UNIVERSITAT POLITÈCNICA DE VALÈNCIA

TRABAJO FIN DE MÁSTER

Máster en Ingeniería del Software, Métodos Formales y Sistemas de Información

---

CLASIFICACIÓN MULTI-ETIQUETA EN ENTORNOS JERÁRQUICOS

Autor: José Luis de la Cruz Garrido

Tutor: Cèsar Ferri Ramírez

Valencia, 15 de septiembre de 2014

Agradecimientos

En primer lugar, quiero agradecer a mi tutor Cèsar Ferri, la confianza que ha depositado en mí, su paciencia, su apoyo y su empuje cuando yo lo daba todo por perdido. Gran parte del mérito de que este trabajo haya sido concluido con éxito es tuyo. Muchas gracias.

Quiero agradecer a Fani, el haber estado ahí, llevándose la parte más dura de este trabajo. No ha sido fácil para mi compatibilizar la vida laboral con la realización de este máster. Sin tu apoyo y tu comprensión no hubiera sido posible.

Agradecer a toda mi familia, en especial, a mi madre, a mi hermana y a Juanse, su apoyo, sus enseñanzas, sus consejos y todo lo que me han aportado, me aportan y me aportarán. Sois uno de los pilares más importantes de mi vida. Mención especial merecen también mis abuelos José y Lola, ejemplo de esfuerzo, dedicación y humildad.

Gracias a todos los compañeros y profesores del máster por hacer de él una gran experiencia en mi vida.

Resumen

El volumen de información almacenada ha ido creciendo exponencialmente en los últimos años. En este contexto, en el que se calcula que cada 20 meses se duplica la cantidad de información almacenada, es necesario disponer de mecanismos que permitan no sólo almacenar y recuperar de una manera adecuada los datos, sino también procedimientos que nos permitan obtener conocimiento de estos datos, pasando estos a ser útiles.

La Minería de Datos trata de resolver este problema, siendo su objetivo extraer conocimiento de grandes volúmenes de datos [20], dándole sentido al hecho de almacenar toda la información posible y no eliminar prácticamente nada.

Una de las tareas de la Minería de Datos es la clasificación, que originalmente se definió como la correspondencia existente entre un conjunto de patrones y una clase única. Posteriormente, se observó que existen multitud de casos en los que la restricción de una etiqueta por patrón no se cumple, por ejemplo, la clasificación de textos, imágenes, detección de spam, clasificación de sonidos, problemas de bioinformática y un largo etc. Aquí entra en juego la clasificación multi-etiqueta, que se define como el paradigma de clasificación en el que un patrón puede estar asociado con más de una clase.

En algunos contextos, dichas etiquetas tienen relación entre sí, formando algún tipo de estructura. Dicho paradigma de clasificación es conocido como clasificación jerárquica a nivel genérico y clasificación jerárquica multi-etiqueta cuando se trata de problemas multi-etiqueta en los que existe una jerarquía de clases.

Un ejemplo de clasificación jerárquica multi-etiqueta es Wikipedia, donde los patrones tienen varias etiquetas asociadas y las clases forman una jerarquía.

En este trabajo se realiza un estudio de la clasificación jerárquica multi-etiqueta, se construyen datasets con datos obtenidos mediante la API Rest de Wikipedia y se propone una métrica de clasificación jerárquica que será testeada después mediante experimentos de clasificación multi-etiqueta jerárquica.

Palabras clave: aprendizaje automático, clasificación jerárquica multi-etiqueta, métricas basadas en distancias, análisis de datos

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introduccion

1.1. Motivación

Cada día son más los usuarios que utilizan Internet. Según un estudio del INE [1] (Instituto Nacional de Estadística), en el año 2013, 7 de cada 10 hogares españoles tenían acceso a Internet y 1 de cada 2 españoles se conectaban a internet diariamente. A nivel mundial, se estima que internet tendrá 3000 millones de usuarios a finales de 2014, un 40 % de la población mundial. Además, el paradigma de navegación por la red ha cambiado por completo. Si bien, en los inicios de internet, los internautas prácticamente se limitaban a consumir la información disponible, hoy día, el internauta está continuamente generando datos; ejemplos de ello son, subir fotos a redes sociales, mandar correos electrónicos, escribir un blog, realizar una compra de un producto, escribir una opinión de un servicio consumido, etc.

Por otro lado, fuera de la red de redes, no existe prácticamente acción humana que se conciba sin la presencia de las nuevas tecnologías; nuestro historial médico, nuestro expediente académico, nuestros trámites con la Agencia Tributaria, las transacciones en el último año en nuestra cuenta bancaria, por citar algunas, están almacenadas en medios informáticos. En definitiva, el número de datos generados crece a ritmos espectaculares, no siendo problema el almacenamiento de los mismos.

El desafío se encuentra en ser capaces de sacar la máxima informacion posible de estos datos almacenados, tomando el total protagonismo la minería de datos. La mayoría de decisiones de empresas, organizaciones e instituciones

se basan también en información de experiencias pasadas extraídas de fuentes muy diversas. Las decisiones colectivas suelen tener consecuencias mucho más graves, especialmente económicas, y, recientemente, se deben basar en volúmenes de datos que desbordan la capacidad humana. Un buen o mal análisis de los datos puede suponer el éxito o el fracaso en todos los ámbitos de la vida. Los bancos realizan análisis de datos para saber que inversión es buena o es mala, que clientes van a pagar y cuales no o posibles fraudes con tarjetas de crédito. En computación se usa la minería de datos para filtrado de spam, para evitar phising o para detectar bots, por poner algunos ejemplos. En medicina se aplica minería de datos al diagnóstico clínico.

En definitiva, prácticamente en todos los campos de la vida, la minería de datos está tomando un papel protagonista. Institución que no aplique minería de datos, perderá competitividad, viéndose relegada a un segundo plano, incluso desapareciendo.

Imaginemos que tenemos un supermercado. Si nuestros competidores aplican minería de datos y nosotros no, ellos tendrán un mayor conocimiento sobre los clientes y sus preferencias, sobre las ventas, sobre los proveedores, pudiendo tomar unas decisiones más acertadas que las nuestras, por lo que su ventaja con respecto a nosotros es brutal. Desafortunadamente, los algoritmos de minería de datos no son totalmente precisos, por lo que necesitamos métricas que nos indiquen como de bien (o de mal) está funcionando el algoritmo. Las métricas que tradicionalmente se han utilizado, solo tienen en cuenta el fallo o el acierto del algoritmo, sin valorar lo cerca o lejos que el algoritmo ha estado acertar. De ahí la importancia de establecer métricas basadas en distancias, objetivo de este trabajo. Si somos capaces de establecer una jerarquía de clases o bien definir distancias entre todas las clases, podremos utilizar métricas basadas en estas distancias para medir la precisión de los algoritmos, siendo estas métricas mucho más efectivas que las tradicionales basadas en la matriz de confusión. Teniendo en cuenta todo lo anterior, hemos decidido definir una métrica jerárquica y testearla en un entorno de clasificación multi-etiqueta jerárquico como es Wikipedia.

1.2. Objetivos

Este trabajo se encuentra en el marco de la clasificación multi-etiqueta en general y dentro de la clasificación jerárquica multi-etiqueta en particular.

A nivel teórico, se ha realizado un estudio de la clasificación multi-etiqueta, la clasificación jerárquica multi-etiqueta y se ha definido una métrica para evaluar modelos de clasificación en entornos jerárquicos, en concreto, cuando las categorías objetivo forman un grafo conexo.

A nivel práctico, se aportan unos experimentos de clasificación jerárquica multi-etiqueta, usando la técnica de relevancia binaria que se encuentra dentro de los métodos de transformación de problemas. Estos experimentos nos han permitido utilizar nuestra métrica y validarla experimentalmente.

Los objetivos que hemos definido para este trabajo son los siguientes:

1. Revisión del campo clasificación jerárquica multi-etiqueta.
2. Construcción de datasets con datos obtenidos mediante el API Rest de la Wikipedia.
3. Realización de experimentos de clasificación jerárquica multi-etiqueta con los datasets construidos.
4. Proponer una métrica basadas en distancias en el grafo de categorías de Wikipedia para evaluar la precisión de los modelos generados.
5. Validar experimentalmente los modelos generados y la métrica propuesta.

1.3. Estructura del documento

Este documento se estructura de la siguiente manera:

En el Capítulo 2 se revisa el campo de la clasificación jerárquica multi-etiqueta, en particular, los diferentes métodos existentes para clasificación multi-etiqueta, trabajos relacionados con clasificación jerárquica y clasificación multi-etiqueta, las diferentes métricas para medir la precisión de los modelos y las aplicaciones de la clasificación multi-etiqueta a problemas reales.

En el Capítulo 3 se explican las tecnologías que se han utilizado.

En el Capítulo 4 hablamos de la metodología que hemos seguido para el presente trabajo, como hemos construido los datasets, la transformación realizada a los textos y la nueva métrica planteada para evaluar los modelos en función de las distancias de las etiquetas en el grafo.

En el Capítulo 5 se describen los experimentos realizados, mostrando imágenes de los grafos construidos y tablas con los resultados obtenidos.

En el Capítulo 6 se aportan las conclusiones obtenidas del presente trabajo y se proponen diversas líneas de trabajo futuro que deberían ser abordadas próximamente.

Capítulo 2

Clasificación Jerárquica Multi-etiqueta

En este capítulo clarificaremos el campo de la clasificación multi-etiqueta y de la clasificación jerárquica. Realizaremos previamente una introducción a los campos de la minería de datos y el aprendizaje automático. Posteriormente, se explicará el campo de la clasificación multi-etiqueta y por último la clasificación jerárquica citando brevemente trabajos existentes en la literatura.

2.1. Minería de datos

Definimos la minería de datos como un campo de las ciencias de la computación referido al proceso que intenta descubir patrones en grandes volúmenes de conjunto de datos. En ella convergen diversas áreas, como son las bases de datos, el aprendizaje automático y la estadística. El aprendizaje automático es el encargado de extraer patrones a partir de los datos, utilizando distintos enfoques para ello. [20]

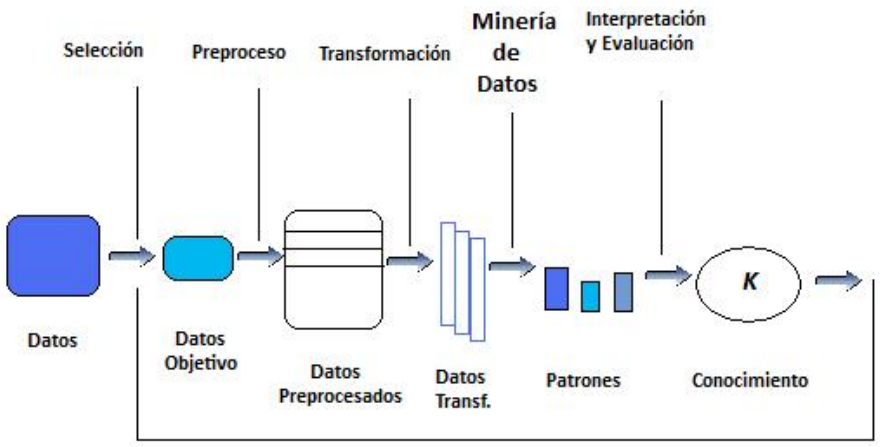


Figura 2.1: Proceso KDD

Un proceso típico de minería de datos consta de las siguientes etapas:

1. Selección del conjunto de datos.

   Consiste en seleccionar el conjunto de datos sobre el que se va a realizar el análisis.

2. Preprocesado de datos.

   Se eliminan el mayor número posible de datos erróneos e inconsistentes.

3. Transformación de los datos.

   En ocasiones, se transforman los datos a un formato más adecuado para el posterior modelado.

4. Modelado.

   Sobre los datos recogidos y preparados, se aplica una técnica de minería de datos para generar un modelo a partir de los datos.

5. Interpretación y evaluación.

   Se validan los modelos generados, por ejemplo, con un dataset independiente de los utilizados para la generación de los mismos.

2.2. Aprendizaje automático

El aprendizaje automático es una rama de la inteligencia artificial cuyo objetivo es desarrollar técnicas que permitan a las computadoras aprender. De forma más concreta, se trata de crear programas capaces de generalizar comportamientos a partir de una información no estructurada suministrada en forma de ejemplos. Es, por lo tanto, un proceso de inducción del conocimiento.

Figura 2.2: Aprendizaje automático

Según el mecanismo que utilice para aprender, puede ser de los siguientes tipos:

1. Aprendizaje supervisado

   El algoritmo produce una función que establece una correspondencia entre las entradas y las salidas deseadas del sistema. Un ejemplo de este tipo de algoritmo es el problema de clasificación, objetivo de este trabajo, donde el sistema de aprendizaje trata de etiquetar (clasificar) una serie de ejemplos utilizando una entre varias categorías (clases). La base de conocimiento del sistema está formada por ejemplos de etiquetados anteriores.

2. Aprendizaje no supervisado

   Todo el proceso de modelado se lleva a cabo sobre un conjunto de ejemplos formado tan sólo por entradas al sistema. No se tiene información

sobre las categorías de esos ejemplos. Por lo tanto, en este caso, el sistema tiene que ser capaz de reconocer patrones para poder etiquetar las nuevas entradas. En nuestro caso, si los ejemplos que forman parte del dataset no estuvieran etiquetados, estaríamos ante un problema de aprendizaje no supervisado.

3. Aprendizaje semisupervisado

   Este tipo de algoritmos combinan los dos algoritmos anteriores para poder clasificar de manera adecuada. Se tiene en cuenta los datos etiquetados y los no etiquetados.

4. Aprendizaje por refuerzo

   El algoritmo aprende observando el mundo que le rodea. Su información de entrada es el feedback o retroalimentación que obtiene del mundo exterior como respuesta a sus acciones.

5. Traducción

   Similar al aprendizaje supervisado, pero no construye de forma explícita una función. Trata de predecir las categorías de los futuros ejemplos basándose en los ejemplos de entrada, sus respectivas categorías y los ejemplos nuevos al sistema.

6. Aprendizaje multi-tarea

   Métodos de aprendizaje que usan conocimiento previamente aprendido por el sistema de cara a enfrentarse a problemas parecidos a los ya vistos.

2.2.1. Algoritmos de categorización y clasificación binaria

Los siguientes algoritmos son los que comunmente se utilizan en categorización y los que hemos utilizado en la parte experimental de este trabajo.

2.2.1.1. Knn

En el método knn es un método de clasificación supervisada que sirve para estimar la función de densidad F(x/C_j) de las predictoras x por cada

clase C_j.

