Universidad
Carlos III de Madrid

Proyecto Fin de Carrera

# Aplicación de técnicas de aprendizaje automático para la extracción de información en textos farmacológicos

Autor: Beatriz Nombela Escobar

Tutor: Isabel Segura Bedmar

Leganés, julio de 2011

# Agradecimientos

A Isabel, por ofrecerme la oportunidad de realizar este proyecto, por su total disponibilidad y porque, a pesar de las dificultades, realmente he disfrutado colaborando con una persona a la que le interesa lo que hace y es buena en ello. Por su gran labor de investigadora la admiro y envidio.

Y a Jose, mi chico, porque sin tus ánimos nunca me habría atrevido a involucrarme en todo lo que hago ahora. Por estar siempre a mi lado. Porque me has revolucionado y lo seguirás haciendo.

## Resúmen

En la actualidad los profesionales del dominio biomédico necesitan tener información actualizada de su campo para llevar a cabo su trabajo de manera fiable y profesional. Dentro del dominio biomédico, la administración de fármacos requiere saber de antemano si dos fármacos interaccionan entre sí, ya que esta interacción puede provocar efectos no deseados en la salud del paciente. Los profesionales cuentan con ingentes cantidades de información, ya sea en textos biomédicos no estructurados o en bases de datos; es por esto que se necesita un método automático para extraer información de estas fuentes de datos para poder detectar interacciones entre fármacos.

En este proyecto se van a estudiar distintas técnicas de aprendizaje automático supervisado para detectar posibles interacciones entre dos fármacos. Partiendo del corpus DrugDDI, creado en la tesis *Application of Information Extraction techniques to pharmacological domain: Extracting drug-drug interactions* [1], se van a aplicar diferentes algoritmos para su posterior estudio y comparación con los resultados obtenidos en dicha tesis.

## Abstract

In the biomedical domain, interaction between two or more drugs is a desired knowing in drugs administration, as that interaction can provoke undesirable effects over a patient health. Medical professional have access to huge amounts of data, whether they are in biomedical unstructured texts or in databases. For this reason it is desirable an automatic method to extract useful information from this data sources for processing and detecting drugs interactions.

In this project we are going to introduce some supervised machine learning techniques in order to detect possible interactions between two drugs. Based on the DrugDDI corpus, gathered in the thesis *Application of Information Extraction techniques to pharmacological domain: Extracting drug-drug interactions* [1], we are going to apply different algorithms for its later research and comparison with the results obtained in the thesis.

# Índice de contenido

# Índice de ilustraciones



# Índice de tablas

# Introducción

## Qué es una interacción farmacológica

Una interacción farmacológica se produce cuando un fármaco modifica la acción de otro al administrarse ambos de manera simultánea, pudiendo aumentar, disminuir o incluso inhibir por completo sus efectos. De ahí la importancia que conlleva la administración de varios fármacos conjuntamente. Algunas interacciones pueden ser beneficiosas, por lo que puede ser incluso deseable su toma, pero desafortunadamente también pueden ser bastante peligrosas, ya que pueden intensificar o disminuir los efectos del otro fármaco o empeorar sus efectos secundarios, lo cual puede perjudicar al paciente o incluso provocar su muerte.

En el campo de la medicina siempre se intentan tener en cuenta dichas interacciones debido al riesgo que conllevan; existen bases de datos con interacciones entre fármacos, pero dada la baja tasa de actualización de estas bases de datos, la cantidad de información de fármacos que existe y el continuo incremento de datos con nuevos fármacos o nuevas investigaciones, un profesional de la medicina se puede ver desbordado por tantos datos e incapaz de conocer ciertas interacciones entre fármacos en un tiempo realista, por lo que son necesarios sistemas informáticos automatizados que puedan proporcionar dicha información. Aun así hoy en día no existe un sistema que sea capaz de estar continuamente actualizado ni de indicar qué interacciones se pueden producir de forma altamente fidedigna.

Es entonces cuando surge la necesidad de un sistema que sea capaz de extraer información de todos los textos biomédicos que se puedan tratar, para posteriormente procesar dicha información y obtener las interacciones entre fármacos.  Además, la automatización de esta tarea permitirá tener una base de datos actualizada según se vaya generando nueva información en el dominio biomédico.

## Cómo pueden ayudar los sistemas de IE

Al existir tanta documentación de fármacos, ya sea en bases de datos o en textos biomédicos desestructurados, resulta bastante impracticable que un profesional pueda buscar información fiable y actualizada en un tiempo razonable, ya que tendría que obtener información relevante de bases de datos, o tendría que leerse una gran multitud de artículos para encontrar la información que busca.

Un sistema de Extracción de Información (IE en adelante) puede procesar tales documentos y extraer los datos relevantes para identificar una interacción entre dos fármacos, como por ejemplo los nombres de los fármacos, el verbo que representa la acción entre ambos, y el efecto que se produce. Por lo tanto podremos automatizar el proceso de extraer datos relevantes de textos no estructurados a través de relaciones entre fármacos, y aplicar el mecanismo de IE a todos los documentos médicos que vaya surgiendo.

Con los datos generados se pueden realizar varias acciones, tales como crear una base de datos de dominio biomédico que pueda ser consultable por profesionales, o utilizarlos como corpus que sirva para predecir futuras interacciones por medio de técnicas de aprendizaje automático.

### Extracción de relaciones

La extracción de relaciones consiste en detectar relaciones semánticas entre entidades en un texto. La relación puede involucrar dos o más entidades, aunque la mayoría de los enfoques sobre la extracción de relaciones se ha centrado en la extracción de relaciones binarias. En nuestro caso, el dominio biomédico, la extracción de relaciones se va a utilizar para detectar interacciones entre dos o más fármacos en oraciones en textos. Por ejemplo, en la Ilustración 1 se muestra una relación entre los fármacos *interferón* y *teofilina*:

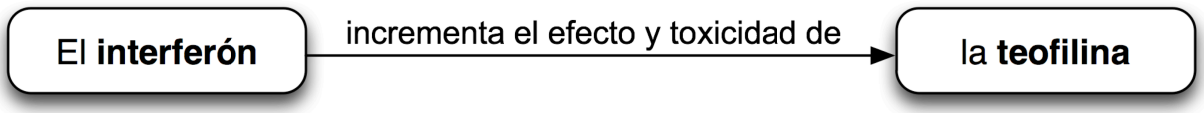


*Ilustración 1: Ejemplo de relación*

Para poder extraer relaciones, es necesario aplicar diferentes técnicas de análisis del lenguaje, ya sean la extracción de raíces, la derivación, el análisis sintáctico y semántico, la desambiguación y la anáfora. Sarawagi [2] presentó los recursos más útiles para extraer relaciones entre entidades. A continuación se resumen algunos de ellos:

- **Información del contexto**: son los elementos que rodean a las entidades. Es el contexto de la frase.
- ***PoS tagging***: es el proceso de clasificar gramaticalmente palabras dentro de su oración y etiquetarlas. Mediante esta técnica podemos identificar las entidades (nombres) y sus relaciones (verbos).
- **Árboles de análisis sintáctico completo**: agrupa las palabras en sus categorías sintácticas, tales como nombre, preposición o verbo. Gracias a esta categorización se pueden detectar mejor las relaciones entre entidades, pero resultar costoso crearlos, tanto computacional como temporalmente.
- **Grafos de dependencia**: representa las dependencias gramaticales de unas palabras con otras dentro de una oración. Cada grafo enlaza una palabra con las que dependen de ésta.

La extracción de relaciones en el campo de la biomedicina consiste en detectar un tipo de relación determinado entre dos entidades dadas. Este tipo de relaciones van desde lo general, tal como asociaciones bioquímicas, hasta lo específico, tal como interacciones entre proteínas o entre fármacos. Normalmente los resultados de este proceso se almacenan en bases de datos para su posterior procesamiento por usuarios o para que sean explorados por algoritmos de minería de datos. Aunque se lleva bastante tiempo investigando en esta línea, hoy en día no se pueden hacer comparaciones ni extraer conclusiones claras de los resultados obtenidos en las diferentes investigaciones, ya que cada grupo ha utilizado distintos conjuntos de datos y los tipos de relaciones son diferentes. Por lo tanto los resultados dependen del tipo de relación a extraer y el conjunto de datos o corpus a procesar.

A continuación se presentan las tres categorías que agrupan los principales enfoques para extraer relaciones entre entidades en el dominio biomédico:

- **Basados en lingüística**: utiliza diferentes técnicas lingüísticas para detectar las relaciones, tales como realizar un análisis sintáctico y semántico. En función de la complejidad de estas técnicas se pueden clasificar en dos tipos: análisis superficial (*shallow parsing*) y análisis profundo (*deep parsing*).
- **Basados en patrones**: se buscan patrones específicos en el texto que conectan entidades y las palabras que intervienen.
- **Basados en aprendizaje automático**: mediante esta técnica automáticamente se adquiere el conocimiento necesario para poder detectar las relaciones.
    - *Métodos basados en características*: se definen una serie de atributos relevantes y cada instancia se representa como un vector con los valores de esos atributos. A

partir de estos vectores se entrena un algoritmo de aprendizaje automático, el cual genera un modelo de aprendizaje. A partir de ese modelo se clasificarán nuevas instancias.

- *Métodos basados en funciones kernel*: las instancias se pueden representar de cualquier forma estructural, tal como grafos de dependencia o árboles de análisis sintáctico. Estos métodos están basados en la definición de una función kernel que recibe como entrada cualquiera de esas estructuras y es capaz de medir la similitud entre dos instancias. Por lo tanto las clasifica en función de su semejanza.

Aun así lo más frecuente es que los sistemas combinen estas aproximaciones, pues las técnicas lingüísticas se suelen utilizar tanto en la creación de patrones como en el aprendizaje automático. De esta manera se pueden eliminar las desventajas de algunos métodos y aprovechar las ventajas de otros.

## Extracción de relaciones basada en aprendizaje automático

Los métodos basados en aprendizaje automático han recibido gran interés en la comunidad biomédica en los últimos años debido a los buenos resultados obtenidos en el ámbito general. Estos métodos representan el problema de extracción de relaciones como una tarea de clasificación y utilizan algoritmos de aprendizaje para extraer información relevante de los textos automáticamente. Sin embargo la ejecución de estos algoritmos requiere un alto coste computacional y sus resultados dependen de la disponibilidad de grandes conjuntos de datos anotados. Los ejemplos que van a entrenar a los algoritmos deben representarse con un formato apropiado para entrenar de la forma más adecuada a un algoritmo.

Según el tipo de representación, podemos distinguir dos categorías: basados en características y basados en métodos kernel:

### Métodos basados en características

Estos métodos extraen un conjuntos de atributos de la entrada y representan cada instancia como un vector de características o atributos. Estos vectores son los que se utilizarán para entrenar al algoritmo.

Los atributos se extraen de las oraciones por medio de técnicas de análisis de textos, tales como división de la oración en palabras (*tokens*), etiquetado PoS, análisis sintáctico superficial y profundo, o recocimiento de entidades. En [2] se estudia cómo se pueden extraer características a través de diferentes niveles de procesamiento lingüístico. Las caracterísiticas pueden clasificarse en dos categorías:

- Propiedades de un token (tipo de entidad, etiquedato POS, ….)

