevedo@uniovi.es (J.R. Quevedo), montaneselena@uniovi.es (E. Montañés), juanjo@uniovi.es (J.J.d. Coz).
[1] http://www.aic.uniovi.es
[2] http://www.utch.edu.co

http://dx.doi.org/10.1016/j.knosys.2017.03.015
0950-7051/

computational time spent in computing the heuristic, the use of this heuristic is limited to linear classifiers as base methods for PCC, except when the depth of the heuristic is 1, for which any base classifier with probabilistic output is allowed. But this is the extreme case of spending the least possible time in computing the heuristic, but at expenses of exploring the most number of nodes.

The main contribution of this paper is an alternative heuristic to the one proposed in [13,14] for the A* algorithm that i) it is also admissible, hence, it guarantees Bayes-optimal predictions for the subset 0/1 loss, ii) it also includes a depth parameter to balance the number of nodes explored and the computational time of computing the heuristic, but iii) it overcomes the limitation of being applicable only when linear classifiers are chosen as base methods in PCC and iv) it is more dominant than the one introduced in [13,14] for the same depth, hence, it theoretically explores fewer nodes. The second contribution is to present an efficient algorithm of the heuristic in order to optimize its computation. This algorithm significantly reduces the number of models that must be evaluated to compute the heuristic and outperforms the approach presented in [13,14] both in terms of number of nodes explored and computational time. The experiments also confirm that the proposed method is competitive with respect to other approaches, especially for complex problems.

The rest of the paper is organized as follows. Section 2 formally states the multi-label problem and describes the PCC method. Section 3 gives a brief overview and discussion of some state-of-the-art algorithms to perform inference in PCC. Section 4 details the proposal of this paper. Finally, Sections 5 and 6 respectively show and discuss the experiments carried out and the conclusions.

2. Multi-label classification and probabilistic classifier chains

Before describing the PCC algorithm and the inference methods, let us formally introduce the multi-label classification problem. Firstly, a non-empty and finite set of m labels $\mathcal{L} = \{\ell_1, \ell_2, \ldots, \ell_m\}$ is provided. Secondly, if $\mathcal{X}$ is the instance description space and $\mathcal{Y}$ is the power set of $\mathcal{L}$, a set of instances $S = \{(\vec{x}_1, \vec{y}_1), \ldots, (\vec{x}_n, \vec{y}_n)\}$ is drawn over $\mathcal{X} \times \mathcal{Y}$ according to an unknown probability distribution $\mathbf{P}(\mathbf{X}, \mathbf{Y})$. Notice that the output space can be redefined as $\mathcal{Y} = \{0, 1\}^m$, since $\vec{y}_i$ is, in fact, a vector $\vec{y}_i = (y_{i,1}, y_{i,2}, \ldots, y_{i,m})$ where $y_{i,j} = 1$ means a positive relevance and $y_{i,j} = 0$ means that label ℓ_j is irrelevant for $\vec{x}_i$.

Under this framework, the goal of MLC is to obtain $\vec{f} : \mathcal{X} \longrightarrow \mathcal{Y}$ from S that minimize the risk in terms of a certain loss function $L(\cdot, \cdot)$. That risk can be defined as the expected loss over the entire joint distribution $\mathbf{P}(\mathbf{X}, \mathbf{Y})$, that is,

$$r_L(\vec{f}) = \mathbb{E}_{\mathbf{X},\mathbf{Y}} L(\mathbf{Y}, \vec{f}(\mathbf{X})). \tag{1}$$

This expression can be simplified if the conditional probability distribution $\mathbf{P}(\vec{y} \,|\, \vec{x})$ of $\mathbf{Y} = \vec{y}$ conditioned by $\mathbf{X} = \vec{x}$ is taken into account, obtaining the following expression

$$r_L(\vec{f}(\vec{x})) = \arg\min_{\vec{f}} \sum_{\vec{y} \in \mathcal{Y}} \mathbf{P}(\vec{y} \,|\, \vec{x}) \cdot L(\vec{y}, \vec{f}(\vec{x})). \tag{2}$$

Among the evaluation measures suitable for studying the performance of the algorithms designed for MLC, this paper focuses its attention to the subset 0/1 loss. This loss function is an instance-oriented measure that evaluates for each instance if the set of predicted labels coincides with the set of actual labels or not. Formally, it is defined as[3]

$$L_{S_{0/1}}(\vec{y}, \vec{f}(\vec{x})) = [\![\vec{y} \neq \vec{f}(\vec{x})]\!]. \tag{3}$$

Taking into account the subset 0/1 loss as loss function, the risk minimizer is reduced to optimize the conditional joint distribution, that is,

$$r_{S_{0/1}}(\vec{f}(\vec{x})) = \arg\max_{\vec{y} \in \mathcal{Y}} \mathbf{P}(\vec{y} \,|\, \vec{x}). \tag{4}$$

This simplification of the risk minimizer for the subset 0/1 loss allows providing optimal predictions just considering all the possible combinations for the values of the labels. This task continues being of exponential order, and, as we will see later on, this is the reason why it is necessary to perform inference. But obtaining optimal predictions for a generic loss function $L(\cdot, \cdot)$ means to directly use Eq. (2), which involves summing over an exponential number of label combinations for all f [15]. Hamming loss is another common measure in MLC, which is easier to optimize than subset 0/1. This loss is also an instance-oriented measure that evaluates if the predicted value for each label coincides with the actual value or not. Formally, it is defined as

$$L_H(\vec{y}, \vec{f}(\vec{x})) = \frac{1}{m} \sum_{i=1}^{m} [\![y_i \neq f_i(\vec{x})]\!]. \tag{5}$$

Hence, the risk minimizer for the Hamming loss measure is

$$r_H(\vec{f}(\vec{x})) = (r_{H,1}(f_1(\vec{x})), \ldots, r_{H,m}(f_m(\vec{x}))), \tag{6}$$

where

$$r_{H,i}(f_i(\vec{x})) = \arg\max_{v \in \{0,1\}} \mathbf{P}(y_i = v \,|\, \vec{x}). \tag{7}$$

Therefore, providing optimal predictions in terms of Hamming loss means to obtain optimal predictions for each label independently of the rest of labels [4], so just applying the well-known Binary Relevance approach one can optimize this measure and no inference is required. Hence, this paper just focuses on subset 0/1 loss, whose optimization requires inference to obtain Bayes-optimal predictions.

Among the methods to cope with MLC, let us concentrate on PCC in what follows. The interest of this approach among researches is that, in addition of considering the conditional dependence among labels, it has the ability of optimally estimating the joint probability of a set of labels and, hence, the risk minimizer for the subset 0/1 loss (see Eq. (4)). Let us now explain the method in order to easily deduce that for this purpose it is just necessary to apply the product rule of probability.

First of all, PCC [9] establishes an order of the labels, defining a chain. Secondly and following this order, PCC trains a binary and probabilistic classifier for each label ℓ_j able to estimate the probability of ℓ_j of being relevant for a given instance $\vec{x}$ and taking into account the predictions for the previous labels in the chain, that is, PCC estimates $\mathbf{P}(y_j \,|\, \vec{x}, y_1, \ldots, y_{j-1})$. Hence, the binary probabilistic classifier that estimates the relevance of the label ℓ_j is of the form

$$f_j : \mathcal{X} \times \{0, 1\}^{j-1} \longrightarrow [0, 1]. \tag{8}$$

The training set employed to induce f_j is $S_j = \{(\vec{x}_1, y_{1,j}), \ldots, (\vec{x}_n, y_{n,j})\}$ where now the instance description is $\vec{x}_i = (\vec{x}_i, y_{i,1}, \ldots, y_{i,j-1})$, that is, the original instance description $\vec{x}_i$ together with the relevance of the labels $\ell_1, \ldots, \ell_{j-1}$ placed before ℓ_j in the chain and the class $y_{i,j}$ is the relevance of ℓ_j.

Consequently, the task of evaluating the risk minimizer for the subset 0/1 loss is simplified to just finding a combination of labels that maximizes the expression obtained after applying the product rule of probability to the conditional joint probability, that is:

$$\mathbf{P}(\vec{y} \,|\, \vec{x}) = \prod_{j=1}^{m} \mathbf{P}(y_j \,|\, \vec{x}, y_1, \ldots, y_{j-1}). \tag{9}$$

Theoretically, this expression is valid for all possible orders of the labels, but in practice, those methods based on Classifier Chains

[3] $[\![p]\!]$ equals 1 if p is true and 0 otherwise.

Please cite this article as: D. Mena et al., A heuristic in A* for inference in nonlinear Probabilistic Classifier Chains, Knowledge-Based Systems (2017), http://dx.doi.org/10.1016/j.knosys.2017.03.015

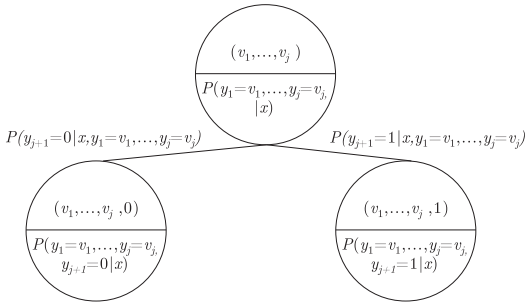


Fig. 1. A generic node and its children of the probability binary tree. The top part of each node contains the combination of labels and the bottom part includes the joint probability of such a combination. The edges are labeled with the conditional probability.

(CC) [16] are label-order dependent and, besides, some of them also present other issues related to error propagation [17,18]. These aspects are topics of interest among researchers but they are out of the scope of this paper. Hence, let us consider an unique order of the chain and concentrate on what follows in performing inference to provide Bayes-optimal predictions in a way that the computational cost may be reduced.

Before analyzing this issue in the next section, note that from a theoretical point of view, this expression holds for any order considered for the labels. In any case, in this paper we assume the order of the labels in the chain to be given, since the goal is just to analyze the performance of the methods, without taking into account the effect of different orders. Hence, we do not include any study about which order can be the best.

Notice that performing inference in PCC can be seen as the different ways of exploring a probability binary tree (see Fig. 1). In this tree, the root is labeled by the empty set of labels and a generic node of level $j < m$ is labeled by $(v_1, v_2, \ldots, v_j)$ with $v_i \in \{0, 1\}$ for $i=1,\ldots,j$. This node has two children who are respectively labeled by $(v_1, v_2, \ldots, v_j, 0)$ and $(v_1, v_2, \ldots, v_j, 1)$ with a respective marginal joint conditional probability $\mathbf{P}(y_1=v_1,\ldots,y_j=v_j,y_{j+1}=0\,|\,\vec{x})$ and $\mathbf{P}(y_1=v_1,\ldots,y_j=v_j,y_{j+1}=1\,|\,\vec{x})$. The probability of the edges between the node and their children are respectively the conditional probabilities $\mathbf{P}(y_{j+1}=0\,|\,\vec{x},y_1=v_1,\ldots,y_j=v_j)$ and $\mathbf{P}(y_{j+1}=1\,|\,\vec{x},y_1=v_1,\ldots,y_j=v_j)$ estimated using the trained classifiers by $1-f_{j+1}(\vec{x},v_1,\ldots,v_j)$ and $f_{j+1}(\vec{x},v_1,\ldots,v_j)$. Finally, the marginal joint conditional probability of the children is estimated using the product rule of probability: $\mathbf{P}(y_1=v_1,\ldots,y_j=v_j,y_{j+1}=v_{j+1}\,|\,\vec{x})=\mathbf{P}(y_{j+1}=v_{j+1}\,|\,\vec{x},y_1=v_1,\ldots,y_j=v_j)\cdot\mathbf{P}(y_1=v_1,\ldots,y_j=v_j\,|\,\vec{x})$.

3. Inference methods in PCC for multi-label classification

Several state-of-the-art methods for inference in PCC have been deeply studied so far (see [19] for an exhaustive review). Let us now briefly describe some of the most promising approaches, with which the method proposed in this paper will be compared later on.

3.1. Greedy search

The so-called method CC [4] is the original inference method for PCC. It performs a Greedy Search (GS) over the probability binary tree, then it just explores one path. At each node, it follows the branch with the highest marginal joint conditional probability. Taking into account the product rule of probability, this branch coincides with that of the highest conditional probability estimated by f_j, since both branches among to choose are children of the same node. Each classifier f_j is successively feed by the labels previously predicted along the path. This method tries to optimize the subset 0/1 loss, but an analysis performed in this direction [10] have shown poor results and in general it does not reach an optimal solution.