Este es un método de clasificación no paramétrico, que estima el valor de la función de densidad de probabilidad o directamente la probabilidad a posteriori de que un elemento x pertenezca a la clase C_j a partir de la información proporcionada por el conjunto de prototipos. En el proceso de aprendizaje no se hace ninguna suposición acerca de la distribución de las variables predictoras.

En el reconocimiento de patrones, el algoritmo knn es usado como método de clasificación de objetos basado en un entrenamiento mediante ejemplos cercanos en el espacio de los elementos. knn es un tipo de "Lazy Learning", donde la función se aproxima solo localmente y todo el cómputo es diferido a la clasificación.

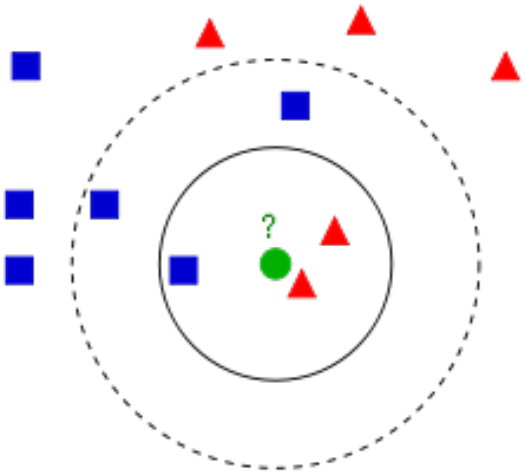


Figura 2.3: Ejemplo Knn

En el ejemplo de la figura anterior se desea clasificar el círculo verde. Para k = 3 este es clasificado con la clase triángulo, ya que hay solo un cuadrado y 2 triángulos, dentro del circulo que los contiene. Si k = 5 este es clasificado con la clase cuadrado, ya que hay 2 triángulos y 3 cuadrados, dentro del circulo externo.

2.2.1.2. Árboles de decisión

Un árbol de decisión es un modelo de predicción utilizado en el ámbito de la inteligencia artificial. Dada una base de datos se construyen diagramas de construcciones lógicas, muy similares a los sistemas de predicción basados en reglas, que sirven para representar y categorizar una serie de condiciones que ocurren de forma sucesiva, para la resolución de un problema.

Un árbol de decisión tiene unas entradas las cuales pueden ser un objeto o una situación descrita por medio de un conjunto de atributos y a partir de esto devuelve una respuesta la cual en últimas es una decisión que es tomada a partir de las entradas. Los valores que pueden tomar las entradas y las salidas pueden ser valores discretos o continuos. Se utilizan más los valores discretos por simplicidad, cuando se utilizan valores discretos en las funciones de una aplicación se denomina clasificación y cuando se utilizan los continuos se denomina regresión.

Un árbol de decisión lleva a cabo un test a medida que este se recorre hacia las hojas para alcanzar así una decisión. El árbol de decisión suele contener nodos internos, nodos de probabilidad, nodos hojas y arcos. Un nodo interno contiene un test sobre algún valor de una de las propiedades. Un nodo de probabilidad indica que debe ocurrir un evento aleatorio de acuerdo a la naturaleza del problema, este tipo de nodos es redondo, los demás son cuadrados. Un nodo hoja representa el valor que devolverá el árbol de decisión y finalmente las ramas brindan los posibles caminos que se tienen de acuerdo a la decisión tomada.

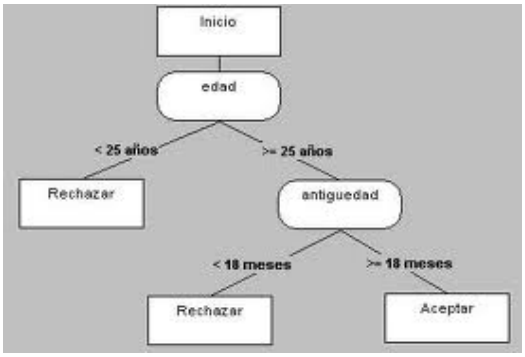


Figura 2.4: Ejemplo Árbol de Decisión

Las principales ventajas de los árboles de decisión son las siguientes:

1. Fáciles de entender
2. Los resultados se adaptan bien a reglas de negocios
3. No se requieren suposiciones acerca de los datos
4. Las variables de entrada pueden ser continuas o categóricas

Los principales inconvenientes de los árboles de decisión son los siguientes:

1. Algunos algoritmos sólo pueden tratar variables binarias
2. Algunos algoritmos no funcionan bien cuando el número de casos de entrenamiento es pequeño
3. Son costosos en términos computacionales

2.2.1.3. Naives Bayes

El algoritmo Naive Bayes (NB), ampliamente usado en procesos de clasificación, se lo considera como una forma especial, o como el modelo más simple de clasificación basado en una red Bayesiana [20] y dentro del campo de las máquinas de aprendizaje y minería de datos, es reconocido como uno de los algoritmos más eficientes y efectivos de aprendizaje inductivo. El presente algoritmo centra su fundamento en la hipótesis de que todos los atributos son independientes entre sí, conocido el valor de la variable clase. El algoritmo representa una distribución de una mezcla de componentes, donde cada componente dentro de todas las variables se asumen independientes. Esta hipótesis de independencia da lugar a un modelo de un único nodo raíz, correspondiente a la clase, y en el que todos los atributos son nodos hoja que tienen como único origen a la variable clase [20]. En varias situaciones se ha demostrado que el algoritmo en cuestión, trabaja mejor en dos casos: cuando los atributos son completamente independientes, como es lógico esperar dada su premisa, y cuando los atributos son funcionalmente dependientes, lo que ya es menos evidente. Presenta sus peores resultados en situaciones intermedias entre estos dos extremos.

2.2.1.4. Máquinas de Vectores Soporte

Las máquinas de soporte vectorial o máquinas de vectores de soporte son un conjunto de algoritmos de aprendizaje supervisado.

Estos métodos están propiamente relacionados con problemas de clasificación y regresión. Dado un conjunto de ejemplos de entrenamiento, podemos etiquetar las clases y entrenar una SVM para construir un modelo que prediga la clase de una nueva muestra. Intuitivamente, una SVM es un modelo que representa a los puntos de muestra en el espacio, separando las clases por un espacio lo más amplio posible. Cuando las nuevas muestras se ponen en correspondencia con dicho modelo, en función de su proximidad pueden ser clasificadas a una u otra clase.

Más formalmente, una SVM construye un hiperplano o conjunto de hiperplanos en un espacio de dimensionalidad muy alta que puede ser utilizado en problemas de clasificación o regresión. Una buena separación entre las clases permitirá un clasificación correcta.

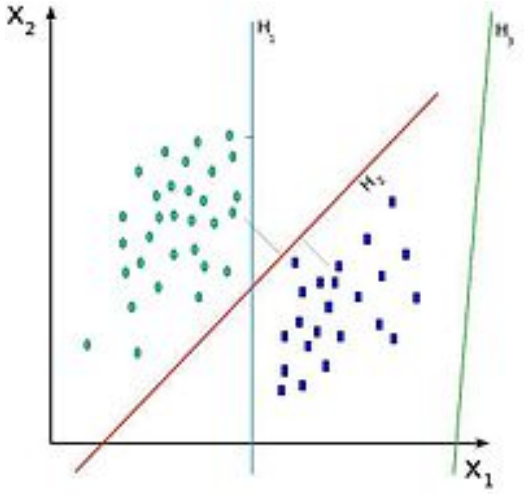


Figura 2.5: Ejemplo SVM en 2 dimensiones

2.2.1.5. Regresión Logística

Los modelos de regresión logística son modelos estadísticos en los que se desea conocer la relación entre: una variable dependiente cualitativa, dicotómica (regresión logística binaria o binomial) o con más de dos valores (regresión logística multinomial). Una o más variables explicativas independientes, o covariables, ya sean cualitativas o cuantitativas, siendo la ecuación inicial del modelo de tipo exponencial, si bien su transformación logarítmica permite su uso como una función lineal. Como se ve, las covariables pueden ser cuantitativas o cualitativas. Las covariables cualitativas deben ser dicotómicas, tomando valores 0 para su ausencia y 1 para su presencia (esta codificación es importante, ya que cualquier otra codificación provocaría modificaciones en la interpretación del modelo). Pero si la covariable cualitativa tuviera más de dos categorías, para su inclusión en el modelo debería realizarse una transformación de la misma en varias covariables cualitativas dicotómicas ficticias o de diseño (las llamadas variables dummy), de forma que una de las categorías se tomaría como categoría de referencia. Con ello cada categoría entraría en el modelo de forma individual. En general, si la covariable cualitativa posee n categorías, habrá que realizar $n - 1$ covariables ficticias.

Por sus características, los modelos de regresión logística permiten dos finalidades:

1. Cuantificar la importancia de la relación existente entre cada una de las covariables y la variable dependiente, lo que lleva implícito también clarificar la existencia de interacción y confusión entre covariables respecto a la variable dependiente (es decir, conocer la odds ratio para cada covariable).

2. Clasificar individuos dentro de las categorías (presente/ausente) de la variable dependiente, según la probabilidad que tenga de pertenecer a una de ellas dada la presencia de determinadas covariables.

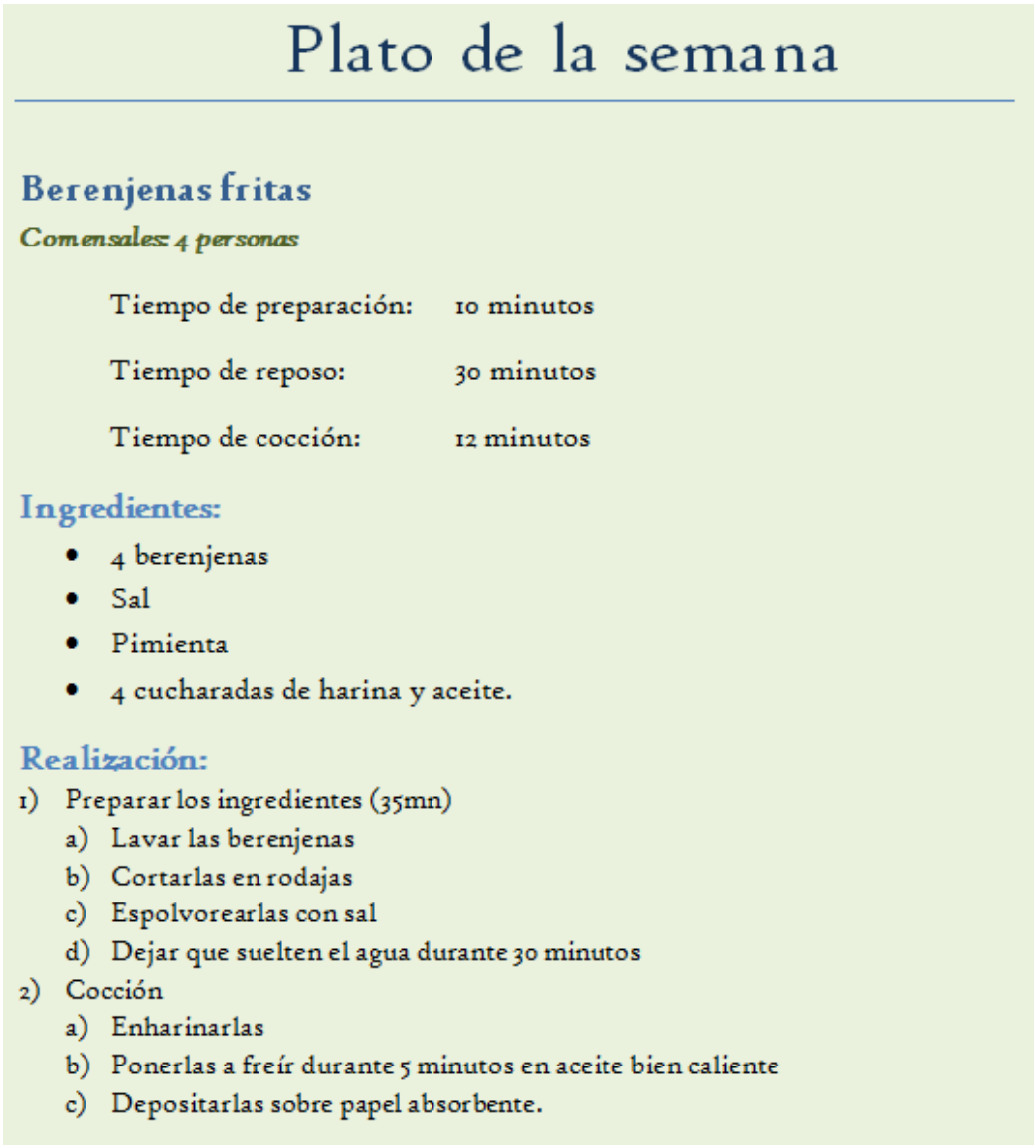
El objetivo principal que resuelve esta técnica es el de modelar cómo influye en la probabilidad de aparición de un suceso, habitualmente dicotómico, la presencia o no de diversos factores y el valor o nivel de los mismos. También puede ser usada para estimar la probabilidad de aparición de cada una de las posibilidades de un suceso con más de dos categorías.

2.3. Clasificación Multi-etiqueta

La clasificación es una de las principales áreas de interés en el campo del aprendizaje automático. Esencialmente, consiste en hallar una función, denominada clasificador, que reciba como entrada los atributos de un determinado patrón y devuelva como salida una clase, de entre una serie de clases predefinidas en el problema, que indique la clase con la que se considera que está asociado dicho patrón.

Habitualmente, este problema se ha definido asumiendo que cada patrón únicamente puede estar asociado a una sola clase. En esta formulación, se habla de clasificación binaria cuando el número de etiquetas es igual a dos y de clasificación multiclase cuando el número de etiquetas es mayor que dos.

Existen numerosos problemas, en los que el requisito de que cada patrón pertenezca a una sola clase es inadmisible. Por ejemplo, si pretendemos modelar un problema de clasificación de textos, un texto como el de la figura 2.1, que muestra la receta de unas berenjenas, que además, son el plato de la semana, podría ser asociado con la clase berenjenas, con la clase receta o con la clase plato de la semana, pero nunca con las tres a la vez.



Plato de la semana

Berenjenas fritas

Comensales: 4 personas

Tiempo de preparación: 10 minutos

Tiempo de reposo: 30 minutos

Tiempo de cocción: 12 minutos

Ingredientes:

- 4 berenjenas
- Sal
- Pimienta
- 4 cucharadas de harina y aceite.