- Relaciones entre tokens (secuencia, sintáctica o relaciones de dependencia entre tokens)

Algunos de los atributos más usados serían los siguientes:

- Palabras entre entidades (incluyendo o no a las entidades)

El **interferón** *incrementa el efecto y toxicidad de la* **teofilina**

- Tipos de entidades

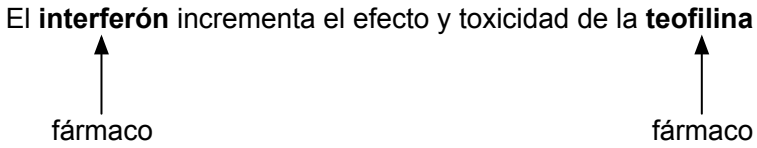


*Ilustración 2: Tipos de entidades en una oración*

- Número de palabras entre entidades (incluyendo o no a las entidades)

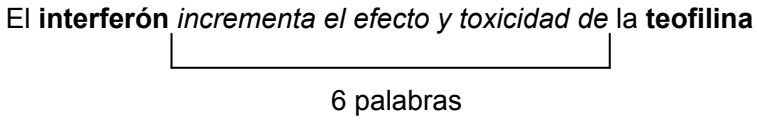


*Ilustración 3: Número de palabras entre entidades*

- Árbol sintáctico de una relación

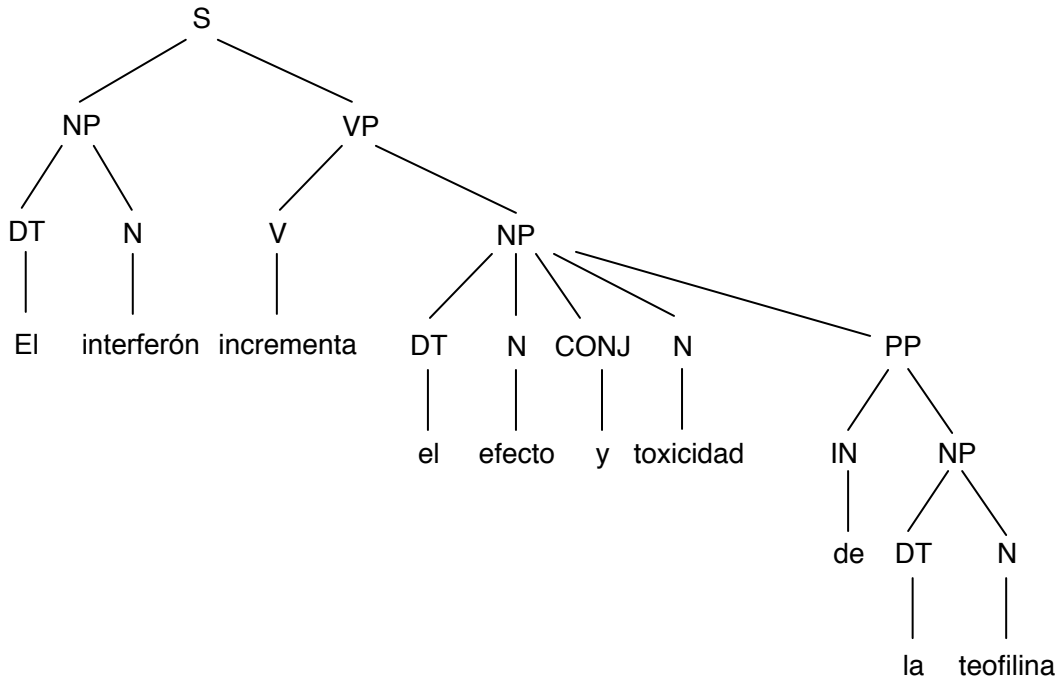


*Ilustración 4: Árbol sintáctico*

- Grafo de dependencia de la relación, o camino sintáctico entre ambas entidades.

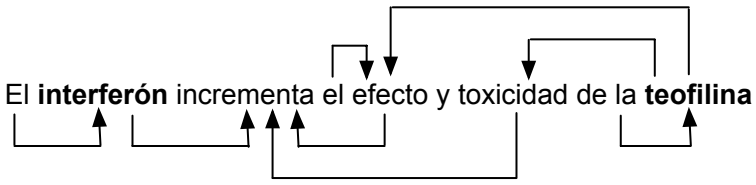


*Ilustración 5: Grafo de dependencia de la relación*

A pesar de que los buenos resultados que han obtenido los métodos basados en características en la extracción de relaciones, hay algunas deficiencias que se deben tener en cuenta:

- La selección del conjunto de características más apropiado requiere un conocimiento profundo del dominio. Además es una tarea que requiere mucho tiempo.
- Las características no son capaces de capturar correctamente la información contenida en estructuras complejas tales como grafos de dependencia o árboles.
- En algunos casos, el espacio de características puede alcanzar una dimensión demasiado elevada. Esto unido a que la gran variabilidad del lenguaje puede provocar que los valores de estas caracterítsicas tengan escasa densidad, dando lugar a que la computación de las características sea inviable computacionalmentete hablando.

### Métodos basados en funciones Kernel

Estos métodos son una buena alternativa a los enfoques basados en características en cuanto a la extracción de relaciones. Su principal ventaja es que se pueden aplicar a cualquier tipo de datos, independientemente de como sea su representación original. Los métodos Kernel toman los datos de entrada (cualquier representación), los transforma a un espacio lineal y compara dichas transformaciones. Esto conlleva la ventaja que se pueden aplicar algoritmos lineales y obtener resultados para datos de entrada no lineales.

En vez de representar cada instancia en un vector de características, los métodos Kernel definen una medida de semejanza que determina la similitud entre instancias. Básicamente un método kernel actúa como una interfaz entre los datos de entrada y un algoritmo de aprendizaje. Los objetos de entrada se transforman para ser representados en un espacio de vectores, en el que el proceso de aprendizaje actúa midiendo la similitud entre objetos. Algunos algoritmos de aprendizaje automático pueden formularse como algoritmos basados en métodos kernel. Los más

populares son máquina de vectores de soporte, los k vecinos más cercanos y perceptrones (redes de neuronas).

Dentro de la extracción de relaciones en el dominio biomédico, el rendimiento de los métodos kernel depende de la selección y diseño de las funciones kernel, que se basan en la representación de las relaciones de las instancias. Las instancias pueden venir representadas en diferentes formas, dependiendo de la cual se podrán obtener diferentes tipos de información contextual. Estas representaciones de instancias son las siguientes:

- Bolsa de palabras: la más simple y usada. Cada relación se representa como un vector donde cada elemento indica la ocurrencia de una determinada palabra en una oración. A pesar de su simpleza en esta representación se pierde información gramatical y el orden de las palabras en la oración.
- Secuencia de palabras: en esta representación sí que se captura el orden secuencial de las palabras. La manera más simple es considerar todas las palabras excepto las dos entidades. También se pueden obtener subsecuencias de una determinada longitud, o sólo incluir en la secuencia algunos tipos determinados de atributos. Bunescu y Mooney propusieron una generalización en los kernel de secuencias en [5]. Ellos comprobaron que los contextos más útiles para determinar si una oración contiene una relación entre dos entidades se pueden representar por los siguientes patrones:
    - *fore-between*: son las palabras situadas antes y entre las dos entidades de la relación.
    - *between*: son las palabras entre las dos entidades. Este contexto contiene la relación.
    - *between-after*: palabras situadas entre y a continuación de las dos entidades involucradas en la relación.
- Árboles sintácticos: en este caso se utiliza la estructura sintáctica de la oración, es decir, su árbol sintáctico. Este tipo de información puede resultar útil para detectar relaciones entre entidades.
- Grafos de dependencia: se utiliza la representación de las dependencias entre las palabras de una misma oración.

## Antecedentes

Antes de exponer los objetivos específicos que se van a llevar a cabo, vamos a ponernos en contexto con los inicios y evolución del trabajo que da lugar a este proyecto.

El sistema descrito en [1] es la primera aproximación desarrollada hasta el momento dedicada a la extracción de interacciones entre fármacos. En dicho trabajo fueron estudidas varias técnicas de extracción de información. Los textos fueron analizados con ayuda de la herramienta MMTx[1] y la ontología UMLS[2]. Para la evaluación de las distintas técnicas, se creó el corpus DrugDDI y se anotó manualmente con interacciones farmacológicas. En particular, una de las aproximaciones desarrolladas estaba basado en el **método kernel** propuesto por Giuliano et al. en [5]. Tras haber evaluado este método kernel sobre el corpus DrugDDI, se obtuvieron los siguientes resultados:

| Experimento | Precisón | Recall | Medida-F |
|---|---|---|---|
| Método **kernel** aplicado sobre el corpus DrugDDI | 0,57 | 0,80 | 0,64 |

*Tabla 1: Resultados obtenidos en la evaluación de métodos kernels en [1]*

Posteriormente, se desarrolló el sistema Druida [4] como resultado del trabajo de un proyecto de fin de carrera dirigido por Isabel Segura-Bedmar y presentado en la UC3M por Víctor Manuel Méndez González. Este sistema realiza las siguientes aportaciones para llevar a cabo la extracción de información:

- **Tecnologías Castor[3] y XSD[4]**: sirven para facilitar la extracción de datos del corpus, ya que mediante Castor el tratamiento de contenidos en formato XML es más eficaz y directo, y mediante los XSD (*Xml Schema Definition*) se pueden generar de forma automática las clases Java para acceder a datos del XML. De esta manera, si hay cambios en la estructura del XML, con sólo modificar el XSD, se vuelven a generar de forma automática los objetos necesarios sin la necesidad de tocar código fuente.
- **Extracción de atributos**: Cada instancia (par de fármacos) se representa como un vector de características formado por los códigos ATC (familias de los fármacos), un valor *boolean* para indicar si la oración contiene un verbo con significado de interacción (*increase*, *decrease*, *interact*, etc), un valor *boolean* para indicar si la oración contiene un verbo modal, y un valor *boolean* para indicar si existe un partícula de negación entre los dos pares de fármacos. El conjunto de instancias se imprime a un fichero en formato CSV.

1 http://metamap.nlm.nih.gov/
2 http://www.nlm.nih.gov/research/umls/
3 http://www.castor.org/
4 http://www.w3.org/TR/2009/WD-xmlschema11-1-20091203/

El sistema Druida [4] está basado en la implementación SMO del algoritmo SVM (*Support Vector Machine*), integrado en la herramienta Weka. Los resultados obtenidos en [4] tras haber evaluado SVM sobre el conjunto de datos fueron los siguientes:

| Experimento | Precisón | Cobertura | Medida-F |
|---|---|---|---|
| SVM | 0,447 | 0,127 | 0,198 |

*Tabla 2: Resultados obtenidos en la evaluación de SVM en [4]*

## Objetivo del Proyecto

Continuando el trabajo realizado en [4], el objetivo del proyecto es mejorar la representación de las instancias (ejemplos) mediante la inclusión de nuevas características que puedan contribuir a mejorar los resultados, así como el estudio de nuevos algoritmos de aprendizaje supervisado para la extracción de interacciones farmacológicas.

Nuestro objetivo final será la comparación de los resultados obtenidos con los reportados por el método kernel presentado en el trabajo [1].

## Objetivos específicos

Como objetivos específicos del proyecto se listan los siguientes:

- **Estudio de trabajo previo**: se va a realizar un amplio estudio sobre el trabajo presentado en [1] para conocer el problema.
- **Extracción de nuevas características**: a partir del corpus DrugDDI, se pretenden extraer más características que puedan enriquecer el conjunto de datos. Entre éstas se encuentran el contexto *before-between-after* [5], representación del contexto de cada fármaco mediante N-gramas, ventanas de diferente tamaño entorno a los fármacos, camino entre los fármacos ó información morfosintáctica. Todas estas características se van a extraer del corpus sin recurrir a recursos externos.
- **Selección de atributos**: aplicación de diferentes algoritmos de selección de atributos con Weka para aplicarlos sobre los datos y así obtener las características más relevantes. De esta manera podemos eliminar posible ruido que puedan introducir algunas características durante el aprendizaje. La selección de atributos nos da las características más discriminatorias. Finalmente, la salida de estos algoritmos de selección de atributos es la entrada de los algoritmos de aprendizaje automático.
- **Experimentos con diferentes algoritmos de aprendizaje supervisado**: se van a aplicar distintos clasificadores para comprobar cuáles dan mejor resultados. Se analizarán los resultados de los distintos algoritmos.
- **Comparación con los resultados del método kernels aplicado en [1]**: se realizará una comparación de los resultados del presente proyecto con los resultados obtenidos en [1]. Se tratará de razonar en qué casos los resultados son mejores.

## Estructura de la memoria

En el capítulo *2. Aprendizaje Automático y los principales clasificadores*, se va a explicar qué es el aprendizaje automático, por qué puede ser útil en el desarrollo de nuestro sistema de detección de DDIs, y los principales clasificadores que se van a usar en el presente proyecto, tales como *Naive Bayes*, *k-nearest neighbor*, *decision trees*, *Neural Networks*, etc. También se hará una breve introducción de la herramienta WEKA, la cual implementa los algoritmos que se aplicarán. Por último se realizará una breve descripción de cómo se evalúan los modelos de clasificación obtenidos, indicando las principales métricas.

A continuación, en el capítulo *3. Análisis de clasificadores aplicados a la extracción de DDI*, se realizará un análisis de los clasificadores que vamos a aplicar para predecir DDIs. Para ello en primer lugar se va a describir la arquitectura del sistema y el corpus a partir del cual se va a extraer la información de los fármacos y sus interacciones. A continuación, se describe el conjunto de características que se utilizará para entrenar los clasificadores. Por últimos se presentan los diferentes experimentos realizados y una comparación con los resultados del método kernel aplicado en [1] y con los resultados del sisema Druida [4].

En el capítulo *4. Conclusiones y líneas de trabajo futuro* se realizará un breve resúmen del proyecto y se expondrán los resultados, comprobando si se han cumplido los objetivos propuestos o no. También se detallará la línea futura del proyecto y posibles mejoras y/o evoluciones.