3.2. ϵ-Approximate algorithm

The *ϵ-Approximate* (ϵ-A) algorithm [10] has promisingly arisen in between the high computational cost of obtaining the entire joint conditional probability by ES and the poor performance of GS. The method explores more than one path and it consists of expanding just the nodes whose marginal joint conditional probability exceeds the value of the threshold $\epsilon=2^{-k}$, with $1 \leq k \leq m$. The values of this threshold determines the strategy of the algorithms. For instance, the particular case of $\epsilon = 0.0$ (or any value in the interval $[0, 2^{-m}]$, that is, $k=m$) means to perform a uniform-cost search (UC) that always reaches an optimal solution. Besides, the method goes towards GS as ϵ increases and equals GS when $\epsilon=2^{-1}=0.5$ ($k=1$). This method optimizes the subset 0/1 loss to a greater or lesser degree depending on the value of ϵ [10].

3.3. Beam search

Beam Search (BS) [11,20] also explores several paths, whose number is limited by a parameter b called *beam width*. In fact, at most b nodes are explored at each level. Hence, i) it performs an ES in the first k^*-1 levels, where k^* is the lowest integer such that $b < 2^{k^*}$ and ii) it explores just b nodes from the k^*-th level to the leaves, those with the highest marginal joint conditional probability. BS encapsulates both GS (with $b=1$) and ES (with $b=2^m$), then, adjusting the beam width, this method establishes a balance between the computational cost and the performance. BS also tends to optimize the subset 0/1 loss when b is large enough. The authors of [11] state that BS converges rapidly to the optimal predictions in terms of subset 0/1 loss with $b < 15$. They also observe that with $b=15$ a significant fraction of the actual labels are predicted as relevant even when the label combination selected is not exactly correct.

3.4. Methods based on Monte Carlo sampling

Monte Carlo sampling was also proposed [10,12] to perform inference in PCC. In them, the classifiers f_j induce probability distributions to draw random values from which the prediction of the value of ℓ_j is obtained whereas in GS, such prediction is taken directly from the evaluation of f_j. This flexibility of Monte Carlo approaches makes possible to perform the process several times, each one offering a label combination, although the same combination may appear more than once. After generating a certain number of label combinations, they must be aggregated to provide the final prediction. In this sense, one can take the most frequent label combination (the mode) [10] or the label combination with the highest joint conditional probability [12]. In any case, Monte Carlo approaches converge to optimize the subset 0/1 loss when the number of predictions drawn is large enough, but they do not guarantee to reach optimal predictions. As final remark, the number of nodes expanded by these approaches is $m \cdot q$, where m is the number of labels and q the number of predictions drawn.

3.5. A* algorithm using an admissible heuristic designed for linear classifiers

The A* algorithm has been also proposed for inference in PCC [13,14] using an admissible heuristic only suitable when the clas-

Please cite this article as: D. Mena et al., A heuristic in A* for inference in nonlinear Probabilistic Classifier Chains, Knowledge-Based Systems (2017), http://dx.doi.org/10.1016/j.knosys.2017.03.015

sifiers f_i are linear. Since the main contribution of the paper continues this research designing an alternative heuristic with more promising properties, let us give a detailed description of this approach. The A* algorithm [21,22] explores the best node according to an evaluation function e which can be decomposed in two other evaluation functions g (called the known part) and h (called the unknown part). In this way, for a generic node k, $g(k)$ computes the cost or gain for going from the root to the node k in a tree, whereas $h(k)$ estimates the cost or gain for going from the node k to a leave. Hence, $e(k)$ computes the cost or gain for going from the root to a leave through the node k. Usually, it is possible to obtain the exact value of $g(k)$, but $h(k)$ must be estimated using an heuristic. One of the principles of A* algorithm is that only if the heuristic is admissible[4], the A* algorithm is able to reach an optimal solution. Another crucial principle is that the A* algorithm is the most efficient algorithm for any heuristic, because there is not any other algorithm that using the same heuristic expands fewer nodes than A*. The particular case of using the A* algorithm for inference in PCC means to consider a gain rather than a cost, since the interest is to optimize the subset 0/1 loss. Besides, the function of g and h to obtain e is the product function due to the product rule of probability, that is, for a node k of the level j, $e(k) = g(k) \cdot h(k)$. Then, $e(k) = \mathbf{P}(y_1, \ldots, y_m \,|\, \vec{x})$ will be the joint conditional probability that applying the product rule of probability is decomposed into $g(k) = \mathbf{P}(y_1, \ldots, y_j \,|\, \vec{x})$, which is the marginal joint conditional probability of labels from ℓ_1 to ℓ_j in the order of the chain, and $h(k) = \mathbf{P}(y_{j+1}, \ldots, y_m \,|\, \vec{x}, y_1, \ldots, y_j)$, which is the marginal joint conditional probability of labels from ℓ_{j+1} to ℓ_m.

Notice that the above-mentioned requirement for g of being exactly computed is satisfied, since the values of $y_1, \ldots, y_j$ are known at level j. On the other hand, h must be estimated, since the values of $y_{j+1}, \ldots, y_m$ are unknown, so h will be an heuristic. Moreover, it must be an admissible heuristic to guarantee reaching an optimal solution. Also, g is, in fact, the same evaluation function that ϵ-A and BS consider in order to choose the node to be explored next. Even more, ϵ-A with $\epsilon = 0$ is equivalent to the A* algorithm with $h = 1$ as heuristic [13,14].

The design of an admissible heuristic requires to define a function that returns an upper bound of $\mathbf{P}(y_{j+1}, \ldots, y_m \,|\, \vec{x}, y_1, \ldots, y_j)$ as close as possible to the exact value. An upper bound can be easily obtained performing the product of the highest probability for each remaining level, from label ℓ_{j+1} to label ℓ_m. Besides, the depth of this computation may be limited to d levels, from ℓ_{j+1} to ℓ_{j+d} and upperbounding the probability by 1 for the remaining levels, from ℓ_{j+d+1} to ℓ_m. This is the heuristic proposed in [13,14] that shall be denoted here as h_l^d, in which d represents the depth and the subscript l means that the heuristic is only valid for linear models. An example is depicted in Fig. 2. There, the value of the heuristic h_l^2 for the root node is 0.48 which is an upper bound of the exact value of $\mathbf{P}(y_{j+1}, \ldots, y_m \,|\, \vec{x}, y_1, \ldots, y_j) = 0.216$. Besides, this value is also an upper bound of the highest joint probability for those nodes at the $j+d$ - th level (0.36).

Therefore, the computation of h_l^d is reduced to calculate the highest probability for each level. This is not difficult for linear classifiers, since this kind of classifiers, f_i, are defined by means of $(\vec{w}^i = [\vec{w}_x^i, \vec{w}_y^i], \beta^i)$, where $\vec{w}_x^i$ and $\vec{w}_y^i$ are respectively the weight vectors for the $\vec{x}$ part (the description of the example) and the $\vec{y}$ part (the values of the previous labels in the chain for the example) and β^i is the intercept term. Mathematically, h_l^d can be computed as follows (see [14] for a complete deduction):

$$h_l^d = \prod_{i=j+1}^{j+d} \max_{v \in \{0,1\}} \{\mathbf{P}(y_i = v \,|\, \vec{x}, y_1, \ldots, y_j, y_{j+1}^{i,v}, \ldots, y_{i-1}^{i,v})\} \cdot \prod_{i=j+d+1}^{m} 1, \tag{10}$$

where $\mathbf{P}(y_i = v \,|\, \vec{x}, y_1, \ldots, y_j, y_{j+1}^{i,v}, \ldots, y_{i-1}^{i,v})$ for each i with $j+1 \leq i \leq j+d$. The maximum value for either $v = 1$ or $v = 0$ only depends on the value that takes each $y_k^{i,v}$ because the part $\langle \vec{w}_x^i, \vec{x} \rangle + \beta_i$ is constant for each instance $\vec{x}$. Hence, it is straightforward to determine $y_k^{i,v}$ looking at the corresponding value of $w_{y,k}^i$ depending on v:

1) using $f_i(\vec{x}, y_1, \ldots, y_j, y_{j+1}^{i,1}, \ldots, y_{i-1}^{i,1})$ when $v = 1$. In this case, we set $y_k^{i,1}$ for $j+1 \leq k \leq i-1$ to 1 if the corresponding $w_{y,k}^i$ of the linear classifier is positive and 0 otherwise. These values for $y_k^{i,1}$ for $j+1 \leq k \leq i-1$ make the probability $\mathbf{P}(y_i = v \,|\, \vec{x}, y_1, \ldots, y_j, y_{j+1}^{i,v}, \ldots, y_{i-1}^{i,v})$ be maximum when $v = 1$.
2) using $1 - f_i(\vec{x}, y_1, \ldots, y_j, y_{j+1}^{i,0}, \ldots, y_{i-1}^{i,0})$ when $v = 0$. In this case, we set $y_k^{i,1}$ for $j+1 \leq k \leq i-1$ to 1 if the corresponding $w_{y,k}^i$ of the linear classifier is negative and 0 otherwise. These values for $y_k^{i,1}$ for $j+1 \leq k \leq i-1$ make the probability $\mathbf{P}(y_i = v \,|\, \vec{x}, y_1, \ldots, y_j, y_{j+1}^{i,v}, \ldots, y_{i-1}^{i,v})$ be maximum when $v = 0$.

Taking the maximum of the probability $\mathbf{P}(y_i = v \,|\, \vec{x}, y_1, \ldots, y_j, y_{j+1}^{i,v}, \ldots, y_{i-1}^{i,v})$ between the two values when $v = 1$, obtained in 1), and when $v = 0$, obtained in 2), for each i with $j+1 \leq i \leq j+d$ and performing the product of this maximum for all i such that $j+1 \leq i \leq j+d$, the value of the heuristic h_l^d of depth d for a node of level j is provided. Remember that the values of y_i for $1 \leq i \leq j$ are known at all nodes of level j, so they depend on the node of the level j for which the heuristic is computed.

Fig. 2 illustrates the computation of the heuristic h_l^2 when $d = 2$ levels of depth for the unique node (the root) of level $j = 0$ and $m = 3$. In this case i takes values $i = 1, 2$ ($1 = j+1 \leq i \leq j+d = 2$).

1) For $i = 1$, $\mathbf{P}(y_1 = 1 \,|\, \vec{x}) = 0.6$ and $\mathbf{P}(y_1 = 0 \,|\, \vec{x}) = 0.4$, hence, the maximum is reached when $v = 1$ and is 0.6.
2) For $i = 2$, $\mathbf{P}(y_2 = 1 \,|\, \vec{x}, y_1^{2,1})$ equals 0.8 when $y_1^{2,1} = 0$ and equals 0.6 when $y_1^{2,1} = 1$, hence, for $v = 1$ the maximum is 0.8. In the same way, $\mathbf{P}(y_2 = 0 \,|\, \vec{x}, y_1^{2,0})$ equals 0.2 when $y_1^{2,0} = 0$ and equals 0.4 when $y_1^{2,0} = 1$, hence, for $v = 0$ the maximum is 0.4. Now, the maximum between 0.8 (for $v = 1$) and 0.4 (for $v = 0$) is 0.8, reached for $v = 1$.

Computing the product of the maximum reached in 1) and 2), one obtains $0.6 \cdot 0.8 = 0.48$, which is the value of the heuristic for the root.

In addition to the limitation of the heuristic h_l^d of being just valid for linear classifiers as base models, it is a quite time consuming heuristic because it requires to evaluate $2d-1$ models. Hence, the power of exploring fewer nodes than other state-of-the-art algorithms is lessened by the time spent in computing h_l^d [14].

4. Exhaustive admissible heuristic in the A* algorithm for nonlinear classifiers

This section details the steps followed to build the exhaustive heuristic proposed in this paper for the A* algorithm as an alternative to the one presented in [13,14]. The goal is to design an heuristic that may be used with both, linear and nonlinear models. First of all, such heuristic must be admissible, hence, it cannot underestimate the marginal joint conditional probability

[4] An heuristic h is admissible if for any node k, $h(k)$ does not overestimate (in case of $e(k)$ is a cost function) or underestimate (in case of $e(k)$ is a gain function) the actual cost or gain.

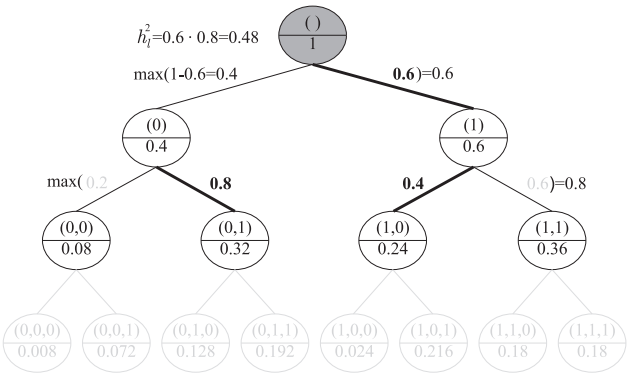


Fig. 2. An example of h_l^d heuristic for $d = 2$ levels of depth, hence, only two levels of depth from there are expanded. The node in dark grey is the node for which the heuristic is computed. Bold lines indicate the evaluations of the classifiers that must be computed to obtain the value of the heuristic, which are, at most, two evaluations per classifier, that which respectively correspond to the maximum of the left branches and to the maximum of the right branches per level. Notice that only one evaluation of the classifier is performed for the contiguous level of the node for which the heuristic is computed.