Realización:

1) Preparar los ingredientes (35mn)
   a) Lavar las berenjenas
   b) Cortarlas en rodajas
   c) Espolvorearlas con sal
   d) Dejar que suelten el agua durante 30 minutos
2) Cocción
   a) Enharinarlas
   b) Ponerlas a freír durante 5 minutos en aceite bien caliente
   c) Depositarlas sobre papel absorbente.

Figura 2.6: Ejemplo de texto multi-etiqueta

Para poder tratar este tipo de problemas adecuadamente, surge la clasificación multi-etiqueta, en la que pueden existir patrones a los que se les asocie más de una etiqueta a la vez.

En la tabla 2.1 se puede ver un conjunto de una sola etiqueta y en la tabla 2.2 un conjunto multi-etiqueta.

Ejemplo	Etiqueta
1	Receta
2	Berenjena
3	Plato de la semana

Tabla 2.1: Ejemplo de una sola etiqueta

Ejemplo	Etiquetas		
	Receta	Berenjena	Plato de la Semana
1	X	X	X
2	X		
3			X
4		X	X

Tabla 2.2: Ejemplo de varias etiquetas

2.4. Métodos de clasificación multi-etiqueta

Principalmente existen 2 enfoques a la hora de realizar clasificación multi-etiqueta: los métodos de transformación de problemas y los métodos de adaptación de algoritmos. A continuación, se va a entrar en detalle en la explicación de cada uno de estos métodos y sus diferentes aproximaciones.

Para clarificar las explicaciones, utilizaremos de ahora en adelante el siguiente dataset multi-etiqueta, a modo de ejemplo:

Ejemplo	Etiquetas			
	Deporte de Equipo	Deporte Individual	Deporte de Pelota	Deporte de Agua
Fútbol	X		X	
Waterpolo	X		X	X
Golf		X	X	
Ciclismo		X		

Tabla 2.3: Ejemplo deportes multi-etiqueta

2.4.1. Métodos de transformación de problemas

Los métodos de transformación transformación de problemas [19], convierten un conjunto de datos multi-etiqueta en uno o varios conjuntos de

datos de una sola etiqueta.

Ventajas:

1. Permite utilizar técnicas de categorización sobre datos multi-etiqueta.

Inconvenientes:

1. Se pierden las correlaciones entre etiquetas.
2. Para determinados conjuntos de datos la transformación puede llegar a ser muy costosa.

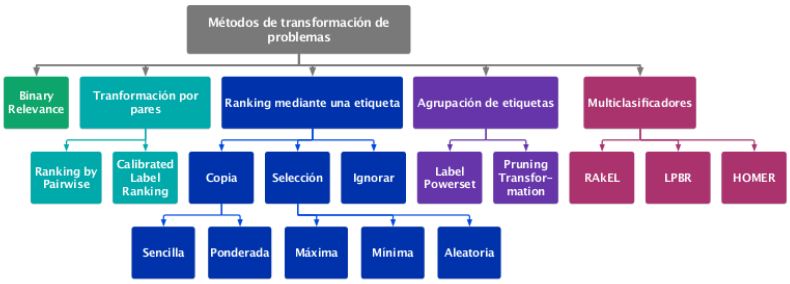


Figura 2.7: Métodos de transformación de problemas [19]

A continuación, señalaremos los principales métodos existentes dentro de este enfoque.

2.4.1.1. Binary Relevance

Este método genera una serie de conjuntos de datos binarios por cada etiqueta presente en el problema. Cada uno de los conjuntos generados, contiene todos los patrones del conjunto original y en cada uno de ellos, están marcados como positivos los patrones asociados a la etiqueta y como negativos el resto. [18]

Una vez realizada la transformación ha de utilizarse un algoritmo de clasificación binaria clásica que permita aprender un clasificador nuevo por cada conjunto generado. El clasificador multi-etiqueta estará formado por la unión de las respuestas de cada uno de los clasificadores aprendidos para determinar el conjunto de etiquetas del patrón.

Ejemplo	Etiquetas			
	Deporte de Equipo	Deporte Individual	Deporte de Pelota	Deporte de Agua
Fútbol	T	F	T	F
Waterpolo	T	F	T	T
Golf	F	T	T	F
Ciclismo	F	T	F	F

Tabla 2.4: Ejemplo de transformación BR aplicado al dataset de la tabla 2.3

2.4.1.2. Ranking mediante una sola etiqueta

Este método aplican un primer lugar una transformación que genera conjunto de datos de una sola etiqueta. Posteriormente se ha de utilizar una técnica de clasificación clásica.

Para transformar el conjunto de datos, existen varios mecanismos como son el método de copia, el método de selección y el método ignorar. [18]

Los métodos de copia transforman cada patrón multi-etiqueta en varios patrones de una sola etiqueta. Para ello copian varias veces el patron con cada una de las etiquetas que contiene.

Ventajas: No se pierde información sobre las clases.

Inconvenientes: Aumenta considerablemente el número de patrones, siendo aquellos que más etiquetas posean los que tendrán más relevancia en el clasificador final. Se puede solventar asignando un peso al patrón, que será menor cuanto mayor número de etiquetas tuviera el patrón multi-etiqueta original.

Deporte	Etiqueta	Peso
Fútbol	Deporte de Equipo	1/2
Fútbol	Deporte de Pelota	1/2
Waterpolo	Deporte de Equipo	1/3
Waterpolo	Deporte de Pelota	1/3
Waterpolo	Deporte de Agua	1/3
Golf	Deporte de Equipo	1/2
Golf	Deporte de Pelota	1/2
Ciclismo	Deporte Individual	1

Tabla 2.5: Ejemplo de transformación de copia sencilla y ponderada aplicada al dataset de la tabla 2.3

El método de selección transforma cada patrón multi-etiqueta en un patrón de una sola etiqueta. El método debe ser capaz de seleccionar una sola etiqueta de entre todas las que el patrón presente. Existen 3 técnicas principales para hacerlo:

1. **Selección máxima:** se selecciona la etiqueta con más ocurrencias en el conjunto de datos
2. **Selección mínima:** se selecciona la etiqueta con menos ocurrencias en el conjunto de datos
3. **Selección aleatoria:** se selecciona aleatoriamente la etiqueta

En el dataset de la tabla 2.3, al tener todos las etiquetas de los ejemplos 1 ocurrencia, cualquier etiqueta de cualquier ejemplo podría ser seleccionada en cualquiera de las técnicas.

Ventajas:

1. Transforman un dataset multi-etiqueta en uno de una sola etiqueta sin aumentar tamaño.

Inconvenientes:

1. Eliminan información sobre etiquetas.
2. Eliminan información sobre las relaciones entre etiquetas.

3. Es una transformación irreversible.

El método ignorar elimina del conjunto de datos los patrones multi-etiqueta.

Ventajas:

1. Muy simple de implementar

Inconvenientes:

1. Eliminan información sobre etiquetas.
2. Eliminan información sobre las relaciones entre etiquetas.
3. Desaparece toda la información multi-etiqueta del conjunto.
4. Es una transformación irreversible.

Deporte	Etiqueta
Ciclismo	Deporte Individual

Tabla 2.6: Ejemplo de transformación ignorar aplicada al dataset de la tabla 2.3

2.4.1.3. Métodos de transformación por pares

Los métodos de transformación por pares [17], convierten un conjunto de datos multi-etiqueta en varios conjuntos de datos binarios, uno por cada par de etiquetas. Posteriormente por cada uno de ellos se aprenderá un clasificador binario, utilizando cualquier técnica de clasificación binaria clásica.

Deporte	Conjunto
	Deporte de Equipo - Deporte individual
Futbol	F
Waterpolo	F
Golf	F
Ciclismo	F
	Deporte de Equipo - Deporte de Pelota
Futbol	T
Waterpolo	T
Golf	F
Ciclismo	F
	Deporte de Equipo - Deporte de Agua
Futbol	F
Waterpolo	T
Golf	F
Ciclismo	F
	Deporte Individual - Deporte de Pelota
Futbol	F
Waterpolo	F
Golf	T
Ciclismo	F
	Deporte Individual - Deporte de Agua
Futbol	F
Waterpolo	F
Golf	F
Ciclismo	F
	Deporte Pelota - Deporte de Agua
Futbol	F
Waterpolo	T
Golf	F
Ciclismo	F

Tabla 2.7: Ejemplo de transformación por pares aplicada al dataset de la tabla 2.3

2.4.1.4. Métodos de agrupación de etiquetas

Los métodos de agrupación de etiquetas [18], transforman los conjuntos multi-etiqueta agrupando las etiquetas entre sí, convirtiendo posteriormente cada grupo de etiquetas en una sola etiqueta, quedando el problema transformado a uno clásico de categorización.

Entre los métodos de agrupación de etiquetas, destaca el método Label Powerset (LP), que une todas las etiquetas de cada ejemplo en una.

Deporte	Etiqueta
Futbol	Deporte de Equipo - Deporte de Pelota
Waterpolo	Deporte de Equipo - Deporte de Agua
Golf	Deporte Individual - Deporte de Pelota
Ciclismo	Deporte Individual

Tabla 2.8: Ejemplo de transformación LP aplicada al dataset de la tabla 2.3

2.4.1.5. Métodos multiclasificadores

Los métodos multiclasificadores (ensemble methods) combinan las respuestas de varios clasificadores, desarrollados mediante la misma o distinta técnica, para obtener una respuesta en general más adecuada que la de cada uno de los clasificadores por separado. [27]

RAKEL

El algoritmo RAKEL [27], aprende una serie de clasificadores de tipo LP, cada uno de ellos especializado en clasificar subconjuntos de etiquetas presentes en el problema. Para ello genera aleatoriamente una serie de subconjuntos de k etiquetas, entrenando un clasificador multietiqueta para cada uno de los subconjuntos. El clasificador final esta formado por la agregación de todos los clasificadores entrenados, y las etiquetas se asignan a los nuevos patrones por votación.

Algoritmo Pruning Transformation

El algoritmo Pruning Transformation [27], está concebido para problemas con grandes combinaciones de etiquetas. Elimina los patrones asociados con combinaciones de etiquetas menos relevantes para el problema, a partir de un umbral definido por el usuario. Estos ejemplos eliminados no son descartados, sino que se pueden reintroducir en el conjunto de entrenamiento si alguno de sus subconjuntos es superior al umbral.

2.4.2. Métodos de adaptación de algoritmos

Los métodos de adaptación de algoritmos emplean una técnica de clasificación clásica adaptada para trabajar directamente con datos multi-etiqueta.

La mayoría de las técnicas empleadas en categorización (explicadas anteriormente) han sido adaptadas para trabajar con datos multi-etiqueta.

2.5. Métricas de evaluación

Principalmente, las métricas existentes para la evaluación de modelos se pueden clasificar en dos tipos: basadas en ejemplos y basadas en etiquetas.

Las métricas basadas en etiquetas son métricas de clasificación binaria clásicas y se calculan para cada etiqueta, siendo posteriormente promediadas para obtener un valor único.

Las métricas basadas en ejemplos están específicamente concebidas para problemas multi-etiqueta, y por esta razón se calculan para cada ejemplo teniendo en cuenta todo el conjunto de etiquetas (predicho y real).

2.5.1. Métricas basadas en etiquetas

Cualquier métrica de evaluación binaria, B, puede ser utilizada como base de una métrica basada en etiquetas. Estas medidas como base la matriz de confusión (explicada a continuación), en la que se representa, para una determinada clase, la salida de clasificador frente al valor esperado.

Así t_p indica el número de ejemplos positivos clasificados correctamente, f_p los ejemplos incorrectamente clasificados como positivos o falsos positivos, t_n los ejemplos negativos correctamente asignados y f_n los ejemplos no asignados o falsos negativos.

Debido a que en clasificación multi-etiqueta hay varias etiquetas presentes en el problema, para promediar se pueden utilizar dos aproximaciones, los denominados enfoques macro y micro [18]. Bajo el enfoque macro primero se calcula la métrica para cata etiqueta y se hace la media por etiquetas para obtener el valor final, mientras que en el enfoque micro se agregan los valores de la matriz de confusión para las etiquetas y posteriormente se calcula la métrica con estos valores.

2.5.1.1. Matriz de confusión

Es una herramienta de visualización que se emplea en aprendizaje supervisado. Cada columna de la matriz representa el número de predicciones de cada clase, mientras que cada fila representa a las instancias en la clase real. Uno de los beneficios de las matrices de confusión es que facilitan ver si el sistema está confundiendo dos clases.

En un hipotético caso de clasificador perfecto, todos los elementos de la matriz que no formen parte de la diagonal deberían ser iguales a 0.

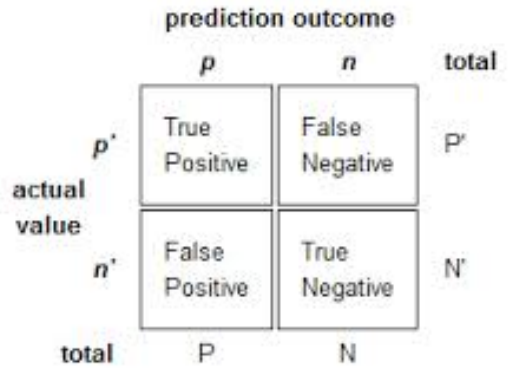


Figura 2.8: Matriz de confusión

2.5.1.2. Precision

La precisión de un clasificador se define como el cociente entre las etiquetas correctamente asignadas y todas las etiquetas asignadas por el clasificador, es decir:

$$precision = \frac{t_p}{t_p + f_p} \tag{2.1}$$

2.5.1.3. Recall

El recall de un clasificador es el cociente entre las etiquetas correctamente clasificadas y las realmente positivas, es decir:

$$recall = \frac{t_p}{t_p + f_n} \tag{2.2}$$

2.5.1.4. F-score

El F-score se define como la media armónica entre la precision y el recall, es decir:

$$F - score = \frac{2 \cdot precision \cdot recall}{precision + recall} \tag{2.3}$$

2.5.2. Métricas basadas en ejemplos

Las métricas basadas en ejemplos toman como entrada la respuesta completa del clasificador para cada instancia. Posteriormente, se promedia el valor de la métrica para obtener el valor medio de ésta sobre todo el conjunto de entrenamiento. Dentro de las métricas basadas en ejemplos encontramos las denominadas métricas binarias, que son utilizadas para medir cualquier algoritmo de clasificación multi-etiqueta y las métricas de ranking que se utilizan en el contexto de algoritmos de ranking. Explicaremos a continuación solamente las métricas binarias, pues no es objetivo de este TFM entrar en detalle sobre algoritmos de ranking y sus métricas.