# Aprendizaje automático y los principales clasificadores

## Qué es el aprendizaje automático

El Aprendizaje Automático es una rama de la Inteligencia Artificial que se encarga del diseño y desarrollo de algoritmos que permiten a una computadora mejorar un comportamiento automáticamente a través de la experiencia. Es por ello que se le da el nombre de Aprendizaje Automático o Aprendizaje de Máquinas, ya que una máquina es capaz de aprender por sí sola por medio de datos empíricos. De forma más concreta, un algoritmo de AA es capaz de extraer características y patrones comunes de un conjunto de datos (ejemplos o datos de entrenamiento), y aplicarlas a nuevos conjuntos de datos (datos de pruebas).

Esta disciplina se utiliza en diferentes campos, tales como diagnósticos médicos, análisis de mercados, robótica, banca, reconocimiento del habla y lenguaje escrito, detección de patrones en imágenes, clasificación de secuencias de ADN, o marketing.

Existen diferentes tipos de algoritmos clasificados en base a la salida de los mismos. La distinción más significativa sería entre el aprendizaje supervisado y el aprendizaje no supervisado. Existen más tipos de algoritmos, tales como el aprendizaje semi-supervisado, por refuerzo, ..., pero para el alcance de este proyecto no son relevantes.

### Aprendizaje supervisado

En este tipo de aprendizaje se deduce una función a partir de un conjunto de datos de entrenamiento que ya están etiquetados correctamente, de ahí que reciba el nombre de supervisado. En estos algoritmos se proporciona un conjunto de datos de entrenamiento en forma de un vector. El algoritmo, tras analizar los vectores de entrada infiere una función conocida como clasificador o función de regresión dependiendo de si la salida es discreta o continua. La función inferida debe ser capaz de predecir la clase para cualquier vector de entrada. A continuación se muestra su funcionamiento en mayor detalle en la siguiente figura:

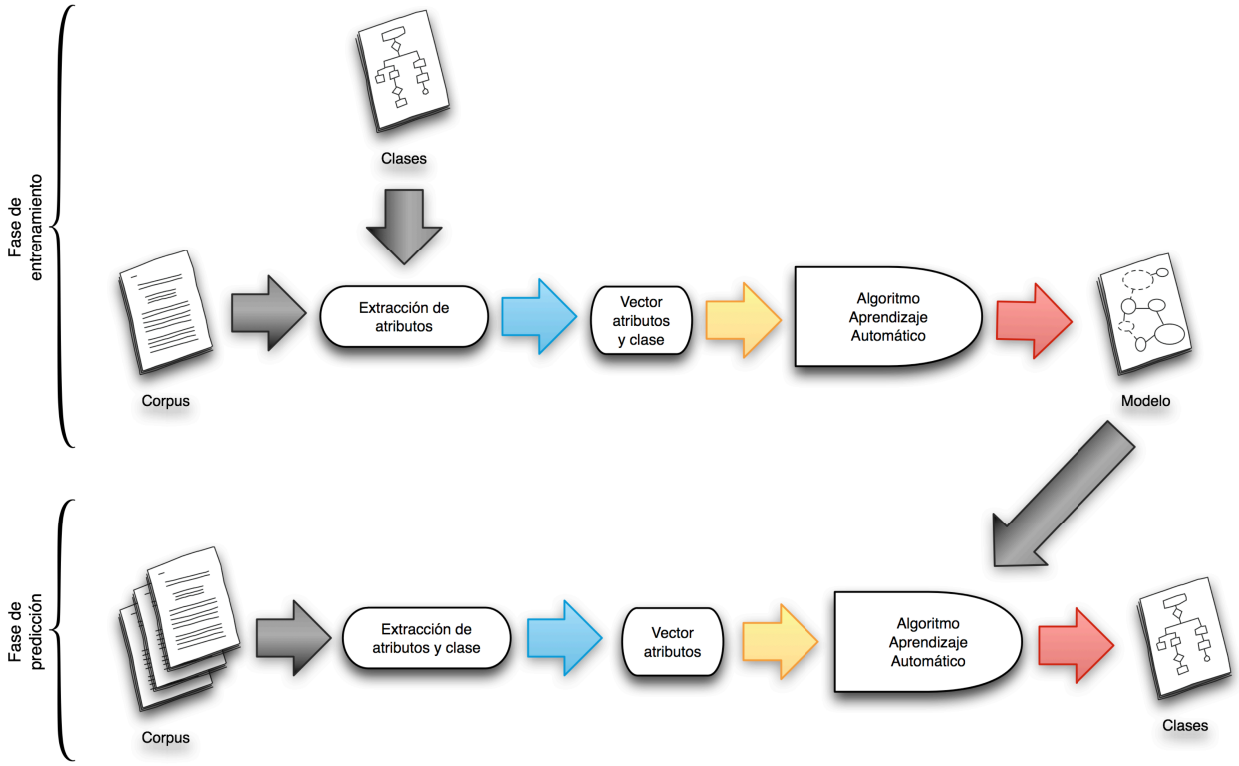


*Ilustración 6: Aprendizaje supervisado*

En primer se lleva a cabo la fase de entrenamiento. En esta fase a partir de un corpus se extraen los atributos relevantes para el clasificador. Estos atributos se habrán clasificado previamente, por lo que a priori ya se conoce su clase. De la extracción de atributos sale un conjunto de vectores que formarán la entrada al algoritmo, el cual a partir de los datos y sus clases generará un modelo que generalice las características extraídas del corpus. En la siguiente fase, la de predicción, se realizará el mismo proceso de extraer atributos para generar un vector, pero ya no conoceremos la clase de estos ejemplos. El algoritmo, como ya habrá sido capaz de *aprender* un modelo, podrá aplicarlo a los datos y de esta manera predecir las clases de estos ejemplos.

### Aprendizaje no supervisado

En este tipo de aprendizaje se intenta determinar la estructura de los datos, es decir, se intentan agrupar grandes cantidades de datos en función de las características que comparten. Recibe ese nombre debido a que no se conoce nada a priori. Es una manera de encontrar jerarquía y orden en un conjunto de datos sin estructura. Estas agrupaciones se pueden ver como un conjunto de elementos similares en algunos sentidos, pero diferentes de los elementos que pertenecen a otros grupos.

Uno de los principales métodos del aprendizaje no supervisado es el *clustering*. Conceptualmente se suele representar viendo qué elementos forman grupos en un eje (x, y), e introduciéndolos en un círculo, tal como se muestra en la siguiente ilustración:

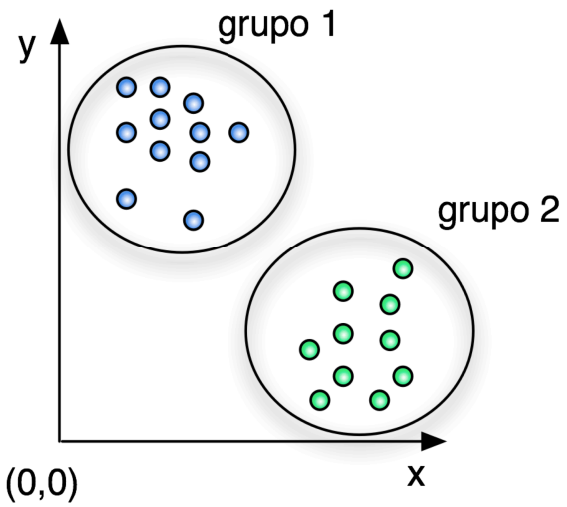


*Ilustración 7: Aprendizaje no supervisado*

En la Ilustración 7 se han encontrado dos grupos según la cercanía de sus elementos en el espacio de un plano (x, y).

Lo más importante en el *clustering* es encontrar una función que sea capaz de cuantificar las similitudes entre dos datos como un número, como por ejemplo, la distancia Euclídea entre dos puntos.

## Principales esquemas

A continuación se van a describir brevemente los principales esquemas de aprendizaje automático. Si bien pueden existir múltiples clasificaciones de estos esquemas, aquí se va a exponer la que se considera más general.

### Árboles de decisión

El aprendizaje por medio de los árboles de decisión está basado en el principio *divide y vencerás*. Estos algoritmos generan árboles de decisión, que son modelos en forma de grafo que se construyen recursivamente seleccionando en primer lugar un atributo que será el nodo, y creando una rama para cada posible valor de dicho nodo. Por cada valor, se vuelve a establecer otro atributo y sus ramas con sus valores, hasta que se llega a los nodos finales, que se corresponden con las clases.

Los árboles de decisión tienen buen rendimiento con grandes volúmenes de datos, ya que no requiere cargar todos los datos en memoria a la vez. El tiempo de cálculo escala bien con un número de columnas creciente linearmente.

Las ventajas de los árboles de decisón es que es sencillo de entender y de interpretar, la generación de reglas es simple, reduce la complejidad del problema, y el tiempo de entrenamiento no es muy largo.

Algunas de las desventajas son que si se comete un error en un nivel alto, los nodos sucesivos estarían mal creados. En la construcción de un árbol de decisión, lo más complicado es determinar en qué atributo basar un nodo ya que si que existen muchas características, el algoritmo tendría muchas opciones de los datos de entrenamiento, y se construiría un modelo que no generalizaría bien con nuevos ejemplos. Este problema se conoce como *overfitting*, y hace que en múltiples ocasiones no de buenos resultados con muchas características en los datos de entrenamiento.

Sin embargo, los árboles de decisión pueden dar buenos resultados si se combinan con métodos Ensemble. Con estos métodos en vez de aprender un único modelo, se aprenden varios, y se combinan las estimaciones de cada modelo.

### Redes bayesianas

Las redes bayesianas son un modelo probabilístico que representan un conjunto de variables y sus dependencias condicionales mediante un grafo acíclico dirigido, que es un grafo dirigido que no tiene ciclos, es decir, para cada vértice no hay un camino directo que empiece y termine en el

mismo vértice. En estos grafos los nodos representan los atributos y los arcos dependencias condicionales entre los atributos. Cada nodo tiene asociada un función de probabilidad que se utiliza para predecir la probabilidad de la clase para cada instancia. Para ello en cada nodo se toman como entrada los valores los nodos padre, y por cada valor de éstos se representa la correspondiente probabilidad para cada valor del atributo representado por el nodo. Dado este modelo, para clasificar una nueva instancia habría que hacer inferencia bayesiana: estimar la probabilidad de las nuevas instancias en base a las variables de las instancias conocidas.

Para construir un algoritmo de aprendizaje para Redes Bayesianas es necesario definir dos componentes: una función para evaluar una red dada basada en los datos y un método para buscar a través del espacio de redes posibles.

Las redes bayesianas se utilizan para modelar conocimiento en diferentes áreas, tales como en bioinformática, clasificación de documentos, recuperación de información, procesamiento de imágenes, sistemas de toma de decisiones, juegos o marketing.

### Reglas de asociación

La asociación trata de descubrir la probabilidad de la existencia de correlaciones entre elementos en un conjunto de datos. Las relaciones entre estos elementos se representan como reglas de asociación. Este tipo de aprendizaje se aplica en diferentes áreas, tales como ventas, minería del uso de la Web, detección de intrusión y bioinformática.

El típico ejemplo que se da para explicar las reglas de asociación es el problema *market-basket*. Supongamos que un mercado guarda las transacciones de todas las compras. Cada transacción es un subconjunto de todos los elementos disponibles en el mercado.

El problema es: analizando muchas transacciones, ¿podríamos descubrir la correlación entre subconjuntos? Por ejemplo, la gente que compra cereales tiene una alta tendencia a comprar leche. Esta correlación sería la regla de asociación, y se interpretaría de manera que partiendo de las estadísticas de las transacciones, la gente que compra todos los elementos del conjunto A tiende a comprar los elementos del conjunto B.

Las reglas se representan en la forma IF-THEN, donde la parte del IF es el antecedente y la parte THEN es la consecuencia. Ambas partes son disjuntas, no tienen ningún elemento en común.

Para seleccionar las reglas más relevantes, se deben tener en cuenta ciertas métricas. Las principales métricas de las reglas de asociación son el soporte y la confianza:

- Soporte: es el porcentaje de transacciones en el conjunto total de datos que contiene una regla determinada.

- Confianza: es la probabilidad de que unos antecedentes y consecuentes aparezcan en la misma transacción.

Para que las reglas de asociación sean interesantes o útiles, deben tener un soporte y confianza mínimos, para ello el proceso de generación de reglas se compone de dos partes: en primer lugar se aplica el soporte mínimo para encontrar todos los subconjuntos frecuentes en la base de datos. Esta tarea es compleja e implica considerar todas las posibles combinaciones de elementos. La segunda parte consiste en formar las reglas de esos conjuntos frecuentes de items y de la restricción de confianza mínima.