$\mathbf{P}(y_{j+1}, \ldots, y_m \,|\, \vec{x}, y_1, \ldots, y_j)$. This leads to trivially think of selecting the maximum value that this probability can take, which will be the optimal possibility h^*:

$$h^* = \max_{(y_{j+1},\ldots,y_m)\in\{0,1\}^{m-j}} \mathbf{P}(y_{j+1}, \ldots, y_m \,|\, \vec{x}, y_1, \ldots, y_j) \tag{11}$$

Applying the product rule of probability, that expression leads to

$$h^* = \max_{(y_{j+1},\ldots,y_m)\in\{0,1\}^{m-j}} \prod_{i=j+1}^{m} \mathbf{P}(y_i \,|\, \vec{x}, y_1, \ldots, y_j, y_{j+1}, \ldots, y_{i-1}) \tag{12}$$

The main problem is that evaluating this maximum is not computationally acceptable, since it means to apply an ES for the subset of labels $\mathcal{L} = \{\ell_{j+1}, \ldots, \ell_m\}$.

The main idea consists in relaxing the optimality of the heuristic h^* in exchange of designing a computationally feasible heuristic. Our proposal involves including a parameter d for limiting the depth of the heuristic in order to control the computational cost. In this sense, an ES is carried out just until d levels of depth in the tree. This is possible thanks to the fact that any probability is delimited by 1. Hence, for a node of the level j and a value of d such that $0 \leq d \leq m - j$, the probabilities $\mathbf{P}(y_i \,|\, \vec{x}, y_1, \ldots, y_j, y_{j+1}, \ldots, y_{i-1})$ are computed i) applying ES from level $j+1$ to $j+d$, that is, evaluating $\mathbf{P}(y_i \,|\, \vec{x}, y_1, \ldots, y_j, y_{j+1}, \ldots, y_{i-1})$ for these levels and taking the combination that reaches the maximum and ii) delimiting $\mathbf{P}(y_i \,|\, \vec{x}, y_1, \ldots, y_j, y_{j+1}, \ldots, y_{i-1})$ by 1 from level $j+d+1$ to m. Hence, we define the heuristic h_{nl}^d, in which the subscript nl indicates that the heuristic is also valid for nonlinear classifiers, as:

$$h_{nl}^d = \max_{(y_{j+1},\ldots,y_{j+d})\in\{0,1\}^d} \prod_{i=j+1}^{j+d} \mathbf{P}(y_i \,|\, \vec{x}, y_1, \ldots, y_j, y_{j+1}, \ldots, y_{i-1}) \cdot \prod_{i=j+d+1}^{m} 1 \tag{13}$$

or equivalently

$$h_{nl}^d = \max_{(y_{j+1},\ldots,y_{j+d})\in\{0,1\}^d} \prod_{i=j+1}^{j+d} \mathbf{P}(y_i \,|\, \vec{x}, y_1, \ldots, y_j, y_{j+1}, \ldots, y_{i-1}) \tag{14}$$

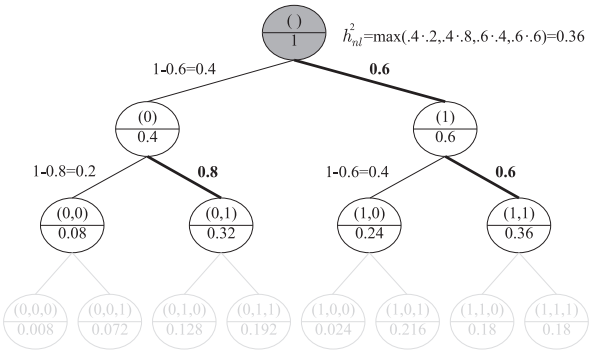
Notice that h_{nl}^d continues being admissible, but less dominant[5] than the optimal heuristic h^*. The particular case of $d = 0$ is the constant heuristic $h = 1$. Hence, h_{nl}^d is in turn more dominant than h_{nl}^0. In general, h_{nl}^d becomes more dominant as d increases, and then, it gets to explore fewer nodes, but at the cost of increasing the computational cost. This computational cost mainly depends on the number of models that must be evaluated to calculate the heuristic. Theoretically, h_{nl}^d requires to evaluate $2^d - 1$ models, which means to evaluate many more models than the number of models evaluated by h_l^d $(2d - 1)$. However, an appropriate implementation of h_{nl}^d (see next Section) can reduce that number to 2^{d-1}, making the computational cost of both heuristics comparable for $d \leq 4$.

Fig. 3 illustrates the ways of computing the heuristic h_{nl}^d for $d = 2$ in two different cases: for a parent node and for one of its child. Looking to the former case (Fig. 3a), the value of h_{nl}^2 for the root node is 0.36, which corresponds to the highest joint conditional probability of the nodes of the 2 - th level (label combination (1,1)). This value is an upper bound of the optimal value $h^* = 0.211$ which corresponds to the path of the label combination (1,0,1). But the most interesting issue of our proposal is that this bound is tighter than the one computed by h_l^2 (0.48) (see Fig. 2). Notice that for the latter case (Fig. 3b), it is just necessary to evaluate the classifiers of the last level because the classifiers of previous levels were evaluated when the heuristic was computed for the parent node.

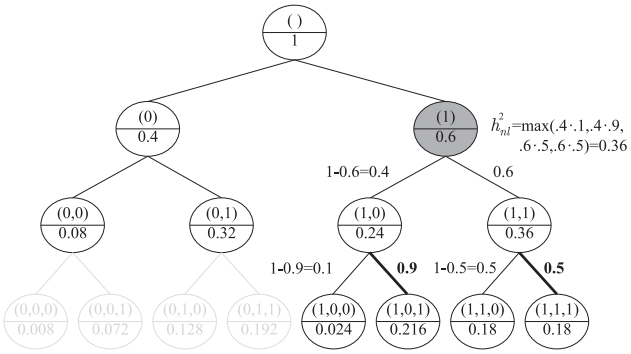
The reason why h_l^d is not able to deal with nonlinear models is because h_l^d is computed taking into account the weights of the linear models (see (10)). In case of h_{nl}^d, those weights are not required because the computation of h_{nl}^d just wisely applies the models over the label combinations at the deepest level of the corresponding subtree. For instance, to compute h_{nl}^2 for the child node in Fig. 3b, we just need to apply model f_3 over the label combinations $(y_1 = 1, y_2 = 0)$ and $(y_1 = 1, y_2 = 1)$. With these two evaluations, it is possible to compute the probabilities $\mathbf{P}(y_3 = 1 \,|\, \vec{x}, y_1 = 1, y_2 = 0) = 0.9$ and $\mathbf{P}(y_3 = 1 \,|\, \vec{x}, y_1 = 1, y_2 = 1) = 0.5$. The other two probabilities are trivially obtained as $\mathbf{P}(y_3 = 0 \,|\, \vec{x}, y_1 = 1, y_2 = 0) = 1 - \mathbf{P}(y_3 = 1 \,|\, \vec{x}, y_1 = 1, y_2 = 0) = 1 - 0.9 = 0.1$ and $\mathbf{P}(y_3 = 0 \,|\, \vec{x}, y_1 = 1, y_2 = 1) = 1 - \mathbf{P}(y_3 = 1 \,|\, \vec{x}, y_1 = 1, y_2 = 1) = 1 - 0.5 = 0.5$.

[5] When e is a gain function, an heuristic h_1 is more dominant than another heuristic h_2, denoted by $(h_1 \prec h_2)$, if the value of h_2 is not lower than the value of h_1 for any node.

Please cite this article as: D. Mena et al., A heuristic in A* for inference in nonlinear Probabilistic Classifier Chains, Knowledge-Based Systems (2017), http://dx.doi.org/10.1016/j.knosys.2017.03.015



(a) A parent node



(b) A child node

Fig. 3. An example of h^d_{nl} heuristic for $d = 2$ levels of depth, hence, only two levels of depth from there are expanded. The nodes in dark grey are the nodes for which the heuristic is computed: (a) Root node (unique node of level $j = 0$) (b) a child node (the right node out of two existing nodes in level $j = 1$). Bold lines indicate the evaluations of the classifiers that must be computed to obtain the value of the heuristic. Notice that those evaluations are only those corresponding to the classifier of the deepest level, except for the root node in which all possible evaluations must be computed.

Finally, Fig. 4 shows how different are the paths followed by A* when h^2_l (top) and h^2_{nl} (bottom) are taken as heuristics. First, notice that both approaches reach a Bayes-optimal solution (unique in this case), which it is the label combination (1,0,1) that has the highest joint conditional probability. However, h^2_{nl} requires fewer steps than h^2_l, since it follows the most direct path. This is due to the fact that the estimates computed by h^2_{nl} are more accurate. In this case, this means that the estimates are smaller and closer to the optimum. Notice that this occurs for the three top-nodes of the tree in Fig. 4 (0.36 vs. 0.48, 0.48 vs. 0.72 and 0.36 vs. 0.54).

4.1. An efficient implementation of the heuristic

This section proposes a complete implementation of the A* algorithm using h^d_{nl} as heuristic, including an efficient algorithm for computing h^d_{nl} that reduces the number of models that must be evaluated from $2^d - 1$ to 2^{d-1}. The key idea is that each node stores the probabilities necessary to compute the heuristic (see Fig. 5). Then, when a node is expanded, those probabilities are copied and reused to calculate the heuristic for its children. Applying this solution, only the probabilities of the d - th level must be computed because the rest were already calculated for the parent node. This implementation makes h^d_{nl} evaluate fewer or equal models than h^d_l when $d \leq 3$.

Algorithm 1 shows the pseudocode of the implementation of A* algorithm using h^d_{nl}. Initially, for the root, all the probabilities until level d must be computed (see *CompProbs* in line 3). When a non root node is expanded, the information of its parent is copied and reused (see *CopyHeuristicVector* in lines 13 and 19). In this case, only the last-level classifier is evaluated for obtaining just the new needed probabilities (see *CompNewProbs* in lines 14 and 21 respectively for the left and right children of the node expanded). Both children of an expanded node are included in the list of candidate nodes to be expanded (see *Insert* in lines 16 and 22), taking each time the node with the highest joint conditional probability (see line 6). The algorithm ends when the expanded node is a leaf or equivalently the level of the expanded node is m.

Algorithm 2 details the procedure to copy reusable probabilities of the vector $\vec{v}$ (*CopyHeuristicVector*). Particularly, when the heuristic must be computed for a node, it first copies the probabilities previously computed for its parent. Also, Algorithm 3 describes the recursive function which computes all the joint probabilities from the vector of probabilities ($\vec{v}$) of a node. In particular, it takes advantage of the probabilities previously saved in the way that none probability is computed twice. Finally, Algorithm 4 presents the iterative function to compute the missing probabilities of the vector

Please cite this article as: D. Mena et al., A heuristic in A* for inference in nonlinear Probabilistic Classifier Chains, Knowledge-Based Systems (2017), http://dx.doi.org/10.1016/j.knosys.2017.03.015

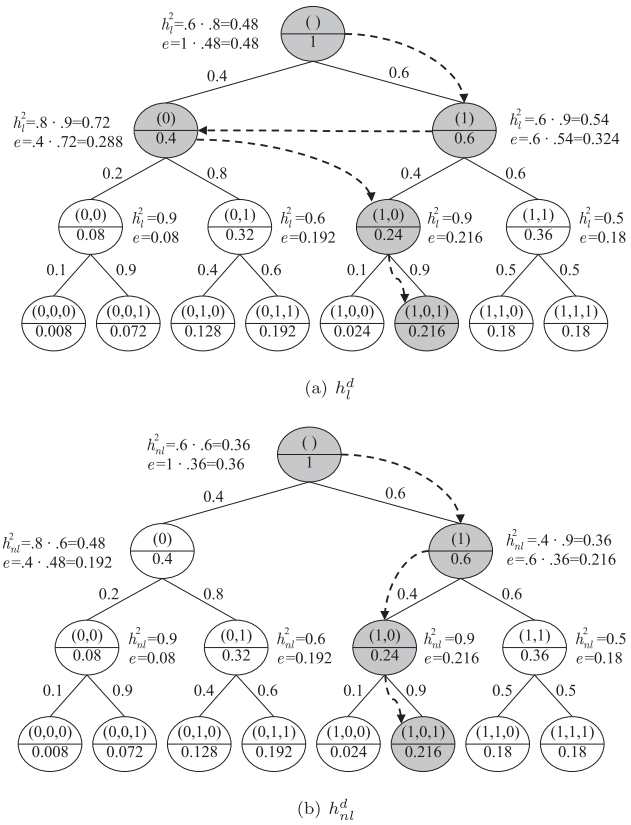


Fig. 4. Path (dotted arrows) followed by the algorithm A* using (a) h_l^d and (b) h_{nl}^d both with $d = 2$ levels of depth. Both approaches reach the Bayes-optimal solution but h_{nl}^2 requires fewer steps than h_l^2.