2.5.2.1. Hamming loss

La distancia de Hamming considera tantos los errores de clasificación (el hecho de que una etiqueta incorrecta sea predicha) como los errores por omisión (cuando una etiqueta que debería estar presente en el clasificador no lo está). La Hamming loss es una medida de la diferencia entre el conjunto de

etiquetas predichas y la real. Está acotada entre 0 y 1, siendo 0 su mejor resultado y 1 su peor. P es el conjunto de etiquetas predichas por el clasificador e Y el conjunto de etiquetas reales del clasificador.

$$Hamming\ loss = \frac{1}{n}\sum_{i=1}^{n}\frac{P_i \triangle Y_i}{n} \tag{2.4}$$

2.5.2.2. Precision

Se define la precision como el promedio entre las etiquetas acertadas y las predichas, es decir:

$$precision = \frac{1}{n}\sum_{i=1}^{n}\frac{Y_i \cap Y_i}{P_i} \tag{2.5}$$

2.5.2.3. Recall

Se define el recall como el promedio entre las etiquetas acertadas y las reales, es decir:

$$recall = \frac{1}{n}\sum_{i=1}^{n}\frac{Y_i \cap Y_i}{Y_i} \tag{2.6}$$

2.5.2.4. Accuracy

La accuracy se define como el promedio de la fraccion de aciertos del clasificador frente a la unión de etiquetas reales y predichas, es decir:

$$accuracy = \frac{1}{n}\sum_{i=1}^{n}\frac{Y_i \cap Y_i}{Y_i \cup P_i} \tag{2.7}$$

2.6. Métricas para la descripción de los datasets

A la hora de plantear un desarrollo experimental sobre un modelo que genere clasificadores multi-etiqueta, así como de discutir los resultados obtenidos por éste, es importante determinar cómo de multi-etiqueta es el conjunto de datos con el que se entrena y con el que se valida. Para ello se han propuesto varias métricas que se explican a continuación.

2.6.1. Cardinalidad

La cardinalidad de un conjunto de datos se define como el número de etiquetas por patrón, es decir:

$$cardinalidad = \frac{1}{n}\sum_{i=1}^{n}|Y_i| \tag{2.8}$$

2.6.2. Distinct o combinaciones de etiquetas

Se define como el número de distintas combinaciones de etiquetas presente en el conjunto de datos. [18].

2.6.3. Densidad

La densidad de un conjunto de datos se define con la cardinalidad dividida entre el número total de etiquetas, es decir:

$$densidad = \frac{cardinalidad}{L} \tag{2.9}$$

2.6.4. Número de etiquetas

Se define como el número de etiquetas que forman parte del conjunto de datos

2.6.5. Numero de ejemplos

Se define como el número de ejemplos que forman parte del conjunto de datos.

2.7. Aplicaciones de la clasificación multi-etiqueta

La clasificación multi-etiqueta se ha aplicado para resolver múltiples problemas reales, en múltiples dominios, como clasificación de documentos, bioinformática, clasificación de imágenes o medicina entre otros muchos. En esta sección se indican las principales características de cada una de estas aplicaciones.

2.7.1. Clasificación de textos

Tema de este trabajo y que, prácticamente está ya explicado. Si queremos clasificar un texto por categorías semánticas, es bastante habitual que a un documento le correspondan dos o más categorías. Esta tarea no era posible abordarla mediante categorización [12] . Los primeros trabajos se centraron en manejar la problemática que se produce cuando se aborda la clasificación de documentos mediante técnicas de clasificación clásicas. Lo normal es que los textos lleven asociada más de una etiqueta, por lo que el problema no puede ser abordado mediante técnicas de categorización. Problemas como la detección de spam son abordados mediante clasificadores multi-etiqueta.

2.7.2. Clasificación de imágenes

La clasificación de imágenes es un problema en el que se trata de asociar automáticamente a cada imagen las clases a las que pertenece. Este problema ha sido abordado utilizando técnicas multi-etiqueta desde numerosas ópticas y paradigmas. Otra aplicación directamente relacionada con la clasificación de imágenes es el etiquetado automático de imágenes.

2.7.3. Bioinformática

El campo de la bioinformática es uno de los que más se ha beneficiado de la utilización de técnicas multi-etiqueta, ya que estas se han aplicado satisfactoriamente a problemas como la clasificación de proteínas o de genes, la determinación de la estructura de las proteínas o la multi-localización de proteínas.

2.7.4. Medicina

En medicina se ha aplicado la clasificación multi-etiqueta, por ejemplo, para modelar el problema del diagnóstico clínico. Hay distintas enfermedades que comparten síntomas entre sí, por lo que estamos ante un problema de clasificación multi-etiqueta. Otro problema que se ha modelado mediante técnicas de aprendizaje multi-etiqueta es el de las interacciones entre medicamentos, siendo las diversas etiquetas las sustancias que presentan interacciones. También se ha aplicado a problemas como la acción de medicamentos anti-cancerígenos sobre distintos tipos de células tumorales. [23]

2.7.5. Clasificación musical o de sonidos

El problema de la categorización de sonidos consiste en clasificar sonidos, sobre todo musicales, según un criterio determinado. De la misma manera que ocurría con las imágenes, con mucha frecuencia es necesario modelar este problema con aprendizaje multi-etiqueta, porque las categorías en las que se agrupan las piezas musicales suelen solaparse entre sí.

2.8. Clasificación Jerárquica

El objetivo de este apartado es presentar trabajos relacionados con la clasificación jerárquica.

En [16] se presenta un estudio sobre clasificación jerárquica. Los objetivos que se persiguen con dicho estudio son los siguientes:

1. Clarificar la definición de clasificación jerárquica.
2. Proponer un framework unificado de clasificacion para los nuevos y existentes métodos de clasificación jerárquica y también para los problemas.
3. Proponer una taxonomía de los diferentes métodos de clasificación jerárquica en función de sus similitudes y diferencias.
4. Sugerir protocolos experimentales con el fin de obtener mejores resultados. Por ejemplo, muchos autores señalan que algunos métodos de clasificación jerárquicos son mejores que otros, pero a menudo se utilizan medidas de evaluación estándar en lugar de utilizar medidas de evaluación jerárquicas.

Según [11] y [30] los métodos de clasificación jerárquicos difieren en diversos criterios. El primer criterio es el tipo de estructura jerárquica que utilizan. Normalmente suele ser un árbol o un DAG. El segundo criterio se refiere a como se realiza la clasificación en la jerarquía. Teniendo en cuenta ésto, el método de clasificación se puede implementar para que siempre prediga un nodo hoja o por el contrario, se permite predecir cualquier nodo de la jerarquía. El tercer criterio se refiere a como es explorada la estructura jerárquica.

Koller and Sahami [22], consideran el problema de clasificación jerárquica de texto en la queRed Bayesiana jerárquica 3 cada do cumento de texto llega a ser exactamente una clase en el último nivel de la jerarquía.

En el trabajo de Cesa-Bianchi et al.[26], cada instancia de datos es etiquetada con un conjunto de etiquetas de clase, las cuales pueden llegar a tener más de una ruta en la jerarquía.

Barutcouglu et al. [14], presentaron una aproximación a dos pasos donde una SVM es aprendida para cada clase por separado. Y luego combinándolas usando un modelo de red bayesiana entonces las predicciones son consistentes con la jerarquía.

Vens et al. [32], utilizan árboles de decisión para la clasificación jerarquíca múlti-etiqueta.

Gjorgjioski et al. [31], utilizan árboles predictivos de clustering para realizar clasificación multi-etiqueta. Además, llevan a cabo una comparación experimental de cuatro medidas basadas en distancias.

Capítulo 3

Tecnologías

Para la realización de los experimentos necesitabamos una librería de aprendizaje automático con soporte para multi-etiqueta, acceso al api de la Wikipedia mediante algún lenguaje de programación para poder descargar los articulos objetivo junto con sus categorías, alguna librería de procesamiento de lenguaje natural para poder preprocesar los textos y alguna libería que nos permitiera trabajar con grafos. A continuación se describen las tecnologías elegidas para cada una de las misiones antes indicadas.

3.1. API Rest de MediaWiki

Prácticamente la totalidad de las personas han oído hablar de Wikipedia. Es el máximo repositorio de conocimiento alimentado por público, cubriendo casi cada tema en el que se pueda pensar y disponible en cualquier navegador Web. Información de prácticamente cualquier tipo puede ser encontrada en Wikipedia, y a menudo, exhaustivamente detallada. Y como se puede editar públicamente, siempre contiene información relevante y actualizada. [2]

Wikipedia está desarrollada con MediaWiki, un motor de wikis que posee una API Rest que permite a los desarrolladores acceder, buscar e integrar contenido Wikipedia en aplicaciones Web personalizadas. Esta API, que funciona sobre HTTP y que devuelve datos en una gran variedad de formatos diferentes incluyendo JSON o XML.

En nuestro caso, con todo ese contenido disponible, hemos creado los datasets para nuestros experimentos.

3.1.1. ¿Cómo funciona la API?

La API funciona aceptando solicitudes HTTP que contengan uno o más argumentos de entrada y que devuelvan respuestas que el cliente pueda analizar y utilizar.

Cada solicitud HTTP a la API debe contener un parámetro obligatorio, el cual especifica la acción requerida. Este parámetro puede ser una consulta, una operación de edición o de eliminación, una solicitud de autenticación o cualquier otra de las acciones soportadas. Además de este parámetro obligatorio y dependiendo de la operación seleccionada, se deben pasar argumentos adicionales—por ejemplo, una palabra clave de búsqueda para consultas, un título de página para operaciones de edición o eliminación, o un nombre de usuario y contraseña para autenticación.

3.1.2. ¿Cómo tengo acceso a los elementos que forman parte de una categoría?

El parámetro principal para obtener los elementos de una categoría es list=categorymembers. Además existen alrededor de 20 parámetros configurables en función de la información que quiera obtener el usuario. En nuestro caso de estudio, hemos utilizado los siguientes:

cmtitle: La categoría cuyo contenido se quiere listar

cmtype: Tipo de contenido a incluir; páginas, subcategorías o archivos.

format: Formato en el que será devuelta la información.

clshow: Condiciones que deben cumplir los elementos (por ejemplo en el caso de las categorías, puede que nos interese que no estén vacías).

cmlimit: Número de elementos a mostrar.

Imaginemos que queremos listar 20 subcategorías no vacías de la categoría Ball_games. La url de la consulta tendría la siguiente forma:

```
http://en.wikipedia.org/w/api.php?action=query&list=categorymembers&cmlimit=20
&cmtitle=Category:Ball_games&&cmtype=subcat&clshow=!hidden&format=json
```

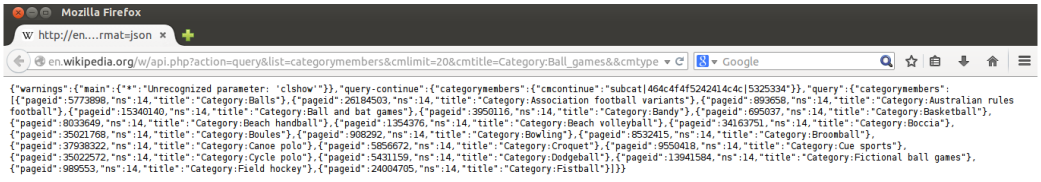


Figura 3.1: Acceso MediaWiki

Si se desea obtener más información sobre el resto de parámetros de categorymembers, se puede consultar la siguiente url:

http://www.mediawiki.org/wiki/API:Categorymembers

No es objetivo de este TFM convertirse en un tutorial avanzado de MediaWiki. Es por ello que incluimos sólo el ejemplo anterior, que contiene los parámetros más utilizados en la parte experimental de este trabajo.

La API completa de Wikipedia pueden consultarse en:

http://en.wikipedia.org/w/api.php

3.2. Sklearn

EL proyecto SKlearn es una librería para aprendizaje automático de código abierto escrita en Python. Cuenta con varios algoritmos de clasificación, regresión y clustering incluyendo máquinas de vectores soporte, regresión logística, Naives Bayes, etc. y está diseñado para interoperar con las bibliotecas numéricas y científicas de Python NumPy y Scipy. [3] Es un proyecto muy popular en github, presentando en agosto de 2014 más de 2100 forks.

3.3. NLTK

NLTK, es un conjunto de bibliotecas y programas para el procesamiento del lenguaje natural (PLN) simbólico y estadísticos para el lenguaje de programación Python. NLTK incluye demostraciones gráficas y datos de muestra. [4]

NLTK está destinado a apoyar la investigación y la enseñanza en PLN o áreas muy relacionadas, que incluyen la lingüística empírica, las ciencias cognitivas, la inteligencia artificial, la recuperación de información, y el aprendizaje de la máquina. NLTK se ha utilizado con éxito como herramienta de enseñanza, como una herramienta de estudio individual, y como plataforma para los sistemas de investigación de prototipos y construcción.

En este TFM hemos usado nltk como librería de apoyo para eliminar las stop words de los conjuntos de datos.

3.4. NetworkX

NetworkX es una libería con licencia BSD de Python para el estudio de grafos y redes. Una de sus características fundamentales, es que posee implementados la mayoría de algoritmos asociados a grafos, como por ejemplo el algoritmo de Dijkstra que utilizamos en este trabajo para calcular la distancia mínima en el grafo entre dos etiquetas, ahorrándonos todo el trabajo de implementación. [5]

Capítulo 4

Clasificacion Jerárquica Multi-etiqueta de artículos de la Wikipedia

La metodología seguida en el presente trabajo ha sido la siguiente; en primer lugar, hemos seleccionado las categorías objetivo de Wikipedia junto con sus correspondientes páginas para crear los datasets.

Sobre esas páginas, se ha realizado en primer lugar una tarea de preprocesado, consistente en eliminar la información correspondiente a etiquetas, imágenes etc. quedándonos sólo con el texto. Una vez teníamos el texto, eliminamos las stops words del mismo, obteniendo como resultado el texto verdaderamente representativo de la página.

Una vez conseguido el conjunto de datos que nos interesa, se procede a realizar una transformación TF-IDF sobre el conjunto.

El resultado del paso anterior, es el conjunto de datos preparado para entrenar el modelo, que será entrenado con diferentes algoritmos.

El modelo obtenido en el paso anterior, es evaluado con diferentes métricas de evaluación, entre ellas, la definida en el presente trabajo.