### Modelos lineares

La forma más natural de trabajar de los métodos que hemos visto, tales como árboles de decisión y reglas, es con atributos nominales. Éstos se podrían extender a atributos numéricos, pero hay métodos que trabajan mejor con este tipo de atributos. A éstos se los conoce como modelos lineares. Estos métodos asumen que la clase puede expresarse como una relación lineal algebraica con unos atributos de entrada, que sería los datos de entrenamiento. En este tipo de modelos existen diferentes métodos, a continuación de da una breve descripción de los principales:

- **Regresión lineal**: aquí los atributos y la clase son numéricos. En estadísticas, la regresión lineal consiste en modelizar una relación entre una variable dependiente (la clase) y un conjunto de variables independientes (los atributos). El aprendizaje en este tipo de método va a consistir en determinar un vector de pesos - uno por cada atributo - de tal manera que la diferencia entre la salida predecida y la actual sea mínima. Una vez que tenemos el vector numérico de pesos, los podemos utilizar para predecir la clase de nuevas instancias. La regresión lineal siempre ha sido un buen método para hacer predicción numérica, pero tiene la desventaja de que los datos no tengan una dependencia lineal.
- **Regresión logística**: se utiliza cuando la salida es binaria en vez de un número real. La primera parte es la misma que en regresión lineal, pero aparte se aplica una función para cada clase, estableciendo una salida igual a 1 para las instancias del entrenamiento que pertenecen a la clase, y asignando 0 a las que no. El resultado del modelo es una expresión lineal para cada clase. Luego, dada una instancia de test de clase desconocida, se calcula el valor de cada expresión lineal para cada clase y se elige la mayor.
- **Redes neuronales**: están inspiradas en cómo trabaja nuestro cerebro. Una red de neuronas consiste en un conjunto de neuronas artificiales interconectadas entre sí. Cada neurona se considera una unidad de procesamiento, recibe una serie de entradas a través de sus interconexiones y genera una salida. Las conexiones cuentan con un peso

determinado. Básicamente, el cómputo de una neurona involucra el cálculo de una función de propagación, que opera con las entradas y sus pesos asociados, una función de activación, que modifica a la anterior, y una función de transferencia, que se aplica al valor devuelto por la función de activación. Habitualmente el aprendizaje se encarga de adaptar los pesos de las interconexiones; en el caso de aprendizaje supervisado se dará la salida deseada para unas entradas determinadas, y en el caso de aprendizaje no supervisado, la red tratará de detectar características comunes en los datos. Las redes de neuronas se utilizan ampliamente en el aprendizaje automático. Cuentan con las ventajas de que son robustas al ruido en los datos y sus cómputos se pueden paralelizar fácilmente, lo que da lugar a una mejora del rendimiento. Se suelen utilizar en problemas de clasificación, en reconocimiento de patrones y de secuencias, robótica, diagnosis métdicos, visualización o minería de datos.

### Support Vector Machine

SVM es una técnica de clasificación muy potente. Su teoría está basada en un modelo lineal, pero también puede controlar modelos no lineales. Además no suele tener problemas con la conocida *maldición de dimensionalidad (curse of dimensionality)*, que ocurre cuando hay muchos atributos con respecto al número de instancias.

En SVM las entradas son numéricas y las salidas binarias.

Este algoritmo construye un hiperplano o conjunto de hiperplanos en un espacio de dimensionalidad muy alta, la máxima posible. En este espacio existirán una serie de puntos de muestra, separando las clases. Cuanto más separadas estén las clases mejor se realizará la clasificación. La separación se basa en los vectores de soporte, que son las instancias de cada clase que generan una función discriminatoria lineal que los separa tanto como sea posible.

Cuando se van a clasificar nuevas instancias de clases desconocidas, en función de su proximidad en el espacio pertenecerán a una clase u otra.

Sus principales aplicaciones son en el análisis de datos, reconocimiento de patrones y categorización de textos.

### Aprendizaje basado en instancias o por vecindad

En este tipo de aprendizaje no se crea una inducción previa del modelo como en otros algoritmos, si no que los ejemplos del entrenamiento se almacenan en memoria, y para clasificar se utiliza una función de distancia para determinar qué datos del conjunto de entrenamiento están más próximos a una nueva instancia de test. Una vez que se ha localizado la instancia de entrenamiento más cercana, se predice la clase para la instancia de test en base a la clase de dicha instancia. El principal problema es definir la función de distancia ya que debe capturar una relación de similitud entre las instancias de entrenamiento y test. Ésta suele ser específica del

dominio.

Al aprendizaje basado en instancias se le conoce como aprendizaje perezoso, ya que todo el cómputo se realiza durante la clasificación de una nueva instancia, no cuando se procesa el conjunto de entrenamiento.

Este tipo de algoritmo puede tener un coste computacional bastante considerable al clasificar una nueva observación debido al proceso de consulta, sobre todo si hay muchos datos de entrenamiento. Además los atributos irrelevantes pueden llevar a error fácilmente.

Las principales técnicas del aprendizaje basado en instancias son el *nearest neighbor (vecino más próximo)* que funciona tal como se ha descrito previamente, y los *k-nearest neighbor* (K-NN), que tiene en cuenta las K instancias más próximas. En este caso escoger el valor de K es una dificultad añadida, ya que un valor bastante elevado puede reducir el ruido en los datos pero hace más difícil los límites entre clases.

Una aplicación bastante popular hoy en día de este tipo de aprendizaje es en los recomendadores utilizando *Filtro colaborativo*.

## Herramientas: Weka

En este proyecto se va a utilizar la herramienta Weka[1] para aplicar las técnicas de aprendizaje automático a los vectores de características que se generen.

Weka (*Waikato Environment for Knowledge Analysis*) es un programa escrito en Java que implementa algoritmos de aprendizaje automático y herramientas de procesamiento y filtrado para hacer minería de datos, y que se ha desarrollado en la Universidad de Waikato, en Nueva Zelanda. Weka es *software* libre distribuido bajo licencia GNU-GPL, por lo que es de libre distribución, modificación y uso.

La versión original de Weka se remonta al año 1993, comenzó desarrollándose en TCL/TK y C, y estaba destinada al análisis de datos del dominio de la agricultura. Pero a partir de que se reescribiese todo el código en Java en 1997, Weka se ha estado utilizando en diferentes ámbitos. Weka está en continuo desarrollo, desde sus inicios sus desarrolladores han estado evolucionando contínuamente el programa introduciendo más algoritmos, utilidades de análisis y mejoras.

Las tareas que soporta son preprocesamiento de datos, *clustering*, clasificación, regresión, visualización y selección de atributos. Todas estas tareas se pueden ejecutar a partir de una interfaz gráfica de una manera intuitiva, esto hace que la curva de aprendizaje de Weka sea menor que con otros programas de minería de datos. También tiene una interfaz con línea de comandos, que aunque puede resultar más compleja, es más potente ya que proporciona acceso a todas las funcionalidades de Weka. Aparte, Weka está preparado para que se pueda extender con algoritmos desarrollados por terceras partes.

Weka se puede utilizar para diferentes propósitos, tales como aplicar un algoritmo de aprendizaje a un conjunto de datos y analizar la salida para conocer mejor los datos, o generar predicciones sobre nuevas instancias a través de un método de aprendizaje, o aplicar diferentes algoritmos para comparar su rendimiento y ver cuál predice mejor.

1 http://www.cs.waikato.ac.nz/ml/weka/

A continuación se detallan los módulos que componen Weka:

**Explorer**

Está compuesto por un grupo de paneles que permite acceder a las distintas tareas de minería de datos que soporta Weka. Es el componente que más se suele utilizar. Se compone de las siguientes partes:

- *Preprocess*: este panel permite definir el origen de los datos, que puede ser importando ficheros en diferentes formatos, como CSV, C4.5, con instancias serializadas, arff o xrff, mediante consultas a una base de datos SQL, o a partir de una URL. Una vez que se han obtenidos los datos se pueden preprocesar aplicando uno o varios de los algoritmos de filtrado que ofrece Weka.
- *Classify*: mediante este panel se pueden aplicar algoritmos de clasificación y regresión sobre los datos, para poder generar un modelo predictivo, estimar la precisión del modelo generado y así visualizar errores en las predicciones. Una vez seleccionado un algoritmo, se puede elegir el modo de entrenamiento especificando si se quieren usar todos los datos para entrenar, otros datos, un porcentaje, o hacer *cross-validation*. En el mismo panel se inicia la ejecución del algoritmo, y una vez finalizado muestra la información referente a su desarrollo, tal como la matriz de confusión, los errores de clasificación, gráficas, el modelo generado, o información sobre el *dataset*. También proporciona meta-clasificadores, los cuales son combinaciones de clasificadores.
- *Cluster*: en esta parte se pueden utilizar las técnias de *clustering* de Weka. Al igual que en el panel de clasificación, se puede aplicar un algoritmo de *clustering* y personalizarlo. También permite mostrar de forma gráfica la asignación de las muestras en *clusters*.
- *Associate*: este panel permite utilizar las reglas de asociación aprendidas para identificar asociaciones entre datos.
- *Select attributes*: mediante este panel se pueden utilizar algoritmos para identificar los atributos más relevantes en los datos, es decir, los atributos que tienen más peso a la hora de clasificar.
- *Visualize*: en este modo se muestra gráficamente la distribución de todos los atributos. De esta manera se pueden ver asociaciones entre los atributos.

**Experimenter**

En este módulo se pueden aplicar varios algoritmos de clasificación sobre un conjunto de datos, y así poder analizar los resultados y determinar cuál de los esquemas es mejor estadísticamente. Resulta útil para realizar comparaciones de rendimiento entre varios esquemas de aprendizaje automático. Permite configurar cada experimento, especificando los datos

involucrandos, los algoritmos a usar, el tipo de evaluación, ... También se puede realizar la ejecución en modo distribuido entre varios nodos mediante RMI.

**KnowledgeFlow**

Este módulo es una alternativa al Explorer que muestra de una forma más explícita el funcionamiento interno de una ejecución. Permite crear flujos de datos partiendo de componentes WEKA que se enlazan y combinan formando un flujo de conocimiento para procesar y analizar datos. En el módulo *KnowledgeFlow* se proporcionan las siguientes funcionalidades:

- Procesamiento de los datos en *batch* o incrementalmente (según los algoritmos que lo permitan).
- Encadenar filtros de preprocesamiento de datos.
- Procesar flujos de datos en paralelo.
- Visualizar los modelos intermedios generados por los clasificadores para cada *fold* en la evaluación *cross-validation*
- Visualización del *performance* de los clasificadores incrementales durante el procesamiento.

**Simple CLI**

El módulo Simple CLI (*Simple Command-Line Interface*) es una interfaz de línea de comandos a través de la cual se pueden ejecutar todas las operaciones que soporta Weka, pudiendo invocar a todas sus clases, ya sean clasificadores, filtros, *clusters*, ... Por ello es bastante potente pero también es más complicada de usar ya que mientras que en la interfaz gráfica es muy intuitivo realizar cualquier operación, mediante línea de comandos se requiere un conocimiento más profundo de toda la herramienta.

## Evaluación en Extracción de Información

Para decidir si un modelo de clasificación es preciso capturando un patrón debemos evaluarlo. El resultado de esta evaluación es importante para decidir si un modelo es fiable o no, y para qué propositos lo podemos usar. La evaluación también puede servir para guiarnos en hacer futuras mejoras al modelo.

En este apartado se van a describir brevemente las principales técnicas de evaluación y las medidas que se van a utilizar posteriormente en nuestros experimentos.

### *Principales métodos de evaluación*

#### Conjunto de datos de entrenamiento y evaluación

La mayoría de las técnicas de evaluación calculan un resultado comparando las clases que predice para las entradas en un conjunto de evaluación, con las clases correctas para dichas entradas. Este conjunto de evaluación normalmente tiene el mismo formato que el conjunto de entrenamiento, y ambos deben ser representativos del problema que se está tratando.

Cuando se construye el conjunto de evaluación, hay que tener en cuenta la cantidad de datos disponibles para pruebas y para entrenamiento, ya que si la tarea de clasificación genera muchas clases diferentes o clases poco frequentes, el tamaño del conjunto de pruebas debería ser elegido para asegurar que las clases menos frequentes ocurran.

También hay que tener en cuenta el grado de similitud entre ambos conjuntos, ya que cuanto más parecidos sean, menos fiable será la capacidad de clasificación de nuestro algoritmo.

Unos de los principales problemas de esta técnica es conseguir una adecuada proporción entre los datos de entrenamiento y evaluación, ya que si el conjunto de evaluación es demasiado pequeño, no nos aseguramos de que nuestro clasificador sea muy preciso. Por otro lado, que el conjunto de evaluación sea muy grande implica que el conjunto de entrenamiento sea pequeño y en consecuencia poco representativo.

Mediante la siguiente técnica, nos aseguramos de que ambos conjuntos de datos siempre sean lo suficientemente representativos.