$\vec{v}$ for a new node. Hence, this function evaluates the classifier of the $j+d$ - th level in order to obtain the new probabilities needed to compute the heuristic for the current node. It also stores such probabilities for being reused just in case the heuristic for any descendant must be computed later on.

Notice that all of these algorithms make sense if $d \geq 2$, since for the particular case of $d = 1$, where only one level is explored, the implementation may be considerably simplified. At sight of them, the computation of h_{nl}^d means to calculate new 2^{d-1} evaluations of the last-level $(j + d)$ classifier each time the heuristic is computed for a node, in contrast to the new $2d - 1$ evaluations of d different classifiers for h_l^d [14] (see Fig. 5). Hence, both heuristics need to compute the same new classifier evaluations for $d = 1$ (1 evaluation). The number of classifier evaluations of h_{nl}^d and h_l^d are approximately equal for $2 \leq d \leq 4$ (2 vs. 3 evaluations when $d = 2$, 4 vs. 5 when $d = 3$, and 8 vs. 7 when $d = 4$). From $d = 5$, the number of classifier evaluations to compute begins to differ, being 16 evaluations for h_{nl}^d and 9 for h_l^d. This fact suggests not to take high values of d, because if not the advantage gained by the dominance of the heuristic would not offset the computational cost involved in its computation. In exchange of that, i) h_{nl}^d is more dominant than h_l^d for the same level of depth d and ii) unlike h_l^d, h_{nl}^d is suitable for nonlinear classifiers as base methods. According to the experiments performed, our advice is not to consider values of d greater than 3.

Table 1
Properties of the datasets.

Datasets	Instances	Attributes	Labels	Cardinality
bibtex	7395	1836	159	2.40
corel5k	5000	499	374	3.52
emotions	593	72	6	1.87
enron	1702	1001	53	3.38
flags	194	19	7	3.39
image	2000	135	5	1.24
mediamill*	5000	120	101	4.27
medical	978	1449	45	1.25
reuters	7119	243	7	1.24
scene	2407	294	6	1.07
slashdot	3782	1079	22	1.18
yeast	2417	103	14	4.24

5. Experiments

The experiments were carried out over the same benchmark multi-label datasets used in [14], whose number of labels ranges from 5 to 374 (see Table 1 for the main properties of the datasets). The algorithm A* using the exhaustive heuristic introduced in this paper, h_{nl}^d, is compared to the heuristic, h_l^d, proposed in [13,14]. As commented before, the values of d must not be so high due to the computational cost, then only values of $d \in \{1, 2, 3\}$ were considered. The algorithm A* using both heuristics are in turn com-

Please cite this article as: D. Mena et al., A heuristic in A* for inference in nonlinear Probabilistic Classifier Chains, Knowledge-Based Systems (2017), http://dx.doi.org/10.1016/j.knosys.2017.03.015

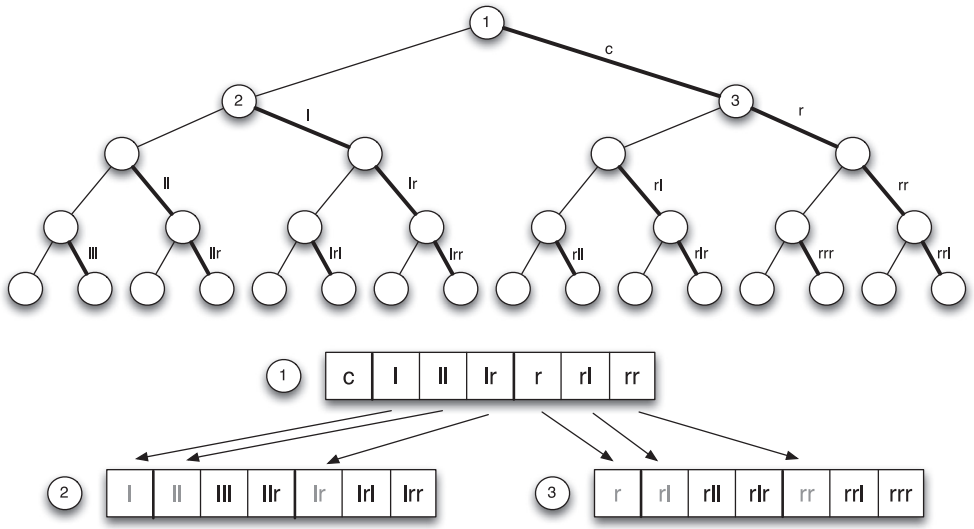


Fig. 5. Optimal computation of the heuristic. The heuristic vector represents the computed probabilities of the tree using a preorder traversal. The example assumes that node 1 is expanded, so its children, nodes 2 and 3, must be added to the *Q* list. The computation of the heuristic for both nodes reuses the information contained in the heuristic structure of node 1. Only the probabilities of the lower level must be computed (positions in bold in vectors 2 and 3).

Algorithm 1 Pseudocode of the implementation of A* algorithm using h^d_{nl}.

function A*
Input: m, d, $\vec{x}$, $\{[\mathbf{W} = \{\vec{w}^i\}, \vec{\beta} = \{\beta^i\}], i = 1, \ldots, m\}$ a CC Linear Model
Output: Label combination with highest probability for $\vec{x}$
  $\vec{v} \leftarrow Inic(\vec{x}, [\mathbf{W}, \vec{\beta}], d)$ // Probabilities of the h_{nl} for the root node
  $e \leftarrow Max(CompProbs(\vec{1}_{2^d}, \vec{v}, d))$
  $Q[1] \leftarrow \{[\,], 0, 1, e, \vec{v}\}$ // root, empty label set, level 0, $g = 1$ and $\vec{v}$
  while true **do**
    $Node \leftarrow Max(Q)$
    if $Node.Level = m$ **then**
      return $Node.Labels$ // Leaf node
    $level \leftarrow Node.Level + 1$
    $P \leftarrow Node.v[1]$ // Probability that the label is 1
    // Left child
    $g \leftarrow Node.g * (1 - P)$
    $\vec{vl} \leftarrow CopyHeuristicVector(Left, \vec{v}, d)$
    $CompNewProbs(\vec{vl}, d, \vec{x}, [\mathbf{W}, \vec{\beta}], [Node.Labels\ 0], level)$
    $h^d_{nl} \leftarrow Max(JointProbs(\vec{1}_{2^d}, \vec{vl}, d))$
    $Insert(Q, \{[Node.Labels\ 0], level, g, g * h^d_{nl}, \vec{vl}\})$
    // Right child
    $g \leftarrow Node.g * P$
    $\vec{vr} \leftarrow CopyHeuristicVector(Right, \vec{v}, d)$
    $CompNewProbs(\vec{vr}, d, \vec{x}, [\mathbf{W}, \vec{\beta}], [Node.Labels\ 1], level)$
    $h^d_{nl} \leftarrow Max(JointProbs(\vec{1}_{2^d}, \vec{vr}, d))$
    $Insert(Q, \{[Node.Labels\ 1], level, g, g * h^d_{nl}, \vec{vr}\})$

Algorithm 2 Copy reusable probabilities of the vector $\vec{v}$.

function *CopyHeuristicVector*
Input: *Child*, $\vec{v}$, d
Output: An incomplete heuristic vector, copy corresponding values from $\vec{v}$
  $\vec{v}' \leftarrow AllocMemory(2^d - 1)$
  $l \leftarrow 2$ // Beginning of left part
  $r \leftarrow 1 + 2^{d-1}$ // Beginning of right part
  if $Child = Left$ **then** // Beginning of the part to be copied
    $i \leftarrow l$
  else
    $i \leftarrow r$
  $\vec{v}'[1] \leftarrow \vec{v}[i]$ // Probability of the top level
  $i \leftarrow i + 1$
  $ElementsPerLevel \leftarrow 1$ // the next level has 1 element
  $level \leftarrow 3$
  while $level \leq d$ **do**
    for $j = [1 : ElementsPerlevel]$ **do**
      $\vec{v}'[l] \leftarrow \vec{v}[i]$
      $l \leftarrow l + 1, i \leftarrow i + 1$
    for $j = [1 : ElementsPerlevel]$ **do**
      $\vec{v}'[r] \leftarrow \vec{v}[i]$
      $r \leftarrow r + 1, i \leftarrow i + 1$
    $ElementsPerLevel \leftarrow ElementsPerLevel * 2$
    $level = level + 1$

pared to other state-of-the art algorithms for inference in PCC. Particularly, it is compared with GS, ϵ-A with different values of $\epsilon \in \{0.0, 0.25, 0.5\}$ and BS with several values of $b \in \{1, 2, 3\}$. Remember that GS, ϵ-A (0.5) and BS with b=1 explore the same path in the tree, so they attain identical results. The same occurs to ϵ-A (0.0), UC and the algorithm A* with both h^0_l and h^0_{nl} (or equivalently whit the heuristic $h = 1$). None experiment is included for the Monte Carlo methods, since they have been shown to be less efficient than the ϵ-A algorithm [19] in addition of not guarantee to obtain an optimal prediction.

Please cite this article as: D. Mena et al., A heuristic in A* for inference in nonlinear Probabilistic Classifier Chains, Knowledge-Based Systems (2017), http://dx.doi.org/10.1016/j.knosys.2017.03.015

Algorithm 3 Recursive function to compute all the joint probabilities from the vector of probabilities ($\vec{v}$) of a given node.

function *JointProbs*
Input: $\vec{P}$, $\vec{v}$, d // P is the vector of the joint probabilities
Output: Each element of P multiplied by the corresponding probabilities of $\vec{v}$
  if $d = 1$ **then**
    return $[\vec{P}[1] * (1 - \vec{v}[1]), \vec{P}[2] * \vec{v}[1]]$
  else
    $\vec{v}' \leftarrow \vec{v}[2 : end]$
    return $[(1-\vec{v}[1]) * JointProbs(\vec{P}[1 : len(\vec{P})/2], \vec{v}'[1 : len(\vec{v}')/2], d-1),$
    $\vec{v}[1] * JointProbs(\vec{P}[len(\vec{P})/2 + 1 : end], \vec{v}'[len(\vec{v}')/2 + 1 : end], d-1)]$

Algorithm 4 Iterative function to compute the missing probabilities of the vector $\vec{v}$ for a new node.

function *CompNewProbs*
Input: $\vec{v}$, d, *Labels*, *Level*, $\vec{x}$, $\{[\mathbf{W} = \{\vec{w}^i\}, \vec{\beta} = \{\beta^i\}], i = 1, \ldots, m\}$ a CC Linear Model
Output: $\vec{v}$
  $LabelComb \leftarrow BinaryPerm(d-1)$ // All bit patterns of length $d-1$
  $j \leftarrow 1$
  for $i = [1 + 2^{d-1} - 2^{d-2} : 2^{d-1}]$ **do** // Left probabilities
    $\vec{v}[i] \leftarrow Prob(\vec{x}, [\mathbf{W}, \vec{\beta}], [Labels\ LabelComb(j)], Level)$
    $j \leftarrow j + 1$
  $j \leftarrow 1$
  for $i = [1 + len(\vec{v}) - 2^{d-2} : len(\vec{v})]$ **do** // Right probabilities
    $\vec{v}[i] \leftarrow Prob(\vec{x}, [\mathbf{W}, \vec{\beta}], [Labels\ LabelComb(j)], Level)$
    $j \leftarrow j + 1$

5.1. Subset 0/1 scores

First of all, we have tested the heuristic h^d_{nl} for the A* algorithm in comparison with other state-of-the-art algorithms (GS, ϵ-A with different values of $\epsilon \in \{0.0, 0.25, 0.5\}$ and BS with several values of $b \in \{1, 2, 3\}$) using as base method a nonlinear classifier. Particularly, the nonlinear base classifier taken was SVM with probabilistic output and with radial basis function as kernel function. The parameters C of SVM and G of the radial basis kernel were optimized applying a grid search over the values of $C, G \in \{10^{-2}, \ldots, 10^{1}\}$ using the brier loss [23] as target function estimated through a 2-fold cross validation of five repetitions. The heuristic h^d_l is out of this comparison, since its computation is only suitable when a linear classifier is taken as base method.