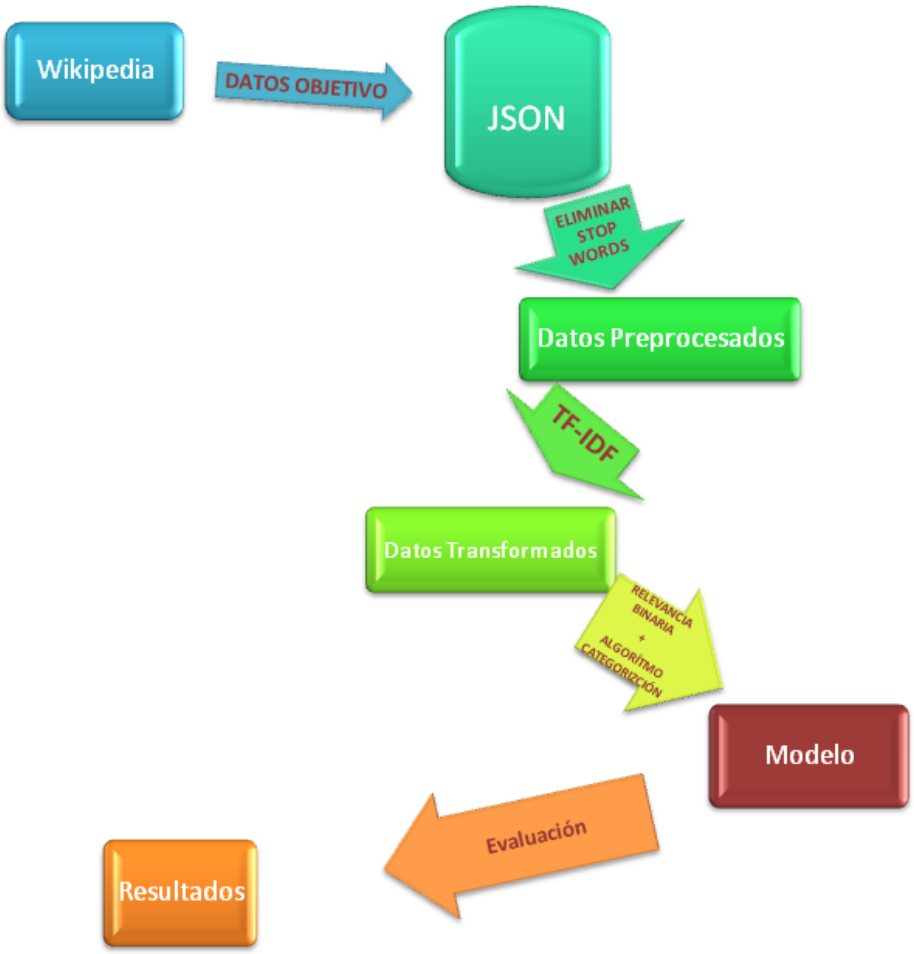


Figura 4.1: Proceso experimentos

4.1. Transformación TF-IDF

Tf-idf [4] es el producto de dos medidas, frecuencia de término y frecuencia inversa de documento. Existen varias maneras de determinar el valor de ambas. En el caso de la frecuencia de término tf(t, d), la opción más sencilla es usar la frecuencia bruta del término t en el documento d, o sea, el número de veces que el término t ocurre en el documento d. Si denotamos la frecuencia bruta de t por f(t,d), entonces el esquema tf simple es tf(t, d) = f(t,d).

Otras posibilidades son:

1. frecuencias booleanas: tf(t,d) = 1 si t ocurre en d, y 0 si no
2. frecuencia escalada logarítmicamente: tf(t,d) = 1 + log f(t,d) y 0 si f(t,d)=0
3. frecuencia normalizada, para evitar una predisposición hacia los documentos largos. Por ejemplo, se divide la frecuencia bruta por la frecuencia máxima de algún término en el documento:

$$tf(t,d) = \frac{f(t,d)}{max\{f(w,d) : w \in d\}}$$

La frecuencia inversa de documento es una medida de si el término es común o no, en la colección de documentos. Se obtiene dividiendo el número total de documentos por el número de documentos que contienen el término, y se toma el logaritmo de ese cociente:

$$idf(t,D) = \frac{log|D|}{1 + |d \in D : t \in d|}$$

es decir, el cociente entre el número de documentos de la colección y 1 + el número de documentos donde aparece el término t (se suma uno para evitar divisiones por 0 cuando el término no se encuentra en la colección)

Luego, tf-idf se calcula como: tfidf(t,d,D) = tf(t,d) x idf(t, D)

4.2. Stop Words

Stop words o palabras vacías es el nombre que reciben las palabras sin significado como artículos, pronombres, preposiciones, etc. que son filtradas antes o después del procesamiento de datos en lenguaje natural [4].

4.3. Algoritmo de Dijkstra

El algoritmo de Dijkstra [15], es un algoritmo para la determinación del camino más corto dado un vértice origen al resto de vértices en un grafo con pesos en cada arista.

La idea subyacente en este algoritmo consiste en ir explorando todos los caminos más cortos que parten del vértice origen y que llevan a todos los demás vértices; cuando se obtiene el camino más corto desde el vértice origen, al resto de vértices que componen el grafo, el algoritmo se detiene. El algoritmo es una especialización de la búsqueda de costo uniforme, y como tal, no funciona en grafos con aristas de coste negativo (al elegir siempre el nodo con distancia menor, pueden quedar excluidos de la búsqueda nodos que en próximas iteraciones bajarían el costo general del camino al pasar por una arista con costo negativo).

Teniendo un grafo dirigido ponderado de N nodos no aislados, sea x el nodo inicial, un vector D de tamaño N guardará al final del algoritmo las distancias desde x al resto de los nodos.

1. Inicializar todas las distancias en D con un valor infinito relativo ya que son desconocidas al principio, exceptuando la de x que se debe colocar en 0 debido a que la distancia de x a x sería 0.
2. Sea a = x (tomamos a como nodo actual).
3. Recorremos todos los nodos adyacentes de a, excepto los nodos marcados, llamaremos a estos nodos no marcados v_i.
4. Para el nodo actual, calculamos la distancia tentativa desde dicho nodo a sus vecinos con la siguiente fórmula: dt(v_i) = D_a + d(a,v_i). Es decir, la distancia tentativa del nodo v_i es la distancia que actualmente tiene el nodo en el vector D más la distancia desde dicho el nodo 'a' (el actual) al nodo v_i. Si la distancia tentativa es menor que la distancia almacenada en el vector, actualizamos el vector con esta distancia tentativa. Es decir: Si dt(v_i) <D_vi → D_vi = dt(v_i)
5. Marcamos como completo el nodo a.
6. Tomamos como próximo nodo actual el de menor valor en D (puede hacerse almacenando los valores en una cola de prioridad) y volvemos al paso 3 mientras existan nodos no marcados.

Una vez terminado al algoritmo, D estará completamente lleno.

4.3.1. Pseudocódigo

```
DIJKSTRA (Grafo G, nodo_fuente s)
     para u ∈ V[G] hacer
         distancia[u] = INFINITO
         padre[u] = NULL
     distancia[s] = 0
     adicionar (cola, (s,distancia[s]))
     mientras que cola no es vacía hacer
         u = extraer_mínimo(cola)
         para todos v ∈ adyacencia[u] hacer
             si distancia[v] > distancia[u] + peso (u, v) hacer
                 distancia[v] = distancia[u] + peso (u, v)
                 padre[v] = u
                 adicionar(cola,(v,distancia[v]))
```

4.4. Validación cruzada

La validación cruzada [20], es una técnica utilizada para evaluar los resultados de un análisis estadístico y garantizar que son independientes de la partición entre datos de entrenamiento y prueba. Consiste en repetir y calcular la media aritmética obtenida de las medidas de evaluación sobre diferentes particiones. Se utiliza en entornos donde el objetivo principal es la predicción y se quiere estimar cómo de preciso es un modelo que se llevará a cabo a la práctica. Es una técnica muy utilizada en proyectos de inteligencia artificial para validar modelos generados.

Existen varios tipos de validación cruzada. Cómo no es objetivo de éste trabajo profundizar en dichos tipos, explicaremos únicamente la validación cruzada de k iteraciones que es la usada en éste trabajo.

4.4.1. Validación cruzada de k iteracciones

En la validación cruzada de K iteraciones, los datos de muestra se dividen en K subconjuntos. Uno de los subconjuntos se utiliza como datos de prueba y el resto (K-1) como datos de entrenamiento. El proceso de validación cruzada es repetido durante k iteraciones, con cada uno de los posiblesés subconjuntos de datos de prueba. Finalmente se realiza la media aritmética de los resultados de cada iteración para obtener un único resultado.

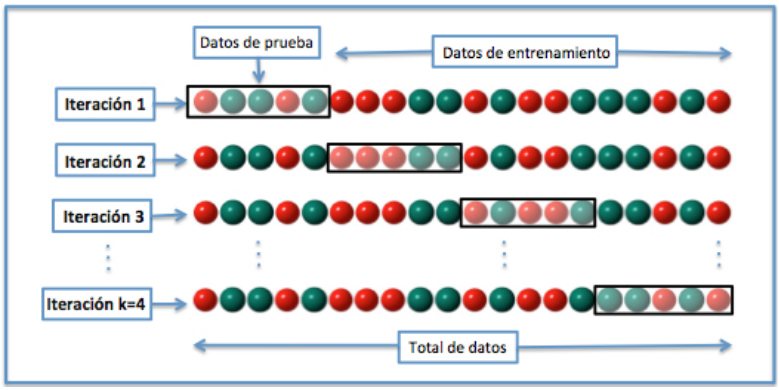


Figura 4.2: Validación Cruzada de 4 iteracciones

Lo más común es utlizar la validación cruzada de 10 iteraciones, y es lo que se ha hecho en este trabajo.

4.5. Construcción de los datasets

Para construir los datasets, en primer lugar, introducimos en un array todas las categorías que queremos que formen parte del dataset. A partir de cada categoría, obtenemos todas las páginas de Wikipedia que formen parte de la misma. Hemos introducido también un entero llamado cardinalidad, para así poder elegir que cardinalidad mínima en el grafo tienen las páginas que forman parte de nuestro dataset. Esto nos permitirá controlar la cardinalidad final del dataset creado. Por ejemplo, si exigimos que todas las páginas tengan una cardinalidad superior a 2, nos aseguramos que el dataset final también tendrá una cardinalidad superior a 2.

4.5.1. Pseudocódigo

```
OBTENER_PAGINAS (Grafo G, cardinalidad_minima_pagina c)
Para cada categoria en el grafo:
    Obtener las paginas de dicha categoria mediante api de mediawiki
    Para cada pagina:
        Obtener titulo, texto y categorias a las que pertenece
        cardinalidad = ComprobarCardinalidad (categorias_pagina, G, c)
        Si cardinalidad > = c:
            Incluir pagina en el dataset
```

4.6. Nueva métrica basada en distancias

Debido a que la clasificación multi-etiqueta en entornos jerárquicos es muy novedosa y apenas hay información en la literatura, siendo éste uno de los motivos por el que decicimos realizar este trabajo, decidimos definir una métrica que nos permitiera conocer la precisión de los modelos generados en función de las distancias existentes en el grafo entre las categorías predichas y las reales de cada patrón.

Se va a explicar la métrica mediante un ejemplo. Imaginemos un conjunto de datos donde las categorías están representadas en el grafo de la Figura 3.1

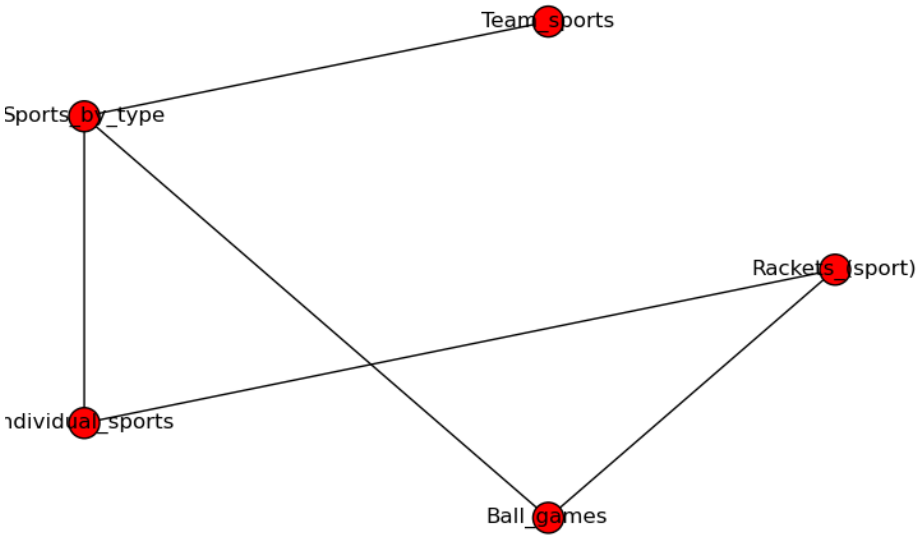


Figura 4.3: Ejemplo grafo de categorías

Pongamos por caso que ya tenemos generado un modelo para clasificar textos y que tenemos tres textos que queremos clasificar. Las siguientes dos tablas muestran las etiquetas reales de los textos y las etiquetas predichas por el clasificador.

Ejemplo	Sports_by_type	Ball_games	Individual_sports	Teams_ports	Rackets_sports
Football		✓		✓	
Chess			✓		
Golf		✓	✓		

Tabla 4.1: Tabla etiquetas reales del conjunto de datos

Ejemplo	Sports_by_type	Ball_games	Individual_sports	Teams_ports	Racket_sports
Football		✓			✓
Chess			✓		
Golf			✓	✓	

Tabla 4.2: Tabla etiquetas predichas por el clasificador

¿Cómo se calcularía la distancia simétrica de este conjunto de datos?

Tendríamos que ir ejemplo por ejemplo, mirando las etiquetas reales del clasificador y las etiquetas predichas y calculando las distancias en el grafo entre las reales y las predichas y viceversa. Hemos asumido que todas las aristas del grafo tienen peso 1.

El ejemplo 1 tiene las etiquetas reales Ball_games y Team_sports y el clasificador ha predicho Ball_games y Racket_sport.

Empezamos calculando la distancia para las etiquetas reales.

Ball_games aparece tanto en las reales como en las predichas, por tanto su distancia parcial es 0.

Team_sports aparece en las reales pero no en las predichas. Por tanto habría que calcular la distancia mínima en el grafo entre Team_sports y las etiquetas predichas por el clasificador. Usamos para ello el Algoritmo de Dijkstra.

Las etiquetas predichas son Team_sports y Racket_sports. La distancia entre Ball_games y Racket_sports en el grafo es 1. La distancia entre Ball_games y Team_sports en el grafo es 2. Por tanto, nos quedamos con la mínima que es 1.