### Validación cruzada

En esta técnica, conocida como *cross-validation*, se subdivide el corpus original en k subconjuntos llamados *folds*. Para cada uno de estos subconjuntos, entrenamos un modelo utilizando todos los datos excepto los datos pertenecientes a dicho subconjunto, y a continuación se prueba el modelo con los datos del subconjunto en cuestión. La elección del valor k dependerá del tamaño y características del conjunto de datos.

En este proyecto utilizamos esta técnica para evaluar nuestros modelos. El número de subconjuntos que generamos va a ser de 10, que además es el valor más común en este método debido a que se ha comprobado a través de muchas pruebas con diferentes técnicas de aprendizaje, que dividiento el corpus en 10 subconjuntos se obtiene la mejor estimación de error.

#### Validación cruzada leave-one-out

Es un caso particular de validación cruzada en el que el número de folds es igual al número de instancias en el corpus. La principal ventaja de esta técnica es que los datos se utilizan de manera óptima, pues la división en conjuntos no se hace al azar. Sin embargo esto conlleva un alto coste de cálculo computacional.

#### Bootstrap

La validación cruzada utiliza muestras sin reemplazo, es decir, una vez que se elige una instancia ésta no se volverá a escoger. Sin embarbo, en la técnica *Bootstrap* se escogen muestras con reemplazo para generar un conjunto de entrenamiento. En concreto, dado un conjunto con N instancias se hacen N selecciones aleatorias con reemplazo para generar el conjunto de entrenamiento. La probabilidad de que una instancia termine quedando en el conjunto de evaluación es de 36.8%, lo cual viene a decir que un 63.2% de todas las instancias formarán parte del conjunto de entrenamiento. Es por esto que a esta técnica se le llama 0.632 bootstrap.

Este tipo de técnica da muy buenos resultados cuando hay pocos datos de entrenamiento.

## *Métricas de evaluación*

#### Exactitud

Es la métrica más simple que se puede utilizar para evaluar un clasificador. La exactitud mide el porcentaje de entradas en el conjunto de evaluación que el clasificador clasifica correctamente. Cuando se analiza la exactitud de un clasificador, hay que tener en cuenta las frecuencias de cada clase individual en el conjunto de test.

**Matriz de confusión**

Una matriz de confusión es una tabla en la que cada columna representa el número de predicciones de cada clase, mientras que cada fila representa a las instancias en la clase real. En estas matrices, cada elemento muestra el número de ejemplos para los que la clase actual es la fila, y la clase predicha es la columna.

**Precision, recall y F-score**

Para definir estas medidas, vamos a considerar las siguientes agrupaciones de elementos en las que se puede clasificar una instancia:

- Verdaderos positivos (VP): son identificados correctamente como positivos.
- Verdaderos negativos (VN): son identificados correctamente como negativos.
- Falsos positivos (FP): son incorrectamente identificados como positivos.
- Falsos negativos (FN): son incorrectamente identificados como negativos.

Dados estos cuatro valores, podemos definir las métricas:

- **Precision**: es la probabilidad de que si una instancia *i* es clasificada por la clase *c*, la instancia realmente pertenece a esa clase, es decir, indica cuántos de los elementos que hemos identificado son positivos. La fórmula es:

$$TP/(TP+FP)$$

- **Recall**: es la probabilidad de que si una instancia *i* es clasificada por la clase *c*, el clasificador la clasifica correctamente, es decir, indica cuántos de los elementos positivos hemos identificado. La fórmula es:

$$TP/(TP+FN)$$

- **medida-F** (*F-score*): combina la precisión y recall para dar un único resultado. La fórmula es:

$$(2*Precision*recall)/(precision+recall)$$

# Análisis de clasificadores aplicados a la extracción de DDI

## Arquitectura del sistema

El desarrollo del sistema de extracción de información se ha realizado como continuación del sistema Druida. En la siguiente imágen podemos ver sus principales componentes:

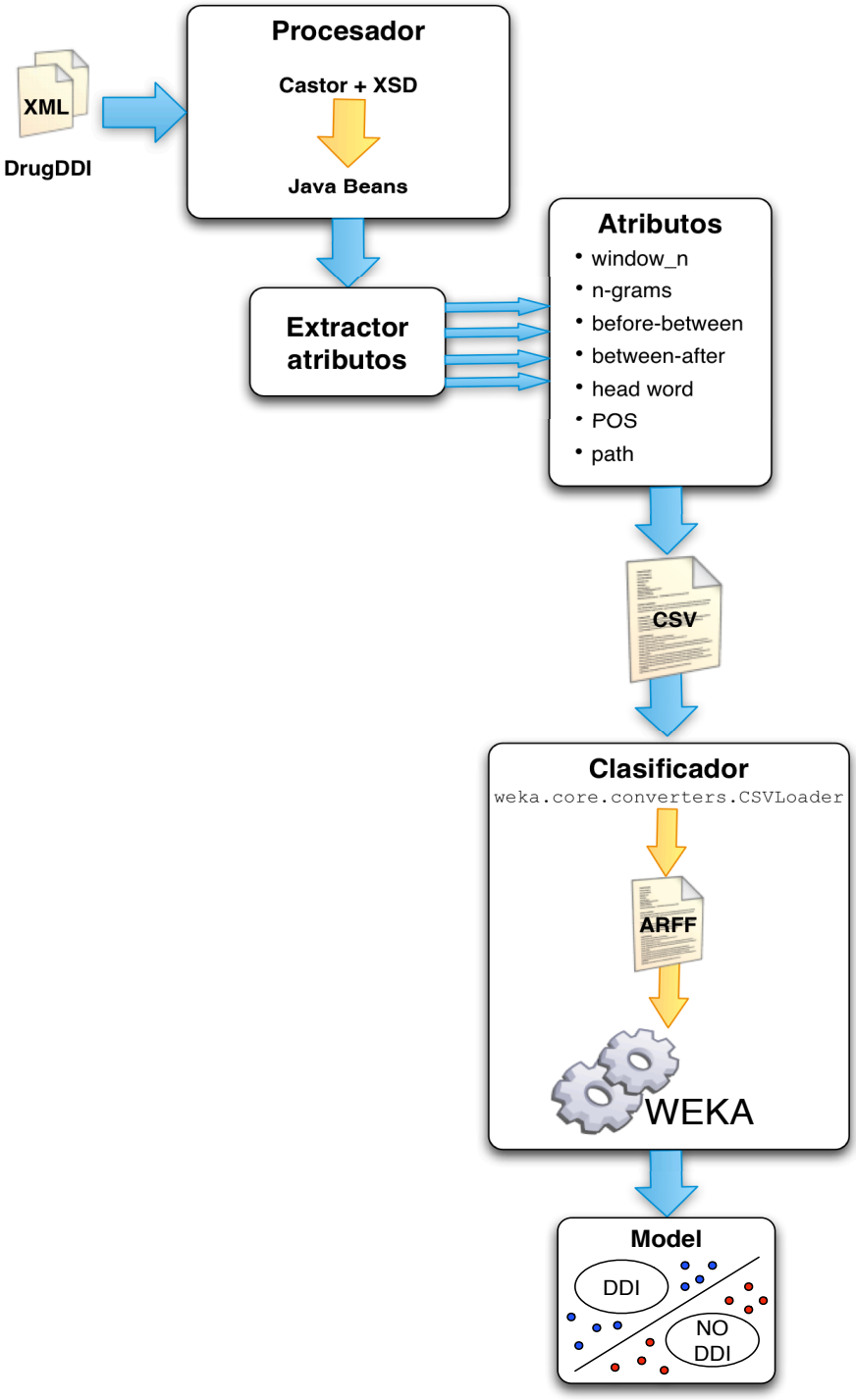


*Ilustración 8: Arquitectura del sistema Druida*

El sistema recibe como entrada el corpus DrugDDI, un conjunto de ficheros XML donde cada oración ha sido anotada con información semántica y sintáctica por la herramienta MMTx y las interacciones farmacológicas han sido anotadas manualmente. El procesador, que es el principal componente, lee los ficheros XML del corpus y mediante la tecnología Castor y los *XML Schema Definition* (XSD) mapea el contenido de los ficheros a instancias de Java. Los objetos Java tendrán como atributo raíz un conjunto de sentencias, las que se listan en cada fichero del corpus. En la siguiente imagen podemos ver un ejemplo de un fichero XML que se va a mapear a un objeto Java:

```
<?xml version="1.0" encoding="UTF-8"?>
<SENTENCES>
    <SENTENCE ID="s0"
        TEXT="Uricosuric Agents: Aspirin may decrease the effects of probenecid, sulfinpyrazone, and phenylbutazone.">
        <PHRASES>
            <PHRASE ID="s0.p0" NUMTOKENS="2" TEXT="Uricosuric Agents"
                TYPE="NP">
                <MAPPINGS>
                    <MAP CUI="C0041983" NAME="Uricosuric Agents" NAME_SHORT="Uricosuric Agents"
                        PROB="1000" SEMTYPES="phsu" USAN="NO">
                    </MAP>
                </MAPPINGS>
                <TOKENS>
                    <TOKEN ISHEAD="false" ORD="0" POS="adj" WORD="Uricosuric">
                    </TOKEN>
                    <TOKEN ISHEAD="true" ORD="1" POS="noun" WORD="Agents">
                    </TOKEN>
                </TOKENS>
            </PHRASE>
            <PHRASE ID="s0.p1" NUMTOKENS="1" TEXT="" TYPE="UNK" USAN="NO">
                <TOKENS>
                    <TOKEN ISHEAD="false" ORD="0" POS="colon" WORD=":">
                    </TOKEN>
                </TOKENS>
            </PHRASE>
            <PHRASE ID="s0.p2" NUMTOKENS="1" TEXT="Aspirin" TYPE="NP">
                <MAPPINGS>
                    <MAP CUI="C0004057" NAME="Aspirin" NAME_SHORT="Aspirin" PROB="1000"
                        SEMTYPES="orch,phsu" USAN="NO">
                    </MAP>
                </MAPPINGS>
                <TOKENS>
                    <TOKEN ISHEAD="true" ORD="0" POS="noun" WORD="Aspirin">
                    </TOKEN>
                </TOKENS>
            </PHRASE>
            <PHRASE ID="s0.p3" NUMTOKENS="1" TEXT="may" TYPE="VP">
                <TOKENS>
                    <TOKEN ISHEAD="false" ORD="0" POS="modal" WORD="may">
                    </TOKEN>
                </TOKENS>
            </PHRASE>
```

*Ilustración 9: Corpus DrugDDI*

Básicamente un fichero del corpus contiene un conjunto de oraciones. Cada oración se divide en frases (es decir, sintagmas) y *tokens*.

A continuación se leen los objetos Java y se realiza un procesamiento a nivel de oración: por cada oración se detectan los *tokens* que son fármacos mediante la comprobación de sus tipos semánticos. Una vez que se han identificado todos los fármacos, se procede a cruzarlos y a anotar si interaccionan entre sí o no, para posteriormente generar las instancias representadas por las características que se utilizarán en el clasificador. Es decir, por cada par de fármacos en una

oración se realizará el siguiente procesamiento:

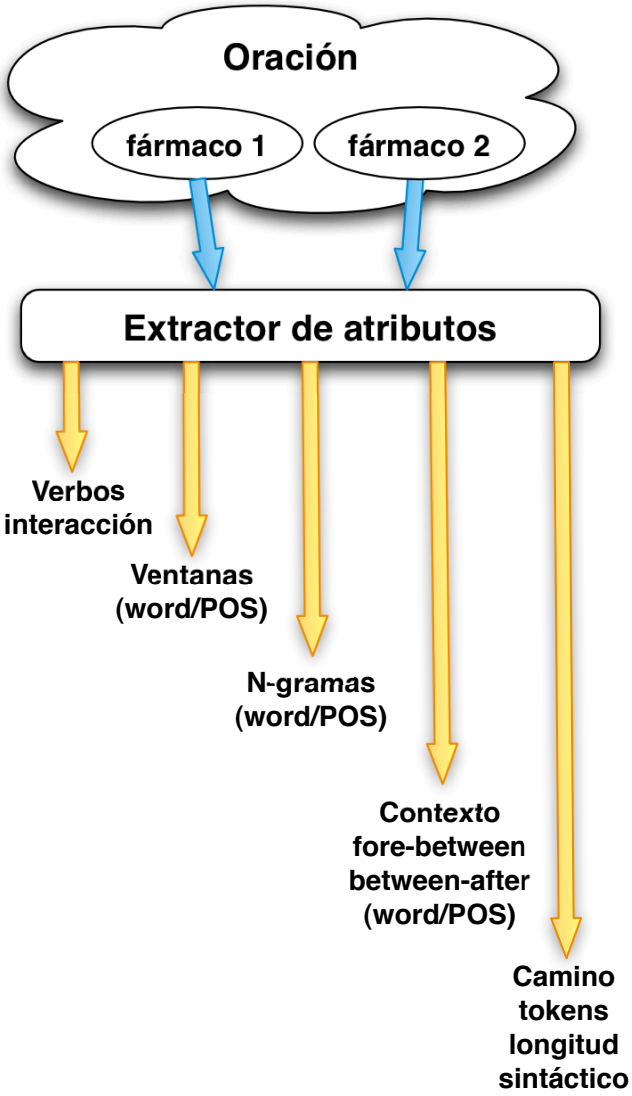


*Ilustración 10: Extracción de características*

En la sección Extracción de características se expone en mayor detalle cómo se han extraído los atributos.