Tables 2 and 3 respectively show the subset 0/1 loss estimated by a 10-fold cross validation for UC, A* with the heuristic h^d_{nl}, GS, ϵ-A algorithm and BS and the p-values of the underlying Wilcoxon signed-rank test. The approaches UC, ϵ-A(0.0) and the A* algorithm using whatever admissible heuristic provide optimal solutions in terms of the subset 0/1 loss (see the scores in the first column of the table). As seen, there is four cases out of 12 for which ϵ-A(0.25) equals them. None case exists where GS (or BS(1) or ϵ-A(0.5)) improves the algorithms with theoretically obtain optimal solutions. BS(2) does it in 2 cases (flags and slashdot) and equals them in other 2 cases. Finally, BS(3) gets equal results in 8 cases and it is never better. The p-values of the Wilcoxon signed-rank test confirm these results, namely, i) the methods that guarantee reaching Bayes-optimal predictions are significantly better than ϵ-A(0.25) and GS (or BS(1) or ϵ-A(0.5)) for $\alpha = 0.01$, and than BS(2) but for $\alpha = 0.05$. ii) The predictions of BS become close to optimal predictions as b increases.

Table 2
Subset 0/1 loss for the methods when a nonlinear classifier (SVM with radial basis kernel) is taken as base method for PCC. Those scores that are equal or lower than the optimal predictions are in bold.

Datasets	UC/A* ϵ-A(0.0)	ϵ-A(0.25)	GS/BS(1) ϵ-A(0.5)	BS(2)	BS(3)
bibtex	**81.20**	81.22	81.34	81.22	**81.20**
corel5k	**98.02**	98.48	98.54	98.22	98.10
emotions	**66.10**	**66.10**	69.79	66.77	**66.10**
enron	**84.20**	84.84	86.78	84.49	84.31
flags	**82.45**	83.47	83.45	82.97	**82.45**
image	**58.20**	58.45	60.20	**58.05**	**58.20**
mediamill*	**84.36**	**84.36**	85.02	84.48	84.40
medical	**28.73**	**28.73**	29.14	**28.73**	**28.73**
reuters	**21.98**	22.00	23.14	22.03	**21.98**
scene	**31.15**	**31.15**	32.40	**31.15**	**31.15**
slashdot	**51.27**	51.53	53.15	**51.24**	51.27
yeast	**77.24**	77.62	79.77	77.33	**77.24**

Table 3
Wilcoxon signed-rank test for the methods when a nonlinear classifier is taken as base method for PCC.

Optimal methods	others	p-value
UC/A* ϵ-A(0.0)	ϵ-A(0.25)	0.0078
UC/A* ϵ-A(0.0)	GS/BS(1) ϵ-A(0.5)	0.0005
UC/A* ϵ-A(0.0)	BS(2)	0.0488
UC/A* ϵ-A(0.0)	BS(3)	0.2500

Table 4
Subset 0/1 loss for the methods when a linear classifier (logistic regression) is taken as base method for PCC. Those scores that are equal or lower than the optimal predictions are in bold.

Datasets	UC/A* ϵ-A(0.0)	ϵ-A(0.25)	GS/BS(1) ϵ-A(0.5)	BS(2)	BS(3)
bibtex	**81.92**	81.95	82.19	**81.88**	**81.92**
corel5k	**97.48**	98.62	98.90	98.30	98.04
emotions	**71.16**	71.82	72.83	72.16	71.32
enron	**83.14**	84.26	85.43	83.43	83.37
flags	**87.13**	87.16	**86.13**	88.21	**87.13**
image	**68.35**	**68.35**	69.75	**68.35**	**68.35**
mediamill*	**83.86**	84.58	85.80	84.10	**83.86**
medical	**30.37**	**30.37**	30.67	**30.37**	**30.37**
reuters	**22.73**	**22.70**	23.60	**22.69**	**22.73**
scene	**31.86**	**31.86**	33.28	31.90	**31.86**
slashdot	**51.80**	52.22	54.49	**51.77**	**51.80**
yeast	**76.95**	77.62	79.77	**76.83**	77.08

Table 5
Wilcoxon signed-rank test for the methods when a linear classifier (logistic regression) is taken as base method for PCC.

Optimal methods	others	p-value
UC/A* ϵ-A(0.0)	ϵ-A(0.25)	0.0117
UC/A* ϵ-A(0.0)	GS/BS(1) ϵ-A(0.5)	0.0034
UC/A* ϵ-A(0.0)	BS(2)	0.1309
UC/A* ϵ-A(0.0)	BS(3)	0.1250

Secondly, and with the aim of also including a comparison of the above methods with the A* algorithm using the heuristic h^d_l, we perform experiments taken a linear classifier as base method in PCC. Particularly, logistic regression with probabilistic output [24] was taken to build the binary classifiers f_i. In this case, only the parameter C was optimized, and it was made using a grid search over the values of $C \in \{10^{-3}, 10^{-2}, \ldots, 10^{2}, 10^{3}\}$ using the brier loss [23] as target function estimated through a 2-fold cross validation of five repetitions.

Please cite this article as: D. Mena et al., A heuristic in A* for inference in nonlinear Probabilistic Classifier Chains, Knowledge-Based Systems (2017), http://dx.doi.org/10.1016/j.knosys.2017.03.015

Table 6
Averaged number of nodes explored (top) and averaged computational prediction time in milliseconds (bottom). The number in brackets is the number of labels of the dataset.

Datasets	h^1_{nl} h^1_l	h^2_{nl}	h^3_{nl}	h^2_l	h^3_l	ϵ-A(0.0)	ϵ-A(0.25)	GS/BS(1) ϵ-A(0.5)	BS(2)	BS(3)
bibtex (159)	283.3	277.7	272.4	278.0	273.2	289.3	184.0	160.0	319.0	477.0
corel5k (374)	1456.9	1440.6	1425.4	1443.1	1431.6	1474.2	517.1	375.0	749.0	1122.0
emotions (6)	9.1	8.0	7.4	8.3	7.8	10.7	10.8	7.0	13.0	18.0
enron (53)	110.3	106.1	102.2	106.5	103.3	114.8	77.3	54.0	107.0	159.0
flags (7)	16.4	10.2	8.7	12.7	10.4	22.6	16.3	8.0	15.0	21.0
image (5)	6.6	6.1	6.0	6.3	6.1	7.3	7.7	6.0	11.0	15.0
mediamill* (101)	188.2	184.8	181.5	185.6	183.7	191.8	142.4	102.0	203.0	303.0
medical (45)	46.6	46.6	46.5	46.6	46.6	46.6	46.6	46.0	91.0	135.0
reuters (7)	8.2	8.1	8.0	8.1	8.1	8.2	8.3	8.0	15.0	21.0
scene (6)	7.2	7.1	7.0	7.2	7.1	7.2	7.3	7.0	13.0	18.0
slashdot (22)	25.0	24.7	24.4	24.9	24.9	25.3	24.9	23.0	45.0	66.0
yeast (14)	23.6	21.4	20.0	22.8	22.4	26.1	26.0	15.0	29.0	42.0
bibtex (159)	19.5	24.8	30.2	31.4	41.1	16.2	9.9	7.9	11.0	15.6
corel5k (374)	168.4	190.8	229.4	257.8	333.3	136.0	16.1	11.0	27.8	40.4
emotions (6)	0.6	0.7	0.7	0.8	0.9	0.6	0.6	0.4	0.5	0.6
enron (53)	8.5	14.5	18.4	14.1	18.5	7.2	3.0	1.9	3.9	5.5
flags (7)	1.1	0.9	0.9	1.4	1.5	1.2	0.7	0.4	0.5	0.7
image (5)	0.5	0.5	0.6	0.6	0.7	0.5	0.5	0.4	0.4	0.5
mediamill* (101)	14.4	18.9	23.3	24.2	32.6	11.5	5.5	3.7	7.2	10.5
medical (45)	3.3	4.6	5.7	5.7	7.7	2.7	2.7	2.7	3.3	4.6
reuters (7)	0.6	0.7	0.8	0.8	1	0.5	0.5	0.5	0.5	0.7
scene (6)	0.5	0.6	0.7	0.7	0.9	0.5	0.5	0.4	0.5	0.6
slashdot (22)	1.8	2.4	2.9	2.9	3.9	1.5	1.4	1.2	1.6	2.2
yeast (14)	1.6	2.0	2.3	2.6	3.4	1.5	1.0	0.6	1.0	1.4

Tables 4 and 5 present the subset 0/1 loss estimated by a 10-fold cross validation for all the methods (including this time the A* with the heuristic h^d_l) and the p-values of the Wilcoxon signed-rank test. Again, the approaches UC, ϵ-A(0.0) and the A* algorithm using whatever admissible heuristic provide optimal solutions in terms of the subset 0/1 loss (see the scores in the first column of the table). In this case, there is just one case (reuters) out of 12 for which ϵ-A(0.25) obtains a better score than the algorithms which theoretically obtain optimal solutions and 3 cases for which it equals them. Also, GS (or BS(1) or ϵ-A(0.5)) improves the optimal algorithms in one case (reuters). BS(2) does it in 4 cases (bibtex, reuters, slashdot and yeast) and equals them in 2 cases. Notice that the wins achieved by BS(2) over optimal methods are much smaller than its losses. Interestingly, these wins, and most of the losses, disappear when $b = 3$, showing that BS converges to the optimal solution when b increases. In fact, BS(3) gets equal results in 8 cases and it is never better. The Wilcoxon signed-rank test in this case provides the same conclusions drawn when using nonlinear classifiers as base methods, but the p-values are slightly greater. The difference continues being significant for ϵ-A(0.25) at level of $\alpha = 0.05$ and for GS (or BS(1) or ϵ-A(0.5)) with $\alpha = 0.01$. The differences with respect to BS(2) and BS(3) are not significant.

Comparing the use of linear and nonlinear classifiers as base methods in PCC, there are 8 datasets out of 12 in which using nonlinear classifiers as base methods improves the subset 0/1 scores with regard to applying linear classifiers. Therefore, as expected, this means that there are problems that require more complex models. This in turn supports the usefulness of the method proposed in this paper because it is the unique method based on the A* algorithm able to perform inference with nonlinear PCC.

5.2. Computational analysis

Let now study both heuristics (h^d_{nl} and h^d_l) in terms of number of nodes explored and computational time. In this case, we restrict the study using just linear classifiers (logistic regression) as base methods because h^d_l cannot be applied with nonlinear models. The rest of the approaches are also included to provide a broader comparison.

Table 6 shows the number of nodes explored by each method and the average time spent in performing the predictions taken in milliseconds. Obviously, GS (or ϵ-A(0.5) or BS(1)) explores the least number of nodes, since it only follows one path in the tree, then, it explores many nodes as labels have the dataset (plus one if the root is also considered). Among the methods that do not guarantee to obtain optimal predictions, ϵ-A(0.25) tends to expand fewer nodes that BS(2), but at expenses of offering worst scores in terms of the subset 0/1 loss. BS(3) expands the most number of nodes, in spite of getting predictions close to be optimal. Among the methods that obtain optimal solutions, the algorithm A* using h^d_{nl} is the method that explores the fewest number of nodes for all values of d, as theoretically expected. In fact, it was shown that h^d_{nl} is the most dominant heuristic. Besides, it is more dominant as the level of depth d increases. Remarkably, the number of nodes explored by A* using h^d_{nl} is close to that of GS in some situations. However, sometimes the number of nodes explored by A* using h^d_{nl} for $d > 0$ is quite similar to that of ϵ-A(0.0), UC or A* using $h = 1$. The reason may be because the probabilities provided by the classifiers f_i take values close to 0 or 1, then, there may not be so much uncertainty and the search may be highly directed.

Concerning computational time, it is quite curious that ϵ-A(0.0) and UC are not so much slower than ϵ-A(0.25) and ϵ-A(0.5). Besides, they are quite similar to BS(2) and BS(3). Also, A* using h^d_{nl} with $d > 0$ is not as faster as ϵ-A(0.0) (or A* using h^0_{nl}), in spite of exploring fewer nodes, due to the computational time spent in evaluating the heuristic. However, the difference is quite low in comparison with the heuristic h^d_l. At this respect, let us include some noise in the dataset in order to cause uncertainty and disturb the search and then, avoiding it being so direct.