Si asumimos que en el peor clasificador posible, tendríamos que haber recorrido todas las aristas del grafo al menos una vez, la distancia quedaria fijada en el valor 1/4

Este ajuste lo hemos introducido para evitar que la distancia dependa del número de aristas del grafo.

Ahora deberíamos hacer la misma operacion con las etiquetas predichas.

Ball_games aparece tanto en las reales como en las predichas, por tanto su distancia parcial es 0.

Racket_sports aparece en las predichas pero no en las reales. Las etiquetas reales son Team_sports y Ball_games. La distancia entre Ball_games y Racket_sports en el grafo es 1, siendo 3 la distancia entre Team_sports y Racket_sports. Nos quedamos con la mínima que es 1.

Al igual que en el caso anterior, ajustando en función del número de aristas del grafo, la distancia sería 1/4

Sumando ambas distancias parciales, tenemos 2/4. Ahora dividimos por el valor 2 * Etiquetas_predichas * Etiquetas_reales quedándonos la distancia total del ejemplo igual a 1/16

De la misma forma, la distancia en el caso del ajedrez sería 0 (misma etiqueta predicha que real).

Por último, en el caso del golf, Individual_sports aparece tanto en las predichas como en las reales. Ball_games aparece en las reales, pero no en las predichas. La distancia entre Ball_games e Individual_sports es 2 y entre Ball_games y Team_sports es 2 también. Por tanto, la mínima es 2. Si asumimos que en el peor clasificador posible, tendríamos que haber recorrido todas las aristas del grafo al menos una vez, la distancia quedaria fijada en el valor 2/4

Ahora hacemos lo mismo con las predichas. Individual_sports aparece tanto en las predichas como en las reales. Team_sports aparece en las predichas, pero no en las reales. La distancia entre Team_sports y Ball_games es igual a 2 y entre Team_sports e Individual_sports es igual a 2 también. Por tanto, la mínima es 2.

Si asumimos que en el peor clasificador posible, tendríamos que haber recorrido todas las aristas del grafo al menos una vez, la distancia quedaria fijada en el valor 2/4

Sumando ambas distancias parciales, tenemos 4/4. Ahora dividimos por el valor 2 * Etiquetas_predichas * Etiquetas_reales quedándonos la distancia total del ejemplo igual a 1/8

Sumando las distancias parciales de todos los ejemplos, tenemos 1/16 +

$0 + 1/8 = 3/16$

4.6.1. Pseudocódigo

```
DISTANCIA_SIMETRICA (Grafo G, Array Etiquetas Predichas, Array Etiquetas Reales)
 dist = 0
 Para cada ejemplo:
   ep = etiquetas_predichas_ejemplo
   er = etiquetas_reales_ejemplo
   Si ep vacio:
      dist = 1
   En caso contrario para cada etiqueta en ep:
       encontrada = buscar(etiqueta, er)
       Si no se encuentra:
           dist + = distancia_minima(etiqueta, er)
   Si er vacio:
       dist = 1
   En caso contrario para cada etiqueta en er:
       encontrada = buscar(etiqueta, ep)
       Si no se encuentra:
           dist + = distancia_minima(etiqueta, ep)
   dist = dist/2*|ep|*|er|
devolver dist/numero_ejemplos
```

Capítulo 5

Experimentos

Siguiendo la metología explicada en el apartado anterior, se han construido 5 datasets con datos obtenidos de la Wikipedia. Las temáticas elegidas son deportes, profesiones, bebidas, software y comidas. En la siguiente tabla se muestran las características de cada uno de los datasets.

dataset	ejemplos	etiquetas	cardinalidad	densidad	combinaciones
deportes	705	10	1.2	0.12	30
software	714	12	2.21	0.18	33
profesiones	344	11	1.014	0.09	15
bebidas	189	11	2.06	0.19	33
comidas	1892	28	1.06	0.037	85

Tabla 5.1: Métricas descripción de los datasets

Sobre ellos, se ha aplicado el método de transformación de problemas Binary Relevance con 5 algoritmos de categorización distintos: Naives Bayes, CART (Decision Trees), Logistic, Knn y LinearSVC (Support Vector Machines).

Para medir la precisión de los modelos, se han utilizado 7 métricas: hamming loss, accuracy, precision, recall, Macro-F1 Score, Micro-F1 Score y la distancia simétrica definida en este trabajo.

Para garantizar la independencia de los resultados obtenidos de los datos, los experimentos han sido realizados usando la técnica de validación cruzada k-folds con k = 10.

En las siguientes figuras se puede apreciar el valor de cada una de las métricas en funcion de los algoritmos utilizados para construir el modelo.

5.1. Grafos de categorías

En este apartado se muestran los grafos de categorías correspondientes a cada uno de los datasets. Se puede observar que son grafos conexos, lo que nos permite calcular distancias entre cualquiera de sus nodos.

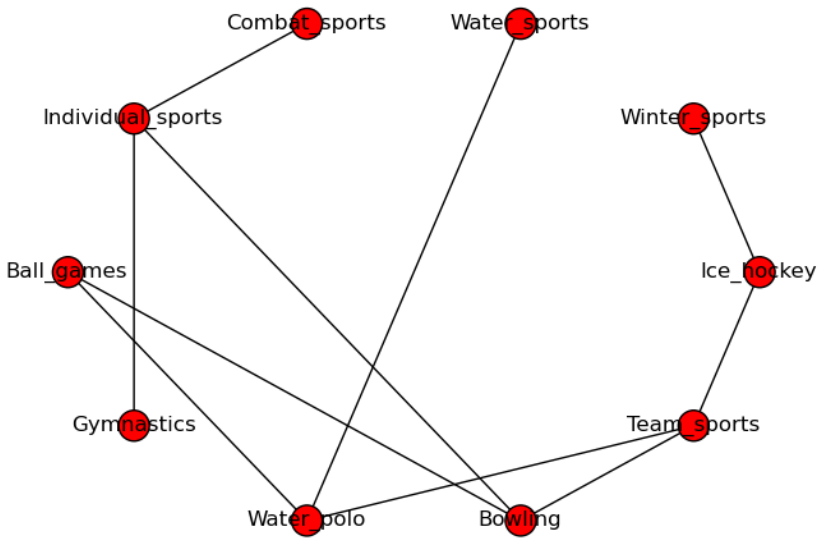


Figura 5.1: Grafo categorías dataset deportes

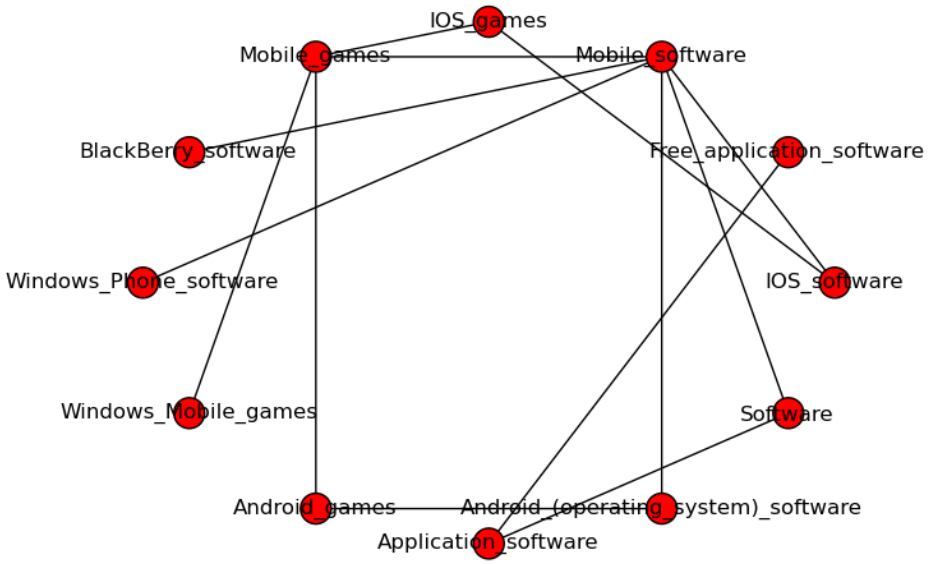


Figura 5.2: Grafo categorías dataset software

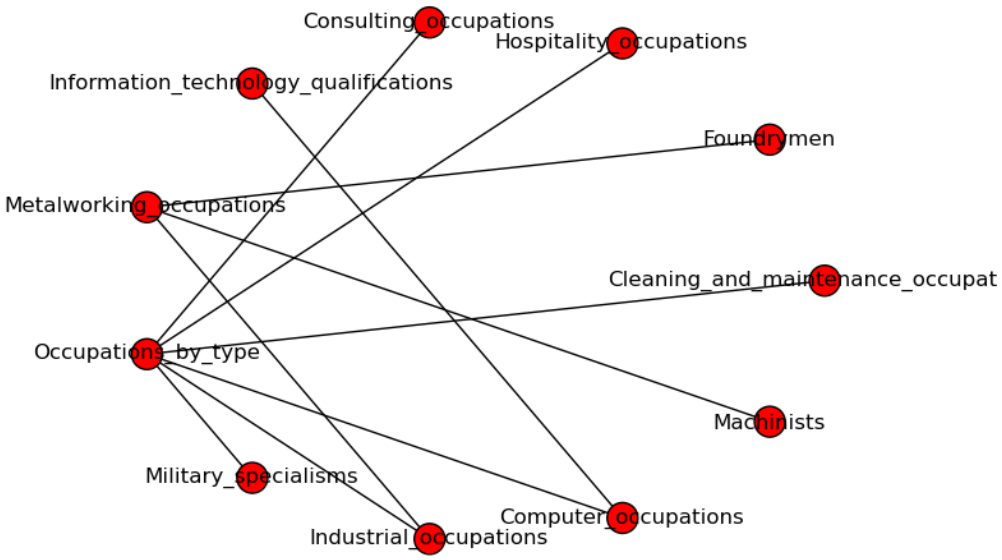


Figura 5.3: Grafo categorías dataset profesiones

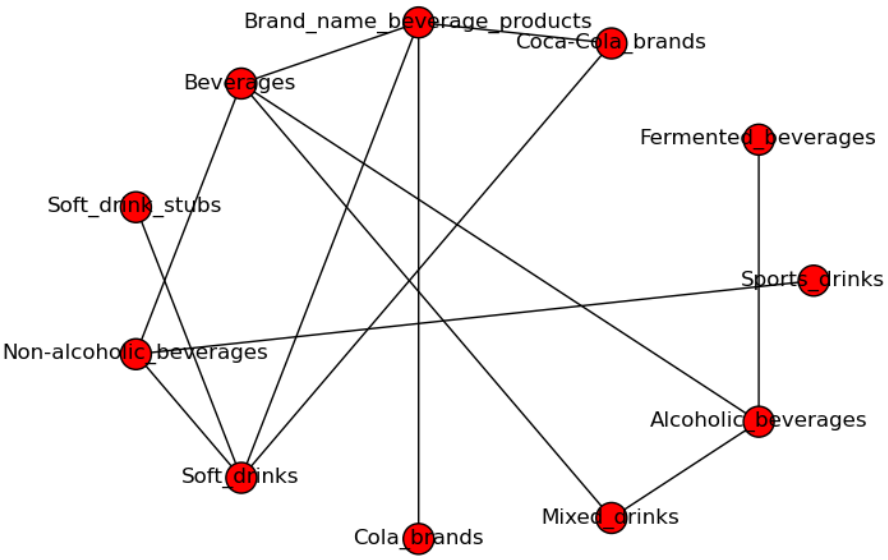


Figura 5.4: Grafo categorías dataset bebidas

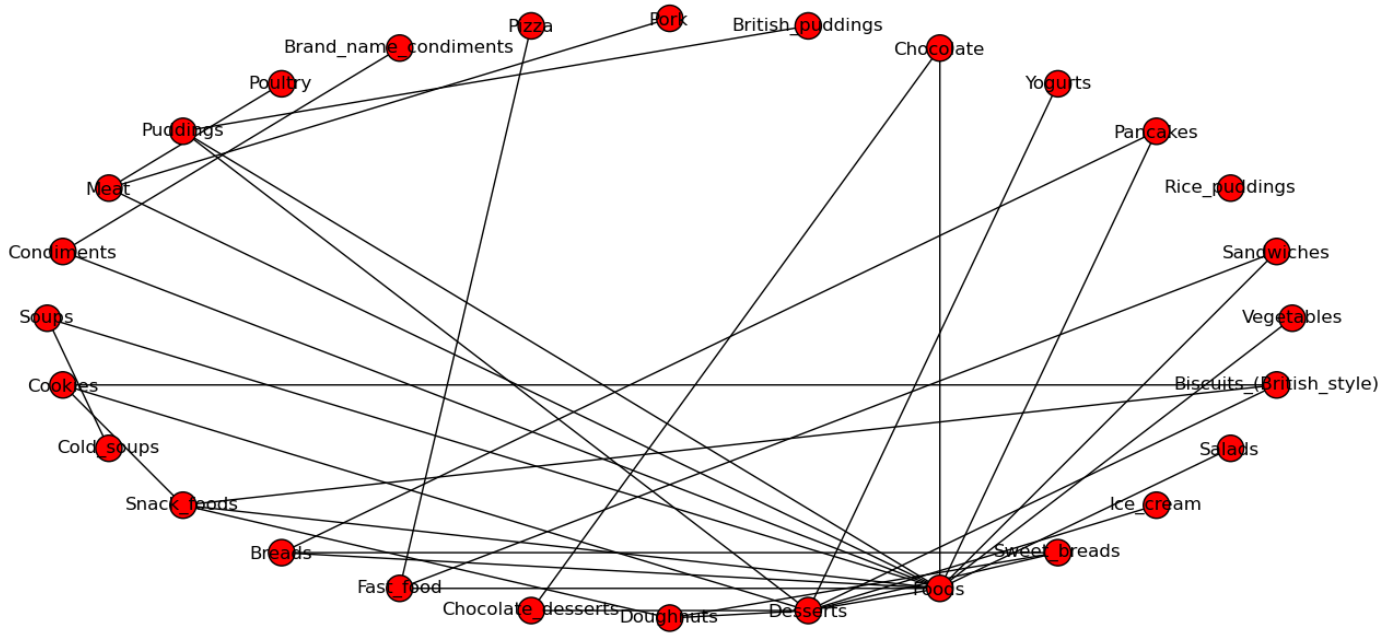


Figura 5.5: Grafo categorías dataset comidas

5.2. Resultados

En las siguientes tablas se muestran los resultados obtenidos para cada uno de los datasets.