Por último, las características se vuelcan a un fichero con formato CSV. Este volcado se realiza mediante un *logger* de la librería log4j[1], con un patrón específico, de tal manera que se escribe en un fichero en formato CSV la cabecera con los nombres de los atributos y a continuación una fila por cada instancia (par de fármacos) con todas las caracterísiticas y un valor que indica si hay interacción o no entre ambos fármacos. Este fichero se convierte posteriormente al formato de Weka *.arff* mediante la utilidad de Weka *CSVLoader*. Este fichero *.arff* será utilizado por Weka para entrenar los distintos algoritmos.

1 http://logging.apache.org/log4j/1.2/

En total nuestro fichero .arff contiene 22.375 instancias y 49 atributos.

## Descripción del corpus y los datasets utilizados

En primer lugar se va a explicar el corpus DrugDDI [1] a partir del cual partimos en este proyecto, para a continuación ver qué nuevos atributos vamos a extraer para generar nuestros conjuntos de datos.

### Corpus DDI

En la actualidad, existen diversos corpus del dominio biomédico anotados con relaciones entre proteínas, genes, o entidades biológicas. Algunos de estos corpus son de pequeño tamaño, otros contienen algunos errores o no están lo suficientemente bien construídos como para detectar todas las interacciones reales que existen en los textos analizados por ser complejas o estar anidadas.

Independientemente de la buena o mala calidad de estos corpus, no existía ninguno con relaciones entre fármacos hasta que se creó el corpus DrugDDI, que se considera el primer corpus anotado de interacciones entre fármacos. Gracias a este corpus se van a poder evaluar diferentes técnicas de detección de interaccinoes entre fármacos, tal como se hizo en la tesis [1], en el sistema Druida [4], y en el proyecto actual.

La base de datos DrugBank[2] fue utilizada para recolectar un conjunto de textos describiendo DDIs. Dicha base de datos contiene 13,242 DDIs descritas en los campos estructurados *Drug Interactions* para cada uno de los fármacos contenidos en la base de datos, mientras los campos *Interactions* contienen enlaces a documentos en texto plano que describen las interacciones de un determinado fármaco.

### Obtención del corpus

En la primera aproximación para obtener el corpus se eligieron aleatoriamente 1000 fármacos de la base de datos DrugBank y a través de la herramienta RobotMaker se descargaron los documentos que describen las interacciones para esos fármacos. En total se consiguieron 930 documentos, ya que algunos fármacos no tenían enlace alguno. Debido a que la labor de anotación puede llegar a ser bastante costosa, sólo fueron anotados 579 documentos (*annotated dataset*).

2 http://www.drugbank.ca/

**Procesamiento del corpus**

Para realizar el procesamiento se utilizó UMLS[3]. El principal objetivo de UMLS es proporcionar ayuda en el desarrollo de tecnologías de procesamiento de lenguaje natural en textos biomédicos. UMLS se compone de las siguientes partes:

- **Metathesaurus**: es una ontología del dominio biomédico. Todos los conceptos que contiene están categorizados según una anotación semántica de la UMLS Semantic Network[4].
- **Semantic Network**: tiene 135 tipos semánticos tales como *Pharmaceutical substance* (*phsu*), *Amino Acid, Peptide or Protein* (*aapp*), *Desease or Syndrome* (*dsyn*) o *Gene or Genome* (*gngm*).
- **Specialist Lexicon**: es un léxico biomédico con información sintáctica, morfológica y ortográfica.

En el procesamiento se utiliza la herramienta MMTx para analizar sintácticamente y semánticamente los documentos del corpus. Este programa mapea texto a conceptos del metasauro de UMLS. MMTx permite procesar sentencias, dividir en *tokens*, *PoS-tagging*, parseo sintáctico profundo y enlace de frases con conceptos UMLS. La salida es un documento en formato XML que contiene los tipos de frases identificados, que pueden ser *Noun phrase* (NP), *prepositional phrase* (PP), *verbal phrase* (VP), *adjectival phrase* (ADJ), *adverbial phrase* (ADV), *conjunctions* (CONJ) and *unknown* (UNK).

Una vez que se ha realizado en análisis sintáctico, MMTx busca las frases en el metatesauro de UMLS, y se obtiene por cada entidad el CUI (*concepto unique identifier*), su nombre y su tipo semántico.

**Anotación del corpus**

Tras revisar la UMLS Semantic Network para obtener los diferentes tipos semánticos que representan fármacos, se consideraron los siguientes:

- **Clinical drug**
- **Pharmacological substante**
- **Antibiotic**
- **Biologically Active Substance**
- **Chemical Viewed Structurally**

3 http://www.nlm.nih.gov/research/umls/
4 http://semanticnetwork.nlm.nih.gov/

- **Amino Acid**, **Peptide**, or **Protein**

En el Corpus DDI sólo se anotaron las interacciones a nivel de oración. Las interacciones que abarcaban varias oraciones no fueron anotadas.

En la Tabla 3 se muestra información sobre el total de documentos, sentencias, frases y tokens que componen el corpus DrugDDI:

| | Conjunto de datos |
|---|---|
| Documentos | 579 |
| Oraciones | 5,806 |
| Frases | 66,021 |
| Tokens | 127,653 |

*Tabla 3: Estadísticas de los elementos del corpus DrugDDI*

Estos datos se han obtenido de [1]. De los totales mostrados en la Tabla 4, podemos observar el número de elementos que contienen interacciones entre fármacos y los que no las contienen:

| | Total | No DDI | DDI |
|---|---|---|---|
| Ficheros | 579 | 164 | 415 |
| Oraciones | 5,806 | 3,762 | 2,044 |
| Total de DDIs anotados | | 3160 | |

*Tabla 4: Estadísticas de DDIs en el conjunto de datos*

Por último se muestra la media de sentencias con DDIs por documento:

| | Media por documento |
|---|---|
| Nº de sentencias | 10,03 |
| Nº de sentencias con al menos una DDI | 3,53 |
| Nº de sentencias que no contienen DDI | 6,5 |
| Nº de DDIs | 5,46 (0,54 por documento) |

*Tabla 5: Media de sentencias con DDI por documento*

## Extracción de características

Los ejemplos (instancias) que vamos a proporcionar a los clasificadores se obtienen a partir del corpus DrugDDI. A partir de la información sintáctica y semántica de cada oración vamos a generar una serie de atributos que son los que van a servir para entrenar los algoritmos.

En este proyecto hemos utilizado mucha información sintáctica relacionada con ambos fármacos para generar los atributos, aparte de otros atributos que indican si el verbo que relaciona

los fármacos es negativo, modal, o de interacción, que ya fueron utilizados en [4]. A continuación se va a detallar el proceso de extracción de atributos.

En primer lugar se extraen tres características heredadas del sistema Druida. Éstas se refieren al tipo de interacción que está entre los dos fármacos:

- Existe un verbo de interacción (*acceler*, *alter*, *decrease*, …)
- Existe negación en la interacción (*no*, *not*, *neither*, *without*, *lack*, ….)
- Existe verbo modal en la interacción (*may*, *might*, etc)

Estos valores se representan como atributos *booleans* (1 ó 0).

El resto de atributos se podrían organizar en tres bloques:

1. Información básica de cada fármaco
2. Ventana o palabras que rodean cada fármaco
3. Contexto de cada fármaco

Veamos cómo se obtienen los atributos en cada bloque:

**Información básica de cada fármaco**

Para obtener estos atributos no ha sido necesario hacer ningún tipo de procesamiento ya que toda la información se encuentra en el corpus DrugDDI. Estos datos son la(s) palabra(s) del fármaco, su PoS, que es la categoría morfosintáctica (*part-of-speech*), el *head word*, que indica si una palabra dentro de una frase es la palabra principal, y el PoS del *head word*. Simplemente ha sido necesario leer los atributos del elemento TOKEN. En concreto *ISHEAD*, *POS* y *WORD*.

Los nombres de los atributos se han formado concatenando el tipo de información que representa (*word*, *hw*, *word_pos*, *hw_pos*) con la cadena *_drug1* si se trata del primer fármaco, o con la cadena *_drug2* si se trata del segundo. Por ejemplo, dada la oración

*Aspirin may decrease the effects of probenecid, sulfinpyrazone, and phenylbutazone*

podemos extraer los siguientes atributos para el primer fármaco, siendo éste *Aspirin*:

- *words_drug1* → aspirin
- *word_pos_drug1* → noun
- *hw_drug1* → Aspirin
- *hw_pos_drug1* → noun

También se han formado combinaciones de los anteriores atributos por cada par de fármacos, de tal manera que por cada par de fármacos se han combinado todas sus características de todas

las maneras posibles. A continuación se detallan los cruces de características para el par de fármacos *Aspirin-probenecid* dentro de la oración *Aspirin may decrease the effects of probenecid, sulfinpyrazone, and phenylbutazone*:

- h*w_drug1/hw_drug2* → combinación de los *head word* de ambos fármacos. Por ejemplo: Aspirin_probenecid
- *hw_drug1/pos_hw_drug2* → combinación del *head word* del primer fármaco con el PoS del *head word* del segundo. Por ejemplo: Aspirin_noun
- *pos_hw_drug1/pos_hw_drug2* → combinación de los PoS de los *head word* de ambos fármacos. Por ejemplo: noun_noun
- *pos_hw_drug1/hw_drug2* → combinación del PoS del primer fármaco con el head word del segundo. Por ejemplo: noun_probenecid

**Ventana de cada fármaco**

La ventana es un determinado número de tokens que rodea a un fármaco. Las ventanas pueden ser de diferente tamaño, esto quiere decir que se obtiene el número de tokens anteriores y posteriores al fármaco indicado por el tamaño de la ventana. Por ejemplo, en la siguiente secuencia de palabras, dado un fármaco representado por la palabra *drug* dentro de una sentencia, se resalta una ventana de tamaño 1:

token1 token2 **drug** token3 token4

Si queremos obtener la ventana de tamaño 2, necesitaríamos los dos tokens anteriores y posteriores al fármaco:

token1 token2 **drug** token3 token4

En este proyecto se han obtenido ventanas de tamaños desde 1 hasta 5. Además, por cada fármaco y tamaño se han creado dos ventanas con diferentes tipos de tokens:

- ventanas de palabras
- ventanas de PoS

Los nombres de los atributos se han formado siguiendo el siguiente patrón:

window{tamaño}_[word|pos]_drug{nº fármaco}

De esta manera, el atributo *window2_pos_drug1* representaría la ventana de tamaño 2 de los PoS del primer fármaco, y el atributo *window4_word_drug2* representaría la ventana de tamaño 4 de las palabras que rodean al segundo fármaco.

En la obtención de las ventanas se ha realizado un procesamiento especial para eliminar ruido en los atributos. En ocasiones hay listados de fármacos que no interaccionan entre sí, pero que sí que lo hacen con otro fármaco. Por ejemplo, dada la oración:

*Aspirin may decrease the effects of probenecid, sulfinpyrazone, and phenylbutazone*

vemos que hay tres fármacos: *probenecid*, *sulfinpyrazone, phenylbutazone*, que no interaccionan entre ellos, pero que sin embargo todos interaccionan con *Aspirin*. Al obtener la ventana de cualquier fármaco de la lista vemos que está rodeado de otros fármacos que probablemente no aportarán información útil a la hora de clasificar instancias (pares de fármacos). Si queremos obtener la ventana de tamaño 5 de *phenylbutazone*, tendríamos la secuencia:

*effects of probenecid, sulfinpyrazone, and* ***phenylbutazone***

Si ignorásemos los demás fármacos de la lista, la ventana sería:

*may decrease the effects of ~~probenecid, sulfinpyrazone, and~~* ***phenylbutazone***

En este último caso la secuencia es más discriminatoria ya que un listado de fármacos puede variar bastante de una oración a otra, aparte de que en el segundo caso se detecta más claramente una relación entre entidades.

Por este motivo, al obtener las ventanas se aplican unas expresiones regulares que detectan el patrón de un listado de fármacos, teniendo en cuenta que las palabras deben ser fármacos separadas por comas y/o conjunciones. Si se detectan estos patrones se eliminan, dejando únicamente el fármaco de la relación con el resto de palabras.