Figs. 6 and 7 show the number of nodes expanded and the computational time employed in milliseconds for ϵ−A with $\epsilon = 0.0$ and for A* using h^d_l and h^d_{nl} for different values of the parameter d (1, 2, 3) when a percentage of noise ranging from 0% to 26% is included in the datasets. Only in case of the datasets bibtex, mediamill and enron, the percentage of noise added has had to be respectively limited to 4%, 8% and 10%, due to the high computational time involved. At sight of these figures, one can observe

Please cite this article as: D. Mena et al., A heuristic in A* for inference in nonlinear Probabilistic Classifier Chains, Knowledge-Based Systems (2017), http://dx.doi.org/10.1016/j.knosys.2017.03.015

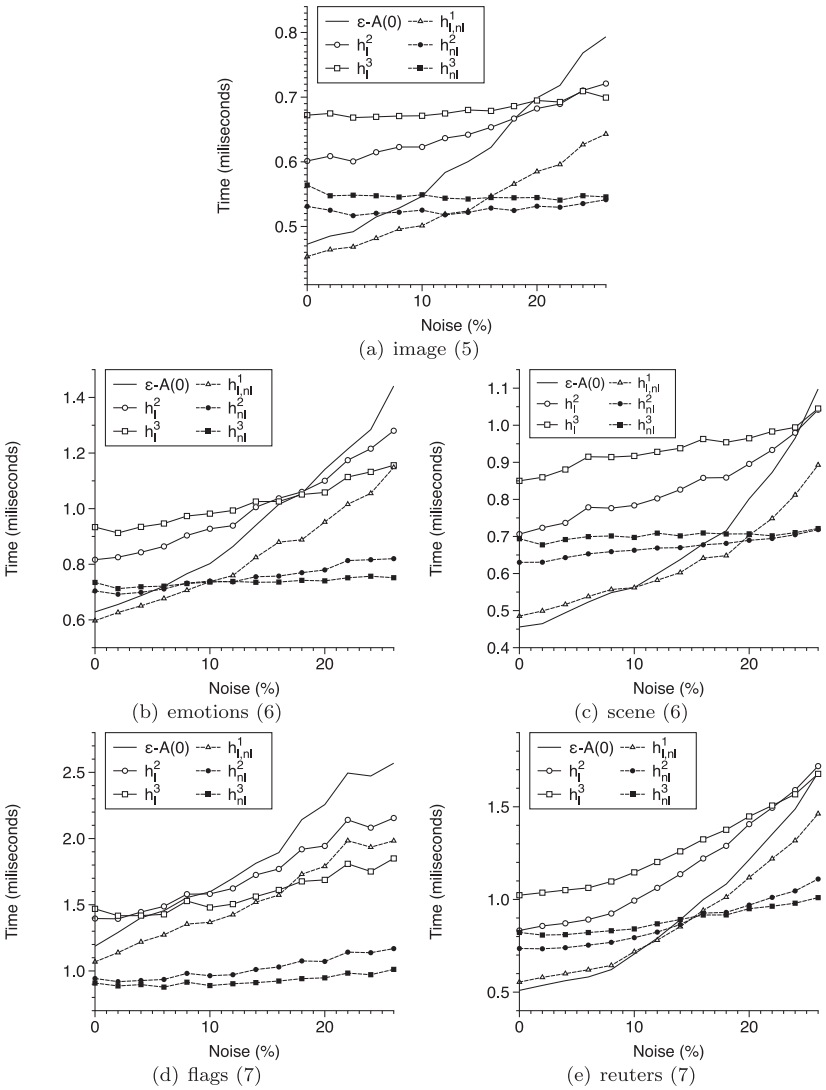


Fig. 6. Computational time employed (ms) for ϵ–A with $\epsilon = 0.0$ and for A* using h_l^d and h_{nl}^d for different values of the parameter d (1, 2, 3) when certain percentage of noise is included in the datasets. Notice that for $d = 1$, the results are equal, since they follow the same path. Different percentages of noise are considered. All the datasets have few labels (from 5 to 7).

that using the heuristic h_{nl}^d hardly increases the computational time when the percentage of noise increases in case of datasets with few labels, being the one which presents the least increasing of computational time for datasets with high number of labels. In this last case, that is, when the increasing of the noise results in a considerable increasing in computational time because more nodes must be expanded to obtain an optimal solution is when the heuristic h_{nl}^d clearly shows its main benefit. It is quite remarkable that computational time spent by h_{nl}^d is always lower than the computational time spent by the heuristic h_l^d for the same level of depth. Hence, unlike h_l^d, the time spent in computing h_{nl}^d is not a disadvantage, but quite the opposite.

6. Conclusions

This paper proposes an admissible heuristic for the algorithm A* for performing inference in PCC either for linear and nonlinear classifiers as base methods. The heuristic is based on an exhaustive search, but including a depth parameter that establishes a balance between the number of nodes explored and the computational time. The method is theoretically shown to provide optimal solutions in terms of the subset 0/1 loss, since the heuristic is admissible. Besides, the heuristic is more dominant than other existing heuristics in the literature for the same purpose, then, it explores fewer nodes among the methods that guarantee to pro-

Please cite this article as: D. Mena et al., A heuristic in A* for inference in nonlinear Probabilistic Classifier Chains, Knowledge-Based Systems (2017), http://dx.doi.org/10.1016/j.knosys.2017.03.015

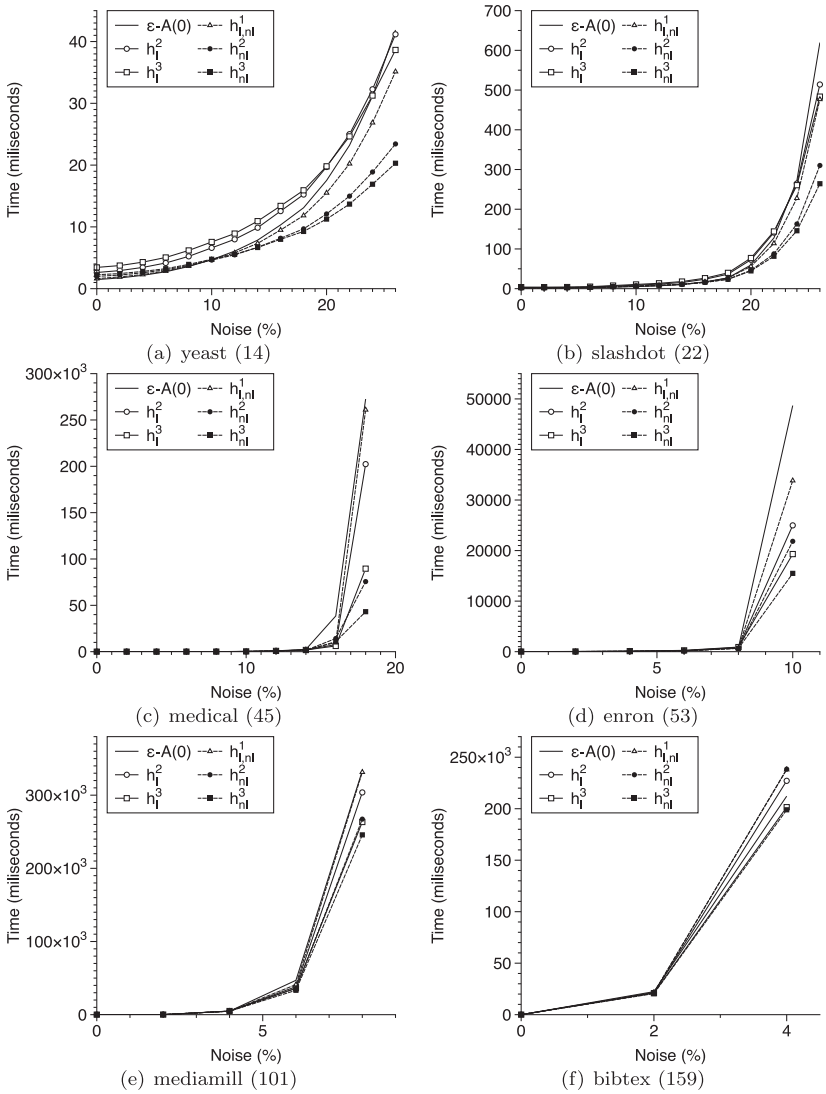


Fig. 7. Computational time employed (ms) for $\epsilon-$A with $\epsilon = 0.0$ and for A* using h_l^d and h_{nl}^d for different values of the parameter d (1, 2, 3) when certain percentage of noise is included in the datasets. Notice that for $d = 1$, the results are equal, since they follow the same path. Different percentages of noise are considered. The number of labels range from 14 to 159.

vide optimal solutions. Moreover, it is a suitable heuristic for any kind of probabilistic classifiers, including nonlinear classifiers. One of the key aspect of our proposal is to present an efficient implementation of the heuristic that significantly reduces the number of models evaluated in its computation.

The experiments confirm the theoretical properties of the proposed heuristic. However, the time spent in evaluating the heuristic can be sometimes a disadvantage. This is, for instance, for datasets that cause few uncertainty in the search and, hence, such search is quite directed because the computational time is slightly higher than that of the $\epsilon - A$ with $\epsilon = 0.0$ and hence, the number of nodes explored are not few enough to compensate that time. Conversely, when the search is not so directed, for instance, because the prediction problem is more uncertain, using the heuristic proposed in this paper makes the A* algorithm provide faster predictions than other state-of-the-art methods that also produce Bayes-optimal predictions, like the $\epsilon - A$ algorithm.

Acknowledgments

This research has been funded by MINECO (the Spanish Ministerio de Economía y Competitividad) and FEDER (Fondo Europeo de Desarrollo Regional), grant TIN2015-65069-C2-2-R (MINECO/FEDER).

Please cite this article as: D. Mena et al., A heuristic in A* for inference in nonlinear Probabilistic Classifier Chains, Knowledge-Based Systems (2017), http://dx.doi.org/10.1016/j.knosys.2017.03.015

References

[1] W. Cheng, E. Hüllermeier, Combining instance-based learning and logistic regression for multilabel classification, Mach. Learn. 76 (2–3) (2009) 211–225.
[2] E. Montañés, J.R. Quevedo, J.J. del Coz, Aggregating independent and dependent models to learn multi-label classifiers, in: ECML'11, 2011, pp. 484–500.
[3] E. Montañés, R. Senge, J. Barranquero, J.R. Quevedo, J.J. del Coz, E. Hüllermeier, Dependent binary relevance models for multi-label classification, Pattern Recognit. 47 (3) (2014) 1494–1508.
[4] J. Read, B. Pfahringer, G. Holmes, E. Frank, Classifier chains for multi-label classification, Mach. Learn. 85 (3) (2011) 333–359.
[5] G. Tsoumakas, I. Vlahavas, Random k-Labelsets: an Ensemble Method for Multilabel Classification, in: ECML/PKDD'07, Springer, 2007, pp. 406–417.
[6] Q. Wu, M.K. Ng, Y. Ye, X. Li, R. Shi, Y. Li, Multi-label collective classification via markov chain based learning method, Knowl. Based Syst. 63 (2014) 1–14.
[7] J.-J. Zhang, M. Fang, J.-Q. Wu, X. Li, Robust label compression for multi-label classification, Knowl. Based Syst. 107 (2016) 32–42, doi:10.1016/j.knosys.2016.05.051. URL www.sciencedirect.com/science/article/pii/S0950705116301563.
[8] K. Dembczyński, W. Waegeman, W. Cheng, E. Hüllermeier, On label dependence and loss minimization in multi-label classification, Mach. Learn. 88 (1–2) (2012) 5–45.
[9] K. Dembczyński, W. Cheng, E. Hüllermeier, Bayes optimal multilabel classification via probabilistic classifier chains, in: ICML, 2010, 2010, pp. 279–286.
[10] K. Dembczynski, W. Waegeman, E. Hüllermeier, An analysis of chaining in multi-label classification., in: ECAI, in: Frontiers in Artificial Intelligence and Applications, vol. 242, IOS Press, 2012, pp. 294–299.
[11] A. Kumar, S. Vembu, A.K. Menon, C. Elkan, Beam search algorithms for multilabel learning, Mach. Learn. 92 (1) (2013) 65–89.
[12] J. Read, L. Martino, D. Luengo, Efficient monte carlo methods for multi-dimensional learning with classifier chains, Pattern Recognit. 47 (3) (2014) 1535–1546.
[13] D. Mena, E. Montañés, J.R. Quevedo, J.J. del Coz, Using a* for inference in probabilistic classifier chains, in: IJCAI 2015, 2015, pp. 3707–3713.
[14] D. Mena, E. Montañés, J.R. Quevedo, J.J. del Coz, A family of admissible heuristics for a* to perform inference in probabilistic classifier chains, Mach. Learn. 106 (1) (2017) 143–169, doi:10.1007/s10994-016-5593-5.
[15] K. Dembczyński, W. Waegeman, E. Hüllermeier, An analysis of chaining in multi-label classification, in: ECAI 2012, 2012, pp. 294–299.
[16] J. Read, B. Pfahringer, G. Holmes, E. Frank, Classifier chains for multi-label classification, in: ECML/PKDD'09, in: LNCS, Springer, 2009, pp. 254–269.
[17] R. Senge, J.J. del Coz, E. Hüllermeier, On the problem of error propagation in classifier chains for multi-label classification, in: Conference of the German Classification Society on Data Analysis, Machine Learning and Knowledge Discovery, 2012, 2012.
[18] R. Senge, J.J. del Coz, E. Hüllermeier, Rectifying classifier chains for multi-label classification, Learning, Knowledge, Adaptation 2013, 2013.
[19] D. Mena, E. Montañés, J.R. Quevedo, J.J. del Coz, An overview of inference methods in probabilistic classifier chains for multilabel classification, Wiley Interdiscip. Rev. 6 (6) (2016) 215–230, doi:10.1002/widm.1185.
[20] A. Kumar, S. Vembu, A.K. Menon, C. Elkan, Learning and inference in probabilistic classifier chains with beam search, in: ECML/PKDD 2012, 2012, pp. 665–680.
[21] J. Pearl, Heuristics: Intelligent Search Strategies for Computer Problem Solving, Addison-Wesley Longman, Boston, MA, USA, 1984.
[22] S.J. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, second ed., Pearson Education, 2003.
[23] G.W. Brier, Verification of forecasts expressed in terms of probability, Mon. Weather Rev. 78 (1) (1950) 1–3.
[24] C.-J. Lin, R.C. Weng, S.S. Keerthi, Trust region Newton method for logistic regression, J. Mach. Learn. Res. 9 (April) (2008) 627–650.