	Relevancia Binaria				
Métrica	Naives Bayes	Knn	Logistic	SVM	DT
Hamming loss	0.119	0.062	0.092	0.064	0.099
Precision	0.180	0.768	0.702	0.825	0.618
Recall	0.011	0.695	0.286	0.584	0.548
Distancia Simétrica	0.092	0.038	0.072	0.043	0.058
Macro-F1	0.009	0.676	0.234	0.552	0.546
Micro-F1	0.022	0.729	0.423	0.683	0.570
Accuracy	0.009	0.567	0.211	0.468	0.327

Tabla 5.2: Resultados para el dataset deportes

	Relevancia Binaria				
Métrica	Naives Bayes	Knn	Logistic	SVM	DT
Hamming loss	0.083	0.067	0.051	0.048	0.079
Precision	0.749	0.832	0.782	0.847	0.808
Recall	0.682	0.801	0.815	0.837	0.796
Distancia Simétrica	0.012	0.007	0.005	0.004	0.008
Macro-F1	0.270	0.391	0.340	0.388	0.418
Micro-F1	0.767	0.828	0.866	0.875	0.801
Accuracy	0.541	0.539	0.655	0.661	0.501

Tabla 5.3: Resultados para el dataset software

	Relevancia Binaria				
Métrica	Naives Bayes	Knn	Logistic	SVM	DT
Hamming loss	0.0953	0.0389	0.0886	0.0558	0.0672
Precision	0.2554	0.8186	0.3637	0.7765	0.7079
Recall	0.0599	0.7288	0.1260	0.4785	0.6560
Distancia Simétrica	0.088	0.033	0.079	0.049	0.045
Macro-F1	0.1043	0.6292	0.133	0.3982	0.4960
Micro-F1	0.1119	0.7907	0.2229	0.6339	0.6629
Accuracy	0.0609	0.7241	0.1279	0.4766	0.4909

Tabla 5.4: Resultados para el dataset profesiones

	Relevancia Binaria				
Métrica	Naives Bayes	Knn	Logistic	SVM	DT
Hamming loss	0.0391	0.0243	0.0351	0.0249	0.03558
Precision	0.0556	0.7963	0.4451	0.8339	0.5754
Recall	0.0154	0.5201	0.1072	0.4115	0.5306
Distancia Simétrica	0.0356	0.0212	0.0319	0.0217	0.0249
Macro-F1	0.0171	0.5547	0.1266	0.4573	0.4697
Micro-F1	0.0264	0.6251	0.1920	0.5628	0.5364
Accuracy	0.0609	0.5079	0.1073	0.3969	0.3430

Tabla 5.5: Resultados para el dataset comidas

	Relevancia Binaria				
Métrica	Naives Bayes	Knn	Logistic	SVM	DT
Hamming loss	0.144	0.118	0.139	0.105	0.124
Precision	0.413	0.663	0.460	0.702	0.674
Recall	0.39	0.613	0.407	0.564	0.652
Distancia Simétrica	0.0428	0.02752	0.04715	0.0279	0.02579
Macro-F1	0.12	0.299	0.133	0.254	0.362
Micro-F1	0.50	0.659	0.523	0.666	0.662
Accuracy	0.06	0.328	0.095	0.290	0.216

Tabla 5.6: Resultados para el dataset bebidas

	Relevancia Binaria				
Métrica	Naives Bayes	Knn	Logistic	SVM	DT
Hamming loss	0	3	0	2	0
Precision	0	1	0	4	0
Recall	0	2	0	1	2
Distancia Simétrica	0	3	0	1	1
Macro-F1	0	3	0	1	1
Micro-F1	0	3	0	2	0
Accuracy	0	3	0	2	0

Tabla 5.7: Tabla algoritmo ganador métrica

	Relevancia Binaria				
	Naives Bayes	Knn	Logistic	SVM	DT
Victorias	0	18	0	13	4

Tabla 5.8: Tabla victorias totales algoritmo

De los resultados podemos deducir que para el enfoque de relevancia binaria, Knn y SVM (en nuestro caso hemos usado el algoritmo LinearSVC)

son los algoritmos que mejores resultados obtienen, siendo mejores prácticamente en la totalidad de las métricas de los 4 conjuntos de datos con los que se ha experimentado en este trabajo.

Por el contrario, Naives Bayes y Logistic, no han conseguido obtener el mejor resultado en ninguna métrica, siendo los que peor parados han salido en estos experimentos.

El algoritmo CART (Decision Tree), ha conseguido salir victorioso en 4 ocasiones, quedando muy cerca en las ocasiones que no ha ganado de SVM y Knn.

Por tanto de los resultados de éstos experimentos podemos concluir que Knn, SVM y DT son los mejores algoritmos para clasificación multi-etiqueta mediante el enfoque de relevancia binaria.

En algunos datasets se observan diferencias importantes entre el Macro-F1 y el Micro-F1. Esto es debido al desbalanceo entre las clases.

Recordemos que otro objetivo que perseguíamos con el presente trabajo, era validar experimentalmente la métrica definida. Pues bien, se pueden observar en los resultados, que la distancia simétrica se comporta de manera similar al resto de métricas, es decir, cuando los resultados para un clasificador del resto de métricas son positivos, también lo es la distancia simétrica y viceversa, lo que nos indica que es una métrica valida para entornos jerárquicos.

Finalmente, cabe resaltar que la métrica propuesta es la única de las que comparamos en el trabajo que es capaz de utilizar la jerarquia de etiquetas a la hora de estimar la calidad de un clasificador. Esto se debe a que emplea las distancias entre la etiqueta predicha y la real, dando un valor peor a las etiquetas más lejanas, y un mejor valor a las más cercanas. El resto de las medidas estudiadas ignoran las distancias entre las etiquetas, tomando el mismo valor para todos los errores en las predicciones de etiquetas.

Capítulo 6

Conclusiones

Las conclusiones obtenidas de este trabajo son la siguientes:

1. Una vez se conoce el API Rest de Wikipedia y se tiene hecho el script para realizar los datasets, es sencillo construir conjunto de datos de cualquier temática y con diversas características, lo que implica no estar atado a datasets de terceros para realizar los experimentos y además, trabajar con datos reales.

2. El hecho de trabajar con datos reales, implica ciertos problemas como desbalanceo entre las clases, dificultad para encontrar conjuntos de datos pequeños con una cardinalidad media de etiqueta grande. A pesar de ser Wikipedia, donde se supone que los textos están revisados y son bastante correctos, he observado que la categorización de los mismos es bastante mejorable. A veces existen dos categorías distintas para expresar lo mismo o textos cuyo etiquetado quizás no es el más correcto.

3. Scikit-learn es un gran proyecto con una comunidad bastante amplia detrás. No es necesario ser un experto en python ni en aprendizaje automático para utilizarlo. No obstante, necesita dar un paso más en cuanto a clasificación multi-etiqueta se refiere, y ampliar la librería a otros enfoques distintos de relevancia binaria.

4. La métrica definida en el presente problema, funciona en consonancia con el resto de métricas, lo que nos indica que es una buena métri-

ca cuando se trata de medir la precisión de los modelos de manera jerárquica.

5. Se han establecido las bases de un problema real de clasificación jerárquica multi-etiqueta. Futuros y algoritmos y métricas podrán ser integrados y validados empíricamente de una manera sencilla.

El presente trabajo es introductorio a la clasificación multi-etiqueta en entornos jerárquicos. Posteriormente, se pueden realizar multitud de cosas, entre las que podemos citar:

1. Ampliar el estudio, abarcando más métodos y algoritmos de clasificación multi-etiqueta y algoritmos de clasificación multi-etiqueta jerárquica y comprobar si la distancia simétrica se sigue comportando en consonancia con el resto de métricas.
2. Extrapolar el estudio realizado con textos a otros campos (imágenes, música o entornos bioinformáticos).
3. Redefinir la distancia simétrica, variando los pesos de las aristas que forman el grafo.
4. Definir más metricas basadas en distancias, por ejemplo teniendo en cuenta distancias que consideran profundidad o amplitud de una manera diferente.
5. Definir algoritmos que se apoyen en la jerarquía para realizar la clasificación.
6. Si se dispusiera de máquinas más potentes y paralelizando cálculos, se podría realizar un estudio a gran escala.
7. En caso de encontrar un algoritmo que proporcionara unos resultados lo suficientemente aceptables, se podría pensar en construir una extensión para mediawiki que recomendara etiquetas.

A nivel personal ha sido una experiencia muy satisfactoria. Al comienzo de este trabajo, mi conocimientos sobre clasificación multi-etiqueta, clasificación jerárquica y sobre el API de la Wikipedia eran nulos y mis conocimientos de Python básicos. La realización de este trabajo ha contribuido de manera esencial a la ampliación de los mismos y me permite finalizar este máster con unas nociones mucho mayores en el área de minería y análisis de datos.

Bibliografía

[1] http://www.ine.es.

[2] http://www.mediawiki.org. [Online; accessed 20-April-2014].

[3] http://www.scikit-learn.org. [Online; accessed 12-February-2014].

[4] http://www.nltk.org. [Online; accessed 20-April-2014].

[5] http://networkx.github.io/. [Online; accessed 20-April-2014].

[6] http://www.kaggle.com. [Online; accessed 10-March-2014].

[7] http://www.wikipedia.org.

[8] http://www.ibm.com/developerworks/ssa/xml/library/x-phpwikipedia/. [Online: accessed 20-March-2014].

[9] http://json.org/.

[10] http://www.python.org/.

[11] A. Freitas A. de Carvalho. A tutorial on hierarchical classification with applications in bioinformatics. 2007.

[12] A. Freitas A. de Carvalho. A tutorial on multi-label classification techniques. 2009.

[13] Antonino Feitosa Neto Araken M Santos, Anne M P Canuto. Evaluating classification methods to multi-label tasks in different domains. 2010.

[14] Schapire R. Troyanskaya O Barutcuoglu, Z. Hierarchical multi-lab el prediction of gene function. 2006.

[15] G. Brassard and P. Bratley. *Fundamentos de algoritmia*. Prentice Hall, 1987.

[16] Alex A. Freitas Carlos N., Silla JR. A survey of hierarchical classification across different application domains. 2010.

[17] J. Furnkranz and E. Hullermeier. Pairwise preference learning and ranking, in in proc. ecml-03, cavtat-dubrovnik. 2003.

[18] I. Vlahavas G. Tsoumakas, I. Katakis. Data mining and knowledge discovery handbook, ch. mining multi-label data. 2009.

[19] Dejan Gjorgjevikj Saso Dzeroski Gjorgii Madjarov, Dragi Kocev. An extensive experimental comparison of methods for multi-label learning. 2012.

[20] José Hernández Orallo, María José Ramírez Quintana, and César Ferri Ramírez. *Introducción a la Minería de Datos*. Pearson Educación, 2004.

[21] X. Zhou J. Chen and Z. Wu. A multi-label chinese text categorization system based on boosting algorith. 2004.

[22] Sahami M. Koller, D. Classifying do cuments using very few words. in pro c.of the 14th international conf. on machine learning (1997) 170-178. 1997.

[23] A. M. Rubinov M. A. Mammadov and J. Yearwood. The study of drug-reaction relationships using global optimization techniques. 2007.

[24] Flkhri Karray Mila AlemZadeh, Richard khoury. Query classification using wikipedia's category graph. 2012.

[25] C. Gentile N. Cesa-Bianchi and L. Zaniboni. Hierarchical classification: combining bayes with svm. 2006.

[26] C. Gentile N. Cesa-Bianchi and L. Zaniboni. Incremental algorithms for hierarchical classification. 2006.

[27] R. Polikar. Ensemble based systems in decision making. 2006.

[28] Matthew A. Russell. *Mining the Social Web*. O'Reilly Media, 2011.

[29] Toby Segaran. *Programming Collective Intelligence*. O'Reilly Media, 2007.

[30] Lim EP Sun A. Hierarchical text classification and evaluation. 2001.

[31] Sašo Džeroski Valentin Gjorgjioski, Dragi Kocev. Comparison of distances for multi-label classification with pcts.

[32] Struyf J. Schietgat L. Dzeroski S. Blo ckeel H. Vens, C. Decision trees for hierarchical multi-label classification. 2008.