Por otro lado el nombre del fármaco cuya ventana se obtiene se representa con la cadena *drugwindow*. De esta manera el valor completo del atributo será más común a la hora de entrenar un clasificador y se podrá generar un modelo más robusto. Los nombres de los fármacos son individuales y si hay una interacción entre dos fármacos lo que más nos interesa conocer es cómo se realiza la interacción, qué patrones aparecen en la oración que hacen que dos fármacos interactúen entre sí. Por este mismo motivo también se ha considerado sustituir cualquier nombre de fármaco o PoS de fármaco con la cadena *drug*.

Para la oración *Aspirin may decrease the effects of probenecid, sulfinpyrazone, and phenylbutazone, a*lgunos ejemplos de atributos de ventana serían los siguientes:

- *window2_word_drug1* → empty_empty_drugwindow_may_decrease
- *window4_pos_drug1* → empty_empty_empty_empty_drugwindow_modal_verb_det_noun
- *window3_word_drug2* → the_effects_of_drugwindow

**Contexto de cada fármaco**

Por último se obtiene información del contexto en el que se encuentran los dos fármacos. En primer lugar se ha procesado el contenido que está entre ambos fármacos. A esta parte se le llama *camino*. Para obtenerlo ha sido necesario recorrer todos los tokens entre los dos fármacos y almacenar información relevante de los mismos.

Por cada par de fármacos se ha obtenido la siguiente información que se encuentra entre ambos dentro de una oración:

- **palabra**: son los atributos *WORD* de los tokens del camino. Si ese token es un fármaco, la palabra se representa por la cadena *DRUG*, de esta manera el clasificador podrá generalizar mejor a la hora de generar el modelo ya que el nombre concreto del fármaco no es relevante. El atributo se representa por la cadena *path_tokens*.
- **PoS**: son los atributos *POS* de los tokens del camino. Al igual que con las palabras (caso anterior), si se trata de un fármaco se representa por la cadena *DRUG*. Este atributo se representa por la cadena *path_pos*.
- **Tipo sintáctico**: son los tipos sintácticos que se encuentran entre ambos fármacos. Dentro del corpus se obtiene del atributo *TYPE* de una oración. El atributo que lo representa es *path_syntactic_type*.
- **Longitud del camino entre fármacos**: es el número de tokens entre ambos fármacos. Se representa por la cadena *path_length*.
- **Longitud del camino de tipos sintácticos**: es el número de tipos sintácticos entre ambos fármacos. Lo representa la cadena *path_syntactic_type_length*.

También se han generado atributos para representar el contexto entre ambos fármacos mediante N-gramas, en concreto con bigramas y trigramas. Los N-gramas son grupos de *n* tokens. En nuestro caso se han generado secuencias de 2 y 3 palabras de entre todas las que hay en el camino. Cada N-grama se ha separado por el símbolo *$*. En el siguiente ejemplo podemos ver cómo se han generado los bigramas entre el par de fármacos *aspirin* y *probenecid* dentro de la oración*:*

***Aspirin** may decrease the effects of **probenecid**, sulfinpyrazone, and phenylbutazone*

- may_decrease
- decrease_the
- the_effects

Si concatenamos estos bigramas con el carácter *$*, tendremos el valor de un atributo que representa los bigramas de palabras entre dos fármacos:

*may_decrease$decrease_the$the_effects*

En este proyecto se han creado N-gramas de las palabras y de los PoS entre cada par de fármacos. El sistema cuenta con un nuevo módulo para crear N-gramas de diferentes tamaños.

A continuación se muestran algunos ejemplos de los atributos que representan el contexto entre el par de fármacos *aspirin* y *probenecid* dentro de la oración *Aspirin may decrease the effects of probenecid, sulfinpyrazone, and phenylbutazone*:

- *path_tokens* → may_decrease_the_effects
- *path_pos* → modal_verb_det_noun
- *path_length* → 5
- *path_syntactic_type* → vp_vp_np
- *path_syntactic_type_length* → 3
- *bigrams* → may_decrease$decrease_the$the_effects (tres secuencias de dos palabras)
- *trigrams* → may_decrease_the$decrease_the_effects (dos secuencias de tres palabras)
- *pos_bigrams* → modal_verb$verb_det$det_noun (tres secuencias de dos PoS)
- *pos_trigrams* → modal_verb_det$verb_det_noun (dos secuencias de tres PoS)

En todos estos casos se han aplicado todas las posibles generalidades comentadas anteriormente, tales como ignorar los fármacos de una lista que no participan en la relación, o mostrar las palabras o PoS de cada fármaco con la palabra *DRUG*.

Por otro lado, se ha obtenido el contexto *fore-between*, *between* y *between-after* de cada fármaco. Estos tres patrones vienen del trabajo realizado en  Bunescu y Mooney [5]. Ellos detectaron que si una sentencia contiene una relación entre dos entidades, la relación se puede representar por algunos de los siguientes patrones:

- *fore-between*: palabras situadas antes y entre las dos entidades de la relación.
- *between*: palabras entre las dos entidades.
- *between-after*: palabras situadas entre y a continuación de las dos entidades involucradas en la relación.

En nuestro caso se ha extraído un contexto de tres tokens. A continuación se muestran tres ejemplos del contexto para el par de fármacos *aspirin* y *probenecid* dentro de la oración *Aspirin may decrease the effects of probenecid, sulfinpyrazone, and phenylbutazone.*

- *before-between*: en este caso no existen palabras delante del primer fármaco, por lo que el contexto vienen a ser las palabras situadas entre las dos entidades:

  ***Aspirin** may decrease the effects of **probenecid**, sulfinpyrazone, and phenylbutazone*

- *between*: son las palabras situadas entre los dos fármacos:

  ***Aspirin** may decrease the effects of **probenecid**, sulfinpyrazone, and phenylbutazone*

- *between-after*: en este caso se eliminan las palabras que vienen a continuación de *probenecid.* Ya que es un listado de fármacos, tal como se explicó en la sección anterior (Ventana de un fármaco), los fármacos situados en listas que no forman parte de la relación se ignoran. Por lo que en nuestro ejemplo no tendríamos palabras a continuación del segundo fármaco:

***Aspirin** may decrease the effects of **probenecid**~~, sulfinpyrazone, and phenylbutazone~~*

En este proyecto se ha extraído el contexto tanto para las palabras como para los PoS de cada fármaco.

A modo de generalización, en estos atributos el primer fármaco se representa por *DRUG1* y el segundo por *DRUG2*. De esta manera el valor completo del atributo será más común a la hora de entrenar un clasificador ya que en una interacción no es importante saber qué fármacos interactúan, si no los patrones de la interacción. Mediante estos patrones el modelo generado por el clasificador será más robusto y podrá generalizar mejor a partir de nuevas instancias.

A continuación se muestran algunos ejemplos de estos atributos para la oración *Aspirin may decrease the effects of probenecid, sulfinpyrazone, and phenylbutazone*:

- *fore_between_pos* → DRUG1_modal_verb_det_noun_DRUG2
- *fore_between_word* → DRUG1_may_decrease_the_effects_DRUG2
- *between_after_word* → DRUG1_may_decrease_the_effects_DRUG2
- between_after_pos → DRUG1_modal_verb_det_noun_DRUG2

### Selección de características

La selección de atributos para generar los datos de entrenamiento es un trabajo complicado y que requiere tiempo, ya que en función de qué atributos se hayan elegido y cómo sean de relevantes, los algoritmos serán más o menos capaces de extraer un buen modelo a partir de los datos. Aunque se puede conseguir un buen rendimiento usando un conjunto de características simple y obvio, normalmente se pueden conseguir ganancias eligiendo un conjunto de

características de calidad basado en una profunda comprensión de la tarea que se esté tratando.

Normalmente las características que se extraen se construyen a través de un proceso de prueba-error, guiado por intuiciones sobre qué información es relevante para el problema. Aun así, puede ser conveniente establecer límites en el número de atributos que se pueden utilizar en un algoritmo de aprendizaje. Si se proporcionan demasiadas características el algoritmo tendrá muchas opciones de los datos de entrenamiento en las cuales confiar, que puede que no generalicen bien con nuevos ejemplos. Este problema se conoce como *overfitting*, y puede resultar bastante problemático cuando se trabaja con pequeños conjuntos de datos. La herramienta Weka tiene una sección que permite seleccionar qué caracterísiticas son las más relevantes.

La técnica **Selección de Características** (*Feature selection)* se usa mucho en aprendizaje automático, y consiste en seleccionar un subconjunto relevante de atributos para aplicarlos a un algoritmo de aprendezaje. En este proceso se crean todas las posibles combinaciones de atributos para ver qué subconjunto de los atributos funciona mejor para la predicción, y para detectar qué atributos generan ruido en los datos. Los atributos irrelevantes o redundantes se eliminan. Es una parte muy importante del preprocesamiento de datos, ya que es una manera de evitar la llamada **maldición de la dimensionalidad** (*curse of dimensionality*), un problema que surge cuando tenemos muchos atributos respecto a la cantidad de instancias, y en consecuencia puede haber muchos grados de libertad, por lo que los patrones extraídos pueden ser poco robustos. Seleccionar los atributos más significativos también permite que el proceso de aprendizaje pueda ser más rápido y la mejora de la interpretabilidad del modelo.

En este proyecto se han aplicado dos técnicas diferentes para seleccionar atributos y los experimentos se han realizado con ambos subconjuntos para comprobar cuál daba mejores resultados. Los algoritmos utilizados para la selección de atributos se han elegido en base a los utilizados en el artículo [6]. Es este artículo se hicieron experimentos con diferentes algoritmos de selección de atributos:

- *Correlation Feature Subset Selection*
- *Symmetrical Uncertain*
- *Information Gain*
- *Gain Ratio*
- *Relief*
- *Chi Squared*

A partir de los diferentes experimentos que se llevaron a cabo en [6], los algoritmos que dieron

mejores resultados en cuanto a la reducción de atributos fueron *Chi Squared*, *Gain Ratio*, *CFS* y *Relief*. De éstos, en el presente proyecto hemos utilizado **Gain Ratio** y **CFS**.

A continuación se muestra la reducción de atributos que han realizado los algoritmos que vamos a utilizar en el presente proyecto:

| Algoritmo | Nº atributos | Reducción |
|---|---|---|
| Gain Ratio | 17 | 65,31 % |
| Correlation Feature Subset Selection | 14 | 71,43 % |

*Tabla 6: Reducción de atributos*

Tal como se puede observar, con el algoritmo *Gain Ratio* se obtiene menor reducción de atributos. En la sección de los experimentos veremos que este algoritmo da mejores resultados que *CFS*.

# Experimentos

## Proceso

Una vez extraídas las características se han llevado a cabo los experimentos. A continuación se detallan los pasos del proceso realizado junto con las dificultades encontradas:

- **Selección de características**. Se aplican los algoritmos *Correlation Feature Selection* y *Gain Ratio* para generar subconjuntos de atributos y de esta manera evitar problemas de memoria debido a la cantidad de datos.
- **Experimentos**. Se realizan experimentos con los métodos de evaluación *cross-validation* y *training-test*. Con la evaluación *cross-validation* se llevaron a cabo experimentos con diferentes clasificadores intentando cubrir todas las técnicas generales de aprendizaje automático y a la vez guiándonos por el artículo [6], ya que los algoritmos que se citan en este artículo se han utilizado anteriormente en tareas de clasificación de textos dando buenos resltados; además estos algoritmos tienen implementaciones eficientes y el modelo resultante permite un rápido proceso de clasificación. En concreto, se realizaron experimentos con los siguientes algoritmos:
  - Naïve Bayes
  - Hyperpipes
  - LogitBoost
  - Ibk
  - J48
  - AdaBoostM1
  - SMO
  - MultilayerPerceptron

  Cada experimento se realizó con ambos conjuntos de datos generados por medio de los algoritmos de selección de características *Gain Ratio* y *Correlation Feature Selection*. De entre todos estos experimentos se han elegido los algoritmos que mejor resultado han dado (Naïve Bayes, Hyperpipes, Ibk y LogitBoost) para posteriormente realizar más experimentos con el método de evaluación *training-test*.
- **Personalización de los parámetros de memoria de la JVM**: en gran parte del proceso de experimentación hemos sufrido muchos problemas de memoria. Los errores de

memoria con Weka son un problema recurrente que aparece sin que existan grandes cantidades de datos. Para poder solucionarlos tuvimos que personalizar algunos parámetros de la Máquina Virtual de Java para poder ejecutar Weka con más memoria que la que el sistema operativo asigna al proceso por defecto. En primer lugar, a la ejecución del programa Weka se le añadió el parámetro *-Xmx* para asignar el máximo tamaño de la pila de Java; con este parámetro establecemos el tamaño máximo de la cantidad de memoria que la JVM va a utiliar para ejecutar Weka. Aparte, también se incluyó en la ejecución de Weka el parámetro de la máquina virtual *-XX:+UseParallelGC*, este parámetro hace que el recolector de memoria de la JVM se utilice en paralelo por medio de los hilos disponibles en la máquina. Con estos dos parámetros se han conseguido eliminar práctiamente todos los problemas de memoria y se ha podido llevar a cabo la mayoría de los experimentos.