Please cite this article as: D. Mena et al., A heuristic in A* for inference in nonlinear Probabilistic Classifier Chains, Knowledge-Based Systems (2017), http://dx.doi.org/10.1016/j.knosys.2017.03.015

Capítulo 6

Informe sobre la calidad de las publicaciones

En este capítulo se presenta un informe sobre la calidad de las revistas donde se publicaron los artículos del capítulo anterior. La información que fue extraída del sitio web InCites^{TM} Journal Citatation Reports de Thomson Reuters, incluyendo estadísticas solo hasta el año 2015, debido a que los años siguientes aún no están disponibles. Para cada revista se muestran tres datos:

(i) Los datos de citación del artículo que se publicó en la revista.

(ii) Un cuadro titulado *Key Indicators* que contiene información de varios indicadores de la revista en los últimos años. Incluyendo, el total de citaciones por año, el índice de impacto por año, el índice de impacto de 5 años, entre otros.

(iii) Un cuadro titulado *JCR Impact Factor* que incluye datos de la posición (o *ranking*) de la revista por categoría(s).

6.1. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery

Artículo:

- D. Mena, E. Montañés, J. R. Quevedo, and J. J. Coz, "An overview of inference methods in probabilistic classifier chains for multilabel classification," Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, vol. 6, no. 6, pp. 215–230, 2016.

Key Indicators

Year	Total Cites	Journal Impact Factor	Impact Factor Without Journal Self Cites	5 Year Impact Factor	Immediacy Index	Citable Items	Cited Half-Life	Citing Half-Life	Eigenfacto Score	Article Influence Score	% Articles in Citable Items	Normalizec Eigenfacto	Average JIF Percentile
2015	297	1.759	1.703	2.185	0.105	19	3.6	7.3	0.00158	0.774	42.11	0.18010	71.969
2014	217	1.594	1.328	1.954	0.154	26	3.0	7.3	0.00125	0.721	19.23	0.13970	69.423
2013	114	1.358	1.098	1.358	0.143	28	2.2	7.7	0.00053	0.390	46.43	0.05796	63.999
2012	73	1.422	1.200	1.422	0.250	36	Not A…	7.4	0.00034	0.438	58.33	Not A…	68.098

JCR Impact Factor

JCR Year	COMPUTER SCIENCE, ARTIFICIAL INTELLIGENCE			COMPUTER SCIENCE, THEORY & METHODS		
	Rank	Quartile	JIF Percentile	Rank	Quartile	JIF Percentile
2015	48/130	Q2	63.462	21/105	Q1	80.476
2014	51/123	Q2	58.943	21/102	Q1	79.902
2013	55/121	Q2	54.959	28/102	Q2	73.039
2012	48/115	Q2	58.696	23/100	Q1	77.500

6.2. Machine Learning

Artículo:

- D. Mena, E. Montañés, J. R. Quevedo, and J. J. del Coz, "A family of admissible heuristics for A* to perform inference in probabilistic classifier chains," Machine Learning, vol. 106, no. 1, pp. 143–169, 2017.

Key Indicators

Year	Total Cites	Journal Impact Factor	Impact Factor Without Journal Self Cites	5 Year Impact Factor	Immediacy Index	Citable Items	Cited Half-Life	Citing Half-Life	Eigenfacto Score	Article Influence Score	% Articles in Citable Items	Normalizec Eigenfacto	Average JIF Percentile
2015	9,656	1.719	1.649	2.454	0.227	75	>10.0	9.6	0.00608	1.473	100.00	0.69315	60.385
2014	8,649	1.889	1.814	2.636	0.211	57	>10.0	8.4	0.00719	1.714	100.00	0.80484	65.447
2013	7,363	1.689	1.533	2.289	0.123	57	>10.0	7.5	0.00667	1.609	100.00	0.73567	64.050
2012	7,004	1.454	1.370	2.134	0.373	51	>10.0	7.7	0.00636	1.523	100.00	Not A…	59.565
2011	5,734	1.587	1.509	2.038	0.098	51	>10.0	8.3	0.00615	1.426	100.00	Not A…	66.216
2010	6,310	1.967	1.888	2.664	0.214	56	>10.0	7.3	0.00556	1.303	100.00	Not A…	69.907
2009	7,305	1.663	1.554	4.099	0.896	48	>10.0	7.2	0.00768	1.740	100.00	Not A…	60.680
2008	6,211	2.326	2.213	4.254	0.595	42	>10.0	7.8	0.01068	2.263	100.00	Not A…	77.128
2007	4,941	1.742	1.707	4.513	0.098	41	>10.0	8.5	0.01081	2.058	100.00	Not A…	76.882
2006	4,483	2.654	2.543	Not A…	0.479	48	9.5	7.7	Not A…	Not A…	100.00	Not A…	93.529
2005	3,568	3.108	3.024	Not A…	0.268	41	8.7	6.9	Not A…	Not A…	100.00	Not A…	93.038
2004	3,610	3.258	3.164	Not A…	0.925	40	8.0	6.7	Not A…	Not A…	100.00	Not A…	94.231
2003	3,061	3.050	2.910	Not A…	0.651	43	7.4	6.9	Not A…	Not A…	100.00	Not A…	92.857
2002	2,087	1.944	1.766	Not A…	0.852	54	7.3	6.4	Not A…	Not A…	100.00	Not A…	91.216
2001	1,685	1.476	1.369	Not A…	0.362	47	7.7	6.5	Not A…	Not A…	100.00	Not A…	86.806
2000	1,663	1.447	1.305	Not A…	0.209	43	7.6	6.6	Not A…	Not A…	100.00	Not A…	85.211

JCR Impact Factor

JCR Year	COMPUTER SCIENCE, ARTIFICIAL INTELLIGENCE		
	Rank	Quartile	JIF Percentile
2015	52/130	Q2	60.385
2014	43/123	Q2	65.447
2013	44/121	Q2	64.050
2012	47/115	Q2	59.565
2011	38/111	Q2	66.216
2010	33/108	Q2	69.907
2009	41/103	Q2	60.680
2008	22/94	Q1	77.128
2007	22/93	Q1	76.882
2006	6/85	Q1	93.529
2005	6/79	Q1	93.038
2004	5/78	Q1	94.231
2003	6/77	Q1	92.857
2002	7/74	Q1	91.216
2001	10/72	Q1	86.806
2000	11/71	Q1	85.211

6.3. Knowledge-Based Systems

Artículo:

- D. Mena, E. Montañés, J. R. Quevedo, and J. J. Del Coz, "A heuristic in A* for inference in nonlinear Probabilistic Classifier Chains," Knowledge-Based Systems, 2017.

Key Indicators

Year	Total Cites	Journal Impact Factor	Impact Factor Without Journal Self Cites	5 Year Impact Factor	Immediacy Index	Citable Items	Cited Half-Life	Citing Half-Life	Eigenfacto Score	Article Influence Score	% Articles in Citable Items	Normalizec Eigenfacto	Average JIF Percentile
2015	4,811	3.325	2.664	3.433	0.760	333	3.2	7.4	0.01326	0.819	99.10	1.51143	87.308
2014	3,424	2.947	2.626	3.011	0.410	295	3.0	7.9	0.00923	0.681	99.66	1.03395	87.398
2013	2,629	3.058	1.854	2.920	0.573	295	3.0	7.7	0.00666	0.603	99.66	0.73387	88.017
2012	2,375	4.104	1.567	3.371	0.703	249	3.2	7.4	0.00492	0.585	99.20	Not A…	95.217
2011	1,254	2.422	1.100	1.967	0.500	130	3.5	7.2	0.00279	0.354	100.00	Not A…	86.937
2010	914	1.574	1.016	1.454	0.297	101	4.3	7.6	0.00217	0.306	100.00	Not A…	59.722
2009	933	1.308	1.255	1.406	0.139	79	6.3	7.9	0.00177	0.270	100.00	Not A…	49.029
2008	680	0.924	0.793	1.103	0.183	104	6.7	7.9	0.00095	0.170	100.00	Not A…	32.447
2007	403	0.574	0.547	0.716	0.103	68	6.5	7.4	0.00109	0.198	100.00	Not A…	25.269
2006	352	0.576	0.530	Not A…	0.039	77	6.4	8.2	Not A…	Not A…	100.00	Not A…	26.471
2005	309	0.696	0.637	Not A…	0.069	29	6.2	8.4	Not A…	Not A…	100.00	Not A…	37.342
2004	344	0.645	0.634	Not A…	0.130	23	5.4	7.1	Not A…	Not A…	100.00	Not A…	44.231
2003	320	0.842	0.831	Not A…	0.171	41	4.6	6.9	Not A…	Not A…	100.00	Not A…	52.597
2002	195	0.384	0.373	Not A…	0.038	52	5.5	8.4	Not A…	Not A…	100.00	Not A…	35.811
2001	175	0.275	0.252	Not A…	0.041	49	6.3	6.7	Not A…	Not A…	100.00	Not A…	21.528
2000	158	0.375	0.307	Not A…	0.020	50	5.5	4.9	Not A…	Not A…	100.00	Not A…	34.507

JCR Impact Factor

JCR Year	COMPUTER SCIENCE, ARTIFICIAL INTELLIGENCE		
	Rank	Quartile	JIF Percentile
2015	17/130	Q1	87.308
2014	16/123	Q1	87.398
2013	15/121	Q1	88.017
2012	6/115	Q1	95.217
2011	15/111	Q1	86.937
2010	44/108	Q2	59.722
2009	53/103	Q3	49.029
2008	64/94	Q3	32.447
2007	70/93	Q4	25.269
2006	63/85	Q3	26.471
2005	50/79	Q3	37.342
2004	44/78	Q3	44.231
2003	37/77	Q2	52.597
2002	48/74	Q3	35.811
2001	57/72	Q4	21.528
2000	47/71	Q3	34.507

Bibliografía

Bassam Al-Salemi, Shahrul Azman Mohd Noah, and Mohd Juzaiddin Ab Aziz. Rfboost: an improved multi-label boosting algorithm and its application to text categorisation. *Knowledge-Based Systems*, 103:104–117, 2016.

Everton Alvares Cherman, Jean Metz, and Maria Monard. A Simple Approach to Incorporate Label Dependency in Multi-label Classification. In Grigori Sidorov, Arturo Hernández Aguirre, and Carlos Reyes García, editors, *Advances in Soft Computing*, volume 6438 of *Lecture Notes in Computer Science*, chapter 3, pages 33–43. Springer, 2010. ISBN 978-3-642-16772-0.

Martin Antenreiter, Ronald Ortner, and Peter Auer. Combining classifiers for improved multilabel image classification. *Learning from Multilabel Data MLD Workshop at ECML 2009*, 2009.

A. Asuncion and D.J. Newman. Uci machine learning repository, 2007.

Glenn W. Brier. Verification of forecasts expressed in terms of probability. *Monthly weather review*, 78(1):1–3, January 1950.

C.C. Chang and C.J. Lin. *LIBSVM: a library for support vector machines*, 2001.

W. Cheng and E. Hüllermeier. Combining instance-based learning and logistic regression for multilabel classification. *Machine Learning*, 76(2-3):211–225, 2009.

A. Clare and R. D. King. Knowledge discovery in multi-label phenotype data. In *European Conference on Data Mining and Knowledge Discovery (2001)*, pages 42–53, 2001. ISBN 3-540-42534-9.

Peter R. de Waal and Linda C. van der Gaag. Inference and learning in multi-dimensional bayesian network classifiers. In Khaled Mellouli, editor, *Symbolic and Quantitative Approaches to Reasoning with Uncertainty, 9th European Conference, ECSQARU 2007, Hammamet, Tunisia, October 31 - November 2, 2007, Proceedings*, volume 4724 of *Lecture Notes in Computer Science*, pages 501–511. Springer, 2007. ISBN 978-3-540-75255-4.