Anexo

código python del script creado para la parte experimental de este trabajo

```
#!/usr/bin/env python
#coding:utf-8
import string
import json
import urllib2
import sys
from urllib2 import urlopen
import networkx as nx
import matplotlib.pyplot as plt
import os
from mwtextextractor import get_body_text
import logging
import re
import random
import shutil
from time import time
from nltk.corpus import stopwords
from nltk import word_tokenize
from nltk.tokenize import SpaceTokenizer
import numpy as np
from sklearn import svm, tree
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.svm import LinearSVC
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.multiclass import OneVsRestClassifier,
    OneVsOneClassifier
from sklearn.linear_model import SGDClassifier
from sklearn.naive_bayes import MultinomialNB, GaussianNB,
```

```
    BernoulliNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction import DictVectorizer
from sklearn.metrics import accuracy_score, hamming_loss,
    recall_score, confusion_matrix, classification_report, f1_score,
    precision_score
from sklearn.datasets import load_files
from sklearn import cross_validation
from sklearn.preprocessing import MultiLabelBinarizer
from scipy.sparse import *
from scipy import *
import numpy as np

#Construye un grafo de a partir del nombre de un conjunto de
    categorias de la Wikipedia
def construir_grafo(categorias):
  G = nx.Graph()
  for categoria in categorias:
    G.add_node(categoria)
    url = "http://en.wikipedia.org/w/api.php?action=query
 &list=categorymembers&cmtype=subcat&cmtitle=Category:" + categoria
     + "&cmlimit=500&format=json&clshow=!hidden"
    data = urllib2.urlopen(url).read()
    data = json.loads(data)
    for raw in data['query']['categorymembers']:
      titulo = raw['title'].replace(' ', '_')
      for categ in categorias:
        if (titulo[9:] == categ):
          G.add_edge(categoria, categ)
  return G

#Funcion auxiliar para guardar los textos en formato json
def escribir_archivo(nombre_archivo, ruta, contenido):
        ruta = ruta + nombre_archivo.replace("/","_")
        f = open(ruta,'a')
        f.write((json.dumps(contenido)))
        f.close()

#Descarga los articulos pertenecientes a una categoria concreta
def descargar_paginas(categoria, ruta, min_car_example):
```

```
url = "http://en.wikipedia.org/w/api.php?action=query
&list=categorymembers&cmtype=page&cmtitle=Category:" +
    categoria + "
&cmlimit=500&format=json&clshow=!hidden"
data = urllib2.urlopen(url).read()
data = json.loads(data)
for raw in data['query']['categorymembers']:
        titulo = raw['title'].replace(' ', '_')
        if not (os.path.exists(ruta + titulo.replace("/","_")
            )):
                url =
                    "http://en.wikipedia.org/w/api.php?action=query&
                prop=revisions&rvprop=content&redirects=1&titles="
                + titulo + "&format=json&clshow=!hidden"
                page = urllib2.urlopen(url).read()
                page = json.loads(page)
                texto =
                    get_body_text(str(page['query']['pages']))
                texto = preprocesa_texto(texto)

                url_cat =
                    "http://en.wikipedia.org/w/api.php?action=query&titles="
                    + titulo +
                    "&prop=categories&format=json&clshow=!hidden"
                data =urllib2.urlopen(url_cat).read()
                data = json.loads(data)
                for key in page['query']['pages']:
                        page_id = key
                try:
                  for key, value in
                      data['query']['pages'][page_id].iteritems():
                         if key == 'categories':
                                itemlist = value

                  cat_example = []
                  categorias = '['
                  for c in itemlist:
                        categ = str((c['title'].replace(" ",
                            "_").replace("Category:", " ")))
                        categoria =
                            '{"category":"'+categ.strip()+'"},'
                        categorias += categoria
                        cat_example.append(categ.strip())
```

```
                        categorias = categorias[:-1] +"]"
                        json_content = {'title': titulo,
                                        'categories': categorias,
                                        'text': texto}
                        if (numero_categorias(G, cat_example) >=
                            min_car_example):
                          escribir_archivo (titulo, ruta,
                              json_content)
                      except:
                        print 'No existe el archivo'

def numero_categorias(G, categories):
    number = 0
    for c in G.nodes():
      for cat in categories:
        if c == cat:
          number += 1
    return number

def preprocesa_texto(texto):
    texto = re.sub("\[\[Category:.*?\]\]", "", texto)
    texto = ''.join([i for i in texto if not i.isdigit() and not i
        in set(string.punctuation)])
    return texto


def calcular_y(y, kf):
  y_c = []
  for element in kf:
    y_c.append(y[element])
  return y_c

def calcular_X(X, kf):
    X_c = []
    for element in kf:
      X_c.append(X[element])
    return X_c

#Funcion que modela y evalua los modelos generados
def clasificar(G):
    target_names = G.nodes()
    wiki_raw = load_files(rutaBase, 'archivos', 'raw')
    X = []
```

```
for f in wiki_raw.filenames:
        texto =json.loads((open(f).read()))['text']
        X.append(str(texto))


filas = len(wiki_raw.filenames)
columnas = len(target_names)

y = []

i = 0
for f in wiki_raw.filenames:
        data = (json.loads((open(f).read())))
        data = json.loads(data['categories'])
        aux = []
        for j in range (0, len(target_names)):
                encontrado = False
                for item in data:
                    if not(encontrado):
                        if (item['category'] == target_names[j]):
                                aux.append(j)
                                encontrado = True
        y.append(aux)


classifier_nb = Pipeline([
    ('tfidf', TfidfVectorizer(stop_words = "english")),
    ('clf', OneVsRestClassifier(BernoulliNB()))])

classifier_knn = Pipeline([
    ('tfidf', TfidfVectorizer(stop_words = "english")),
    ('clf', OneVsRestClassifier(KNeighborsClassifier()))])

classifier_log = Pipeline([
    ('tfidf', TfidfVectorizer(stop_words = "english")),
    ('clf', OneVsRestClassifier(LogisticRegression()))])

classifier_svc = Pipeline([
    ('tfidf', TfidfVectorizer(stop_words = "english")),
    ('clf', OneVsRestClassifier(LinearSVC()))])
```

```
kf = cross_validation.KFold(len(X), 10)

hmnb = 0
pnb = 0
rsnb = 0
dsnb = 0
f1micronb = 0
f1macronb = 0
acnb = 0


hmknn = 0
pknn = 0
rsknn = 0
dsknn = 0
f1microknn = 0
f1macroknn = 0
acknn = 0

hmlog = 0
plog = 0
rslog = 0
dslog = 0
f1microlog = 0
f1macrolog = 0
aclog = 0

hmsvc = 0
psvc = 0
rssvc = 0
dssvc = 0
f1microsvc = 0
f1macrosvc = 0
acsvc = 0

hmdt = 0
pdt = 0
rsdt = 0
dsdt = 0
f1microdt = 0
f1macrodt = 0
acdt = 0
```

```
for X_tra, X_te in kf:
  X_train = calcular_X(X, X_tra)
  X_test = calcular_X(X, X_te)
  y_train = calcular_y(y, X_tra)
  y_test = calcular_y(y, X_te)

  mlb = MultiLabelBinarizer().fit(y)
  yb_train = mlb.transform(y_train)
  yb_test = mlb.transform(y_test)

  classifier_nb.fit(X_train, yb_train)
  predicted_nb = classifier_nb.predict(X_test)
  hmnb += hamming_loss(yb_test, predicted_nb)
  pnb += precision_score(yb_test, predicted_nb)
  rsnb += recall_score(yb_test, predicted_nb)
  dsnb += distancia_simetrica(y_test,
      mlb.inverse_transform(predicted_nb), G)
  f1micronb += f1_score(yb_test, predicted_nb, average='micro')
  f1macronb += f1_score(yb_test, predicted_nb, average='macro')
  acnb += accuracy_score(yb_test, predicted_nb)

  classifier_knn.fit(X_train, yb_train)
  predicted_knn = classifier_knn.predict(X_test)
  hmknn += hamming_loss(yb_test, predicted_knn)
  pknn += precision_score(yb_test, predicted_knn)
  rsknn += recall_score(yb_test, predicted_knn)
  dsknn += distancia_simetrica(y_test,
      mlb.inverse_transform(predicted_knn), G)
  f1microknn += f1_score(yb_test, predicted_knn, average='micro')
  f1macroknn += f1_score(yb_test, predicted_knn, average='macro')
  acknn += accuracy_score(yb_test, predicted_knn)

  classifier_log.fit(X_train, yb_train)
  predicted_log = classifier_log.predict(X_test)
  hmlog += hamming_loss(yb_test, predicted_log)
  plog += precision_score(yb_test, predicted_log)
  rslog += recall_score(yb_test, predicted_log)
  dslog += distancia_simetrica(y_test,
      mlb.inverse_transform(predicted_log), G)
  f1microlog += f1_score(yb_test, predicted_log, average='micro')
  f1macrolog += f1_score(yb_test, predicted_log, average='macro')
  aclog += accuracy_score(yb_test, predicted_log)
```

```
  classifier_svc.fit(X_train, yb_train)
  predicted_svc = classifier_svc.predict(X_test)
  hmsvc += hamming_loss(yb_test, predicted_svc)
  psvc += precision_score(yb_test, predicted_svc)
  rssvc += recall_score(yb_test, predicted_svc)
  dssvc += distancia_simetrica(y_test,
      mlb.inverse_transform(predicted_svc), G)
  f1microsvc += f1_score(yb_test, predicted_svc, average='micro')
  f1macrosvc += f1_score(yb_test, predicted_svc, average='macro')
  acsvc += accuracy_score(yb_test, predicted_svc)

  tf = TfidfVectorizer(stop_words = "english").fit(X_train)
  tfXtrain = TfidfVectorizer(stop_words =
      "english").fit_transform(X_train)

  classifier_dt = OneVsRestClassifier(DecisionTreeClassifier())
  classifier_dt.fit(tfXtrain.toarray(), yb_train)
  tfXtest = tf.transform(X_test)
  predicted_dt = classifier_dt.predict(tfXtest.toarray())
  hmdt += hamming_loss(yb_test, predicted_dt)
  pdt += precision_score(yb_test, predicted_dt)
  rsdt += recall_score(yb_test, predicted_dt)
  dsdt += distancia_simetrica(y_test,
      mlb.inverse_transform(predicted_dt), G)
  f1microdt += f1_score(yb_test, predicted_dt, average='micro')
  f1macrodt += f1_score(yb_test, predicted_dt, average='macro')
  acdt += accuracy_score(yb_test, predicted_dt)



print 'hamming_loss_nb', hmnb/float(10)
print 'precicion_nb', pnb/float(10)
print 'recall_nb', rsnb/float(10)
print 'ds_nb', dsnb/float(10)
print 'f1_nb_micro', f1micronb/float(10)
print 'f1_nb_macro', f1macronb/float(10)
print 'acnb ', acnb/float(10) , '\n'

print 'hamming_loss_knn', hmknn/float(10)
print 'precision_knn', pknn/float(10)
print 'recall_knn', rsknn/float(10)
print 'ds_knn', dsknn/float(10)
```

```
    print 'f1_knn_micro', f1microknn/float(10)
    print 'f1_knn_macro', f1macroknn/float(10)
    print 'acknn', acknn/float(10), '\n'

    print 'hamming_loss_log', hmlog/float(10)
    print 'precision_log', plog/float(10)
    print 'recall_log', rslog/float(10)
    print 'ds_log', dslog/float(10)
    print 'f1_log_micro', f1microlog/float(10)
    print 'f1_log_macro', f1macrolog/float(10)
    print 'aclog ', aclog/float(10), '\n'

    print 'hamming_loss_svc', hmsvc/float(10)
    print 'precision_svc', psvc/float(10)
    print 'recall_svc', rssvc/float(10)
    print 'ds_svc', dssvc/float(10)
    print 'f1_micro_svc', f1microsvc/float(10)
    print 'f1_macro_svc', f1macrosvc/float(10)
    print 'acsvc' , acsvc/float(10), '\n'


    print 'hamming_loss_dt', hmdt/float(10)
    print 'precision_dt', pdt/float(10)
    print 'recall_dt', rsdt/float(10)
    print 'ds_dt', dsdt/float(10)
    print 'f1_micro_dt', f1microdt/float(10)
    print 'f1_macro_dt', f1macrodt/float(10)
    print 'acdt' , acdt/float(10), '\n'


    print 'cardinalidad etiquetas ', cardinalidad_etiquetas(y)
    print 'densidad etiquetas', densidad_etiquetas(y, G)
    print 'combinaciones etiquetas', combinaciones_etiquetas(y)
    print 'numero etiquetas', len(G)

    print y
    print predicted_knn

def cardinalidad_etiquetas(y):
  cardinality = 0
  for element in y:
     cardinality = cardinality + len(element)
  cardinality = float(cardinality) / float(len(y))
```

```
  return cardinality


def densidad_etiquetas(y, G):
  density = 0
  for element in y:
     density = density + len(element)
  density = (float(density) / float(len(y))) / len(G)
  return density

def combinaciones_etiquetas(y):
  distintos = []
  for i in range (0, len(y)):
    igual = False
    for j  in range (0, len(distintos)):
     if not igual:
       ejem1  = y[i]
       ejem2  = distintos[j]
       if(len(ejem1) == len(ejem2)):
         igual = True
         for k in range (0, len(y[i])):
            if(ejem1[k] != ejem2[k]):
               igual = False
    if not igual:
      distintos.append(y[i])
      igual = True

  return len(distintos)

#Funcion que implementa la distancia simetrica definida en el
    presente trabajo
def distancia_simetrica(y_test, predicted, G):
  target_names = G.nodes()
  md = []
  distancia_total = 0
  distancia_parcial = 0
  for i in range(0, len(predicted)):
         pred = predicted[i]
         test = y_test[i]
         encontrado = False
         for p in pred:
            for t in test:
               if p == t:
```

```
                encontrado = True
            if not(encontrado):
                distancia_parcial =
                    (float(1)/float((len(target_names)* len(pred))))
                for t in test:
                    dist, path = nx.bidirectional_dijkstra(G,
                        target_names[t], target_names[p])
                    if dist < distancia_parcial:
                        distancia_parcial =
                            (float(dist)/float(len(target_names)))
                        distancia_parcial = (float(distancia_parcial)
                            / float(len(test)))
                distancia_total += distancia_parcial
                encontrado = False

        for t in test:
            for p in pred:
                if t == p:
                    encontrado = True
            if not(encontrado):
                distancia_parcial =
                    (float(1)/float((len(target_names)*len(test))))
                for p in pred:
                    dist, path = nx.bidirectional_dijkstra(G,
                        target_names[t], target_names[p])
                    if dist < distancia_parcial:
                        distancia_parcial =
                            float(dist)/float((len(target_names)))
                        distancia_parcial =
                            (float(distancia_parcial)/float((len(pred))))
                distancia_total += distancia_parcial
                encontrado = False
    distancia_total = float(distancia_total) / float(len(y_test))
    return distancia_total

categSoftware = ['Mobile_software',
    'Android_(operating_system)_software', 'BlackBerry_software',
    'IOS_software', 'Windows_Phone_software', 'Android_games',
    'Mobile_games', 'Android_games', 'Windows_Mobile_games',
    'IOS_games', 'Software', 'Application_software',
    'Free_application_software' ]

categDeportes = [ 'Ball_games', 'Individual_sports', 'Team_sports',
```

```
    'Combat_sports', 'Water_sports', 'Water_polo', 'Bowling',
    'Winter_sports', 'Ice_hockey', 'Gymnastics']

categBebidas = ['Beverages', 'Non-alcoholic_beverages',
    'Alcoholic_beverages', 'Sports_drinks', 'Soft_drinks',
    'Fermented_beverages', 'Mixed_drinks',
    'Brand_name_beverage_products', 'Coca-Cola_brands',
    'Cola_brands', 'Soft_drink_stubs']

categProfesiones = ['Occupations_by_type', 'Computer_occupations',
    'Information_technology_qualifications',
    'Cleaning_and_maintenance_occupations',
    'Industrial_occupations', 'Metalworking_occupations',
    'Machinists', 'Foundrymen', 'Consulting_occupations',
    'Hospitality_occupations', 'Military_specialisms']

categComidas = ['Foods', 'Chocolate', 'Desserts', 'Meat',
    'Sandwiches', 'Snack_foods', 'Soups', 'Salads', 'Vegetables',
    'Condiments', 'Breads', 'Sweet_breads', 'Pancakes',
    'Chocolate_desserts', 'Cookies', 'Biscuits_(British_style)',
    'Ice_cream', 'Yogurts', 'Puddings', 'British_puddings',
    'Rice_puddings', 'Doughnuts', 'Sweet_breads', 'Fast_food',
    'Pizza', 'Brand_name_condiments', 'Poultry', 'Pork',
    'Cold_soups']
```