K. Dembczyński, W. Cheng, and E. Hüllermeier. Bayes Optimal Multilabel Classification via Probabilistic Classifier Chains. In *ICML, 2010*, pages 279–286, 2010a.

K. Dembczyński, W. Waegeman, W. Cheng, and E. Hüllermeier. Regret analysis for performance metrics in multi-label classification: the case of hamming and subset zero-one loss. In *ECML/PKDD'10, Part I*, pages 280–295. Springer, 2010b. ISBN 3-642-15879-X, 978-3-642-15879-7.

K. Dembczyński, W. Waegeman, W. Cheng, and E. Hüllermeier. On label dependence and loss minimization in multi-label classification. *Machine Learning*, 88(1-2):5–45, 2012.

K. Dembczynski, W. Waegeman, and E. Hüllermeier. An analysis of chaining in multi-label classification. In *ECAI*, volume 242 of *Frontiers in Artificial Intelligence and Applications*, pages 294–299. IOS Press, 2012. ISBN 978-1-61499-097-0.

Krzysztof Dembczyński, Willem Waegeman, and Eyke Hüllermeier. An analysis of chaining in multi-label classification. In *ECAI 2012*, pages 294–299, 2012.

J. Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006. ISSN 1532-4435.

A. Elisseeff and J. Weston. A Kernel Method for Multi-Labelled Classification. In *ACM Conf. on Research and Develop. in Information Retrieval (2005)*, pages 274–281, 2005.

R.E. Fan, K.W. Chang, C.J. Hsieh, X.R. Wang, and C.J. Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9: 1871–1874, 2008.

J. Fürnkranz, E. Hüllermeier, E. Loza Mencía, and K. Brinker. Multilabel classification via calibrated label ranking. *Machine Learning*, 73:133–153, 2008. ISSN 0885-6125.

S. García and F. Herrera. An extension on statistical comparisons of classifiers over multiple data sets for all pairwise comparisons. *Journal of Machine Learning Research*, 9:2677–2694, 2008.

N. Ghamrawi and A. McCallum. Collective multi-label classification. In *ACM Int. Conf. on Information and Knowledge Management*, pages 195–200. ACM, 2005. ISBN 1-59593-140-6.

Eva Gibaja and Sebastián Ventura. Multi-label learning: a review of the state of the art and ongoing research. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 4(6):411–444, 2014.

S. Godbole and S. Sarawagi. Discriminative methods for multi-labeled classification. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining (2004)*, pages 22–30, 2004.

Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, SSC-4(2):100–107, 1968.

Ioannis Katakis, Grigorios Tsoumakas, and Ioannis Vlahavas. Multilabel Text Classification for Automated Tag Suggestion. In *Proceedings of the ECML/PKDD 2008 Discovery Challenge*, 2008.

Abhishek Kumar, Shankar Vembu, Aditya Krishna Menon, and Charles Elkan. Learning and inference in probabilistic classifier chains with beam search. In *ECML/PKDD 2012*, pages 665–680, 2012.

Abhishek Kumar, Shankar Vembu, Aditya Krishna Menon, and Charles Elkan. Beam search algorithms for multilabel learning. *Machine Learning*, 92(1):65–89, 2013. ISSN 0885-6125.

C.-J. Lin, R. C. Weng, and S. S. Keerthi. Trust region Newton method for logistic regression. *Journal of Machine Learning Research*, 9(Apr):627–650, 2008.

Óscar Luaces, Jorge Díez, Jose Barranquero, Juan José del Coz, and Antonio Bahamonde. Binary relevance efficacy for multilabel classification. *Progress in Artificial Intelligence*, 4(1):303–313, 2012.

Gjorgji Madjarov, Dragi Kocev, Dejan Gjorgjevikj, and Sašo Džeroski. An extensive experimental comparison of methods for multi-label learning. *Pattern Recognition*, 45(9):3084–3104, 2012.

A. K. McCallum. Multi-label text classification with a mixture model trained by em. In *AAAI 99 Workshop on Text Learning*, 1999.

Deiner Mena, Elena Montañés, José Ramón Quevedo, and Juan José del Coz. Using a* for inference in probabilistic classifier chains. In *IJCAI 2015*, pages 3707–3713, 2015a.

Deiner Mena, Elena Montañés, José Ramón Quevedo, and Juan José del Coz. Heurístico exhaustivo de profundidad limitada para cc probabilístico. In *Actas de la XVI Conferencia de la Asociación Española para la Inteligencia Artificial, CAEPIA 2015, Albacete, Noviembre 9-12, 2015*, pages 801–810, 2015b.

Deiner Mena, Elena Montañés, José Ramón Quevedo, and Juan José del Coz. An overview of inference methods in probabilistic classifier chains for multilabel classification. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 6(6):215–230, 2016. ISSN 1942-4795. doi: 10.1002/widm.1185. URL http://dx.doi.org/10.1002/widm.1185.

Deiner Mena, Elena Montañés, José Ramón Quevedo, and Juan José del Coz. A heuristic in a* for inference in nonlinear probabilistic classifier chains. *Knowledge-Based Systems*, 2017a. ISSN 0950-7051. doi: 10.1016/j.knosys.2017.03.015. URL http://dx.doi.org/10.1016/j.knosys.2017.03.015.

Deiner Mena, Elena Montañés, José Ramón Quevedo, and Juan José del Coz. A family of admissible heuristics for a* to perform inference in probabilistic classifier chains. *Machine Learning*, 106(1):143–169, 2017b. ISSN 1573-0565. doi: 10.1007/s10994-016-5593-5. URL http://dx.doi.org/10.1007/s10994-016-5593-5.

E. Montañés, J.R. Quevedo, and J. J. del Coz. Aggregating independent and dependent models to learn multi-label classifiers. In *ECML'11*, pages 484–500, 2011a. ISBN 978-3-642-23782-9.

E. Montañés, R. Senge, J. Barranquero, J.R. Quevedo, J. J. del Coz, and E. Hüllermeier. Dependent binary relevance models for multi-label classification. *Pattern Recognition*, 47(3):1494 – 1508, 2014. ISSN 0031-3203.

Elena Montañés, José Ramón Quevedo, and Juan José del Coz. Improving stacking approach for multi-label classification. In *Proceedings of the 2011 Spanish Conference on Artificial Intelligence*, pages 484–500, 2011b.

F. Pachet and P. Roy. Improving Multilabel Analysis of Music Titles: A Large-Scale Validation of the Correction Approach. *IEEE Transactions on Audio, Speech, and Language Processing*, 17(2):335–343, 2009. ISSN 1558-7916.

Judea Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1984a. ISBN 0-201-05594-5.

Judea Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley Longman, Boston, MA, USA, 1984b. ISBN 0-201-05594-5.

Guo J. Qi, Xian S. Hua, Yong Rui, Jinhui Tang, Tao Mei, and Hong J. Zhang. Correlative multi-label video annotation. In *Proceedings of the International Conference on Multimedia*, pages 17–26, NY, USA, 2007. ACM. ISBN 978-1-59593-702-5.

J. Read, B. Pfahringer, and G. Holmes. Multi-label classification using ensembles of pruned sets. In *IEEE Int. Conf. on Data Mining*, pages 995–1000. IEEE, 2008. ISBN 978-0-7695-3502-9.

J. Read, B. Pfahringer, G. Holmes, and E. Frank. Classifier chains for multi-label classification. In *ECML/PKDD'09*, LNCS, pages 254–269. Springer, 2009.

J. Read, B. Pfahringer, G. Holmes, and E. Frank. Classifier chains for multi-label classification. *Machine Learning*, 85(3):333–359, 2011.

Jesse Read, Luca Martino, and David Luengo. Efficient monte carlo methods for multi-dimensional learning with classifier chains. *Pattern Recognition*, 47(3): 1535 – 1546, 2014. ISSN 0031-3203.

Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition, 2003. ISBN 0137903952.

Robert E. Schapire and Yoram Singer. Boostexter: A boosting-based system for text categorization. *Machine Learning*, pages 135–168, 2000.

R. Senge, J. J. del Coz, and E. Hüllermeier. On the problem of error propagation in classifier chains for multi-label classification. In *Conference of the German Classification Society on Data Analysis, Machine Learning and Knowledge Discovery, 2012*, 2012.

Robin Senge, Juan José del Coz, and Eyke Hüllermeier. Rectifying classifier chains for multi-label classification. In *Learning, Knowledge, Adaptation 2013*, 2013.

Muhammad Tahir, Josef Kittler, Krystian Mikolajczyk, and Fei Yan. Improving Multilabel Classification Performance by Using Ensemble of Multi-label Classifiers. In Neamat Gayar, Josef Kittler, and Fabio Roli, editors, *Multiple Classifier Systems*, volume 5997, chapter 2, pages 11–21. Springer, Berlin, Heidelberg, 2010. ISBN 978-3-642-12126-5.

G. Tsoumakas and I. Katakis. Multi label classification: An overview. *International Journal of Data Warehouse and Mining*, 3(3):1–13, 2007.

G. Tsoumakas and I. Vlahavas. Random k-Labelsets: An Ensemble Method for Multilabel Classification. In *ECML/PKDD'07*, pages 406–417. Springer, 2007. ISBN 978-3-540-74957-8.

G. Tsoumakas, I. Katakis, and I. Vlahavas. Effective and Efficient Multilabel Classification in Domains with Large Number of Labels. In *Proc. ECML/PKDD 2008 Workshop on Mining Multidimensional Data (MMD'08)*, 2008.

G. Tsoumakas, A. Dimou, E. Spyromitros, V. Mezaris, I. Kompatsiaris, and I. Vlahavas. Correlation-based pruning of stacked binary relevance models for multi-label learning. In *Workshop on Learning from Multi-Label Data (2009)*, pages 101–116, 2009.

G. Tsoumakas, I. Katakis, and I. Vlahavas. Mining multi-label data. In *Data Mining and Knowledge Discovery Handbook*, pages 667–685. 2010a.

Grigorios Tsoumakas, Jozef Vilcek, and Lefteris Spyromitros. Mulan: A java library for multi-label learning (http://mulan.sourceforge.net), 2010b.

Linda C. van der Gaag and Peter R. de Waal. Multi-dimensional bayesian network classifiers. In *Probabilistic Graphical Models*, pages 107–114, 2006.

V.N. Vapnik. *The nature of statistical learning theory*. Springer-Verlag New York, Inc., New York, NY, USA, 1995. ISBN 0387945598. URL http://portal.acm.org/citation.cfm?id=211359.

Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005. ISBN 0120884070.

D. H. Wolpert. Stacked generalization. *Neural Networks*, 5:214–259, 1992.

Qingyao Wu, Michael K Ng, Yunming Ye, Xutao Li, Ruichao Shi, and Yan Li. Multi-label collective classification via markov chain based learning method. *Knowledge-Based Systems*, 63:1–14, 2014.

Julio H. Zaragoza, Luis Enrique Sucar, Eduardo F. Morales, Concha Bielza, and Pedro Larrañaga. Bayesian chain classifiers for multidimensional classification. In *IJCAI'11*, pages 2192–2197, 2011.

Ju-Jie Zhang, Min Fang, Jin-Qiao Wu, and Xiao Li. Robust label compression for multi-label classification. *Knowledge-Based Systems*, 107:32 – 42, 2016. ISSN 0950-7051. doi: http://dx.doi.org/10.1016/j.knosys.2016.05.051. URL //www.sciencedirect.com/science/article/pii/S0950705116301563.

M.-L. Zhang and Z.-H. Zhou. Multilabel neural networks with applications to functional genomics and text categorization. *IEEE Trans. on Knowl. and Data Eng.*, 18:1338–1351, 2006. ISSN 1041-4347.

M.-L. Zhang and Z.-H. Zhou. Ml-knn: A lazy learning approach to multi-label learning. *Pattern Recognition*, 40(7):2038–2048, 2007. ISSN 0031-3203.

Min-Ling Zhang and Kun Zhang. Multi-label learning by exploiting label dependency. In Bharat Rao, Balaji Krishnapuram, Andrew Tomkins, and Qiang Yang, editors, *KDD*, pages 999–1008. ACM, 2010. ISBN 978-1-4503-0055-1.

M.L. Zhang and Z. Zhou. M3MIML: A maximum margin method for multi-instance multi-label learning. In *Eighth IEEE International Conference on Data Mining, 2008. ICDM'08*, pages 688–697, 2008.

X. Zhu and X. Wu. Class noise vs. attribute noise: a quantitative study of their impacts. *Artificial Intelligence Review*, 22(3):177–210, 2004. ISSN 0269-2821.