### Experimentos – evaluación cross-validation

En los primeros experimentos se ha utilizado el método de evaluación *cross-validation*. Con esta técnica se subdivide el corpus original en *k* subconjuntos llamados *folds* (o pliegues). Para cada *fold*, entrenamos un modelo utilizando todos los datos excepto los datos pertenecientes a dicho subconjunto, y a continuación se prueba el modelo con los datos del subconjunto en cuestión. La elección del valor *k* dependerá del tamaño y características del conjunto de datos, aunque el valor más común es 10, ya que se ha comprobado a través de muchas pruebas con diferentes técnicas de aprendizaje, que dividiento el corpus en 10 subconjuntos se obtiene la mejor estimación de error. En nuestros experimentos hemos utilizado validación cruzada de 10 pliegues.

El fichero de entrenamiento de estos experimentos tenía un total de 22.375 instancias, de las cuales un 11.62% eran interacciones entre fármacos, y el resto, un 88.38%, no lo eran.

En todos los experimentos se ha aplicado un selector de atributos para trabajar con un conjunto de datos más pequeño pero más eficaz, ya que según la calidad del selector de atritubos nuestros datos permitirán al clasificador generar un modelo más robusto. Los algoritmos que hemos utilizado son *Gain Ratio* y *Correlation Feature Selection* basándonos en [6].

A continuación se exponen los mejores resultados que hemos encontrado ejecutando experimentos con diferentes clasificadores:

| Experimento (cross-validation) | Precision | Recall | F-score |
|---|---|---|---|
| CFS - Naïve Bayes | 0,262 | 0,792 | 0,393 |
| CFS - Hyperpipes | 0,609 | 0,344 | 0,440 |
| CFS - LogitBoost | 0,638 | *0,052* | *0,095* |
| CFS - Ibk | 0,605 | 0,491 | 0,542 |
| GainRatio - Naïve Bayes | *0,239* | **0,818** | 0,370 |
| GainRatio - Hyperpipes | 0,592 | 0,353 | 0,442 |
| GainRatio - LogitBoost | **0,667** | 0,072 | 0,129 |
| GainRatio - Ibk | 0,613 | 0,487 | **0,543** |

*Tabla 7: Resultados experimentos cross-validation*

Tal como se puede observar en la Tabla 7, los resultados resultan bastante variados. Si nos fijamos en la precisión y el recall, en el caso de Naïve Bayes y LogitBoost muestran una gran variabilidad entre ambas métricas. En Naïve Bayes, con ambos conjuntos de datos, la precisión es muy baja, mientras que el recall es alto. Sin embargo en LogitBoost, la precisión es relativamente alta, mientras que el recall es muy bajo. Los algoritmos Ibk e Hyperpipes tienen estos valores más cercanos, en estos casos los valores de precisión y recall se aproximan más.

En cuanto al valor de la medida-F, que combina la precisón y el recall, los valores no son muy altos. Teniendo en cuenta que el mejor valor de la medida-F es 1 y el peor 0, vemos que el algoritmo que peor se comporta es LogitBoost para ambos conjuntos de datos. Los valores que da de 0,095/0,129 (CFS/GR) son los más bajos de todos los resultados. Sin embargo, los valores más altos corresponden a Ibk, siendo 0,542/0,543 (CFS/GR).

Si nos fijamos en los diferentes conjuntos de datos, dan un ligero mejor resultado los que han sido filtrados con el selector de atributos *GainRatio*. Calculando la media de ambos conjuntos, los experimentos con *CFS* dan un valor de 0,3675, mientras que los que tienen atributos obtenidos con *GainRatio*, dan una media de 0,371. Tal como podemos ver, la diferencia es mínima.

Del resto de experimentos que se intentaron llevar a cabo, con los algoritmos SMO y MultilayerPerceptron no pudimos obtener resultados ya que ambos dieron errores de memoria. Por otro lado, los resultados obtenidos por J48 y AdaBoostM1 los consideramos inválidos, ya que el valor de verdaderos negativos era 0, y esto no es posible en nuestro proyecto ya que sí que existen verdaderos negativos.

**Experimentos – evaluación training-test**

A continuación se han realizado experimentos con el método de evaluación *training-test*. En este método tenemos todos los datos repartidos en dos conjuntos: entrenamiento y pruebas. Los datos del entrenamiento se van a utilizar para entrenar el modelo, mientras que los datos de pruebas se utilizan para validar el modelo. Estos datos de entrenamiento y pruebas vienen del mismo corpus DrugDDI. Estos experimentos se llevan a cabo para hacer una comparación con los resultados obtenidos con el método kernel aplicado en [1]; estos experimentos se realizaron sobre la misma distribución de datos que se va a aplicar a continuación.

El fichero de entrenamiento de estos experimentos tiene un total de 17.906 instancias, de las cuales un 11.05% son interacciones entre fármacos, mientras que el resto, un 88.95% no lo son. El conjunto test está compuesto por el 25% (aproximadamente 145) de los ficheros del corpus total (579 documentos). El fichero de prueba o evaluación tiene 4.469 instancias, de las cuales un 13.94% son interacciones entre fármacos y un 86.06% no lo son.

En estos experimentos hemos tenido un problema que nos ha obligado a utilizar un clasificador *envoltorio* para realizar cierto procesamiento antes de entrenar nuestros algoritmos. El problema era que la mayoría de las cabeceras de los ficheros de entrenamiento y evaluación no coincidían. Estas cabeceras contienen valores nominales y debido a la cantidad de datos diferentes que hay de los valores de estos atributos, se hizo necesario ejecutar previamente el clasificador InputMappedClassifier[5]. Este clasificador actúa como un envoltorio sobre el nuestro que resuelve el problema de incompatibilidad entre los ficheros de entrenamiento y evaluación, mapeando los datos utilizados en el entrenamiento del modelo con los datos que entran a evaluar el modelo. Este clasificador es nuevo en Weka desde la versión 3.7.3, que está actualmente en desarrollo. Debido a que se ha tenido que utilizar esta versión no se ha encontrado el clasificador Hyperpipes, no se sabe si porque lo eliminan o porque aún no lo han añadido en esta última versión. En consecuencia, en estos experimentos no hay resultados para el clasificador Hyperpipes.

5 http://weka.sourceforge.net/doc.dev/weka/classifiers/misc/InputMappedClassifier.html

A continuación se muestran en la siguiente tabla los diferentes experimentos:

| Experimento (training-test) | Precision | Recall | F-score |
|---|---|---|---|
| CFS - Naïve Bayes | 0,755 | 0,193 | 0,307 |
| CFS - Ibk | *0,612* | 0,302 | 0,404 |
| CFS - LogitBoost | **0,8** | *0,013* | *0,025* |
| GainRatio - Naïve Bayes | 0,71 | 0,326 | 0,447 |
| GainRatio - Ibk | 0,678 | **0,342** | **0,455** |
| GainRatio - LogitBoost | 0,692 | 0,058 | 0,107 |

*Tabla 8: Resultados experimentos training-test*

Tal como podemos observar en la Tabla 8, los resultados son peores que los obtenidos en los anteriores experimentos. En cuanto a la precisión y el recall, todos tienen un desiquilibrio considerable, sobre todo en los casos de Naïve Bayes y LogitBoost. Ambos casos tienen un recall bastante bajo comparado con la precisión..

Si nos fijamos al valor de la medida-F, podemos comprobar que al igual que en los anteriores experimentos el algoritmo con peor resultados es LogitBoost, con una medida de 0,025/0,107(CFS/GR). Sin embargo el experimento que mejor resultado ha dado es Ibk, con una medida-F de 0,404/0,455 (CFS/GR), seguido por Naïve Bayes con una medida de 0,307/0,447 (CFS/GR).

En cuanto a los diferentes conjuntos de datos, en general dan mejor resultado los que han sido filtrados por el selector de atributos *GainRatio*. Si calculamos la media de ambos conjuntos, los experimentos con *CFS* dan una media de medida-F de 0,245, mientras que los que tienen atributos obtenidos con *GainRatio*, dan una media de 0,336. Tal como podemos ver, aunque la diferencia entre ambos conjuntos de datos no es muy grande, sí que deja ver que las características seleccionadas con *GainRatio* dan mejor resultado.

En la siguiente tabla vamos a comparar la mejor medida-F de los experimentos realizados con evaluación *cross-validation* con los resultados de los evaluados con *training-test*.

| Experimento | F-score cross-validation | F-score training-test |
|---|---|---|
| Naïve Bayes | 0,393 | **0,447** |
| Ibk | **0,543** | 0,455 |
| LogitBoost | **0,129** | 0,107 |

*Tabla 9: Comparación resultados cross-validation/training-test*

Según podemos ver en la Tabla 9, la evaluación con el método *cross-validation* da mejor resultado con Ibk y LogitBoost. Sin embargo, el algoritmo Naïve Bayes da un resultado mucho mejor con *training-test*. Según esta tabla los resultados tienen una diferencia considerable si cambiamos el método de evaluación.

### Discusión

A continuación vamos a comparar nuestros resultados con los obtenidos mediante el método kernel aplicado en [1].

En primer lugar vamos a comparar nuestros mejores resultados obtenidos con el método *training-test* con los obtenidos en [1] con el método kernel, evaluados también con *training-test*.:

| Experimento | Precisión | Recall | F-score |
|---|---|---|---|
| Ibk (training-test) | **0,678** | 0,342 | 0,455 |
| Método kernel aplicado en [1] (training-test) | 0,50 | **0,71** | **0,58** |

*Tabla 10: Comparación resultados Ibk (training-test) - método kernel aplicado en [1](training-test)*

Tal como podemos observar en la Tabla 10, el método kernel da mejor resultado que los obtenidos en este proyecto con la evaluación *training-test*. La diferencia más grande se da en el recall, y es de 36.8 puntos. En la precisión la diferencia es 17.8 puntos, que a pesar de ser menor también es considerable. Ésta ha sido la única medida en la que se ha obtenido mejor resultado en este proyecto. La medida-F tiene una diferencia de 12.5 puntos.

En la siguiente gráfica podemos ver de una manera visual cómo difieren los resultados de ambos experimentos:

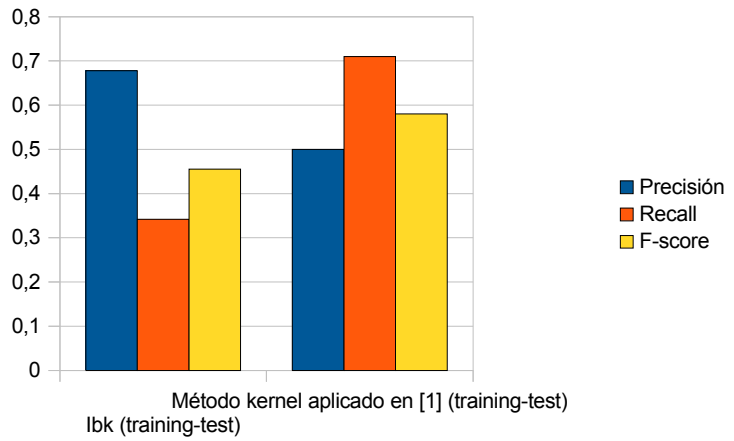


*Ilustración 11: Comparación resultados Ibk (training-test) - método kernel aplicado en [1] (training-test)*

En la siguiente tabla podemos ver la comparación entre nuestros resultados y los obtenidos en [1] pero evaluando ambos experimentos con *cross-validation*:

| Experimento | Precisión | Recall | F-score |
|---|---|---|---|
| Ibk (cross-validation) | **0,613** | 0,487 | 0,543 |
| Método kernel aplicado en [1] (cross-validation) | 0,57 | **0,800** | **0,640** |

*Tabla 11: Comparación resultados Ibk (cross-validation) - método kernel aplicado en [1] (cross-validation)*

Al igual que con la evaluación *training-test*, los resultados obtenidos en [1] son mejores que los obtenidos en este proyecto con la evaluación *cross-validation*. La diferencia más grande se da en el recall, y es de 31.3 puntos. En la precisión la diferencia es 4.3 puntos, que aunque menor, también es considerable. La medida-F tiene una diferencia de 9.7 puntos. Tal como podemos ver, en esta comparación la diferencia entre los resultados es bastante menor que en el caso anterior.

En definitiva podemos concluir que el método kernel aplicado en [1] es capaz de detectar en mejor medida si dos fármacos interaccionan entre sí.

En la Ilustración 12 podemos ver de una manera visual cómo difieren los resultados de ambos experimentos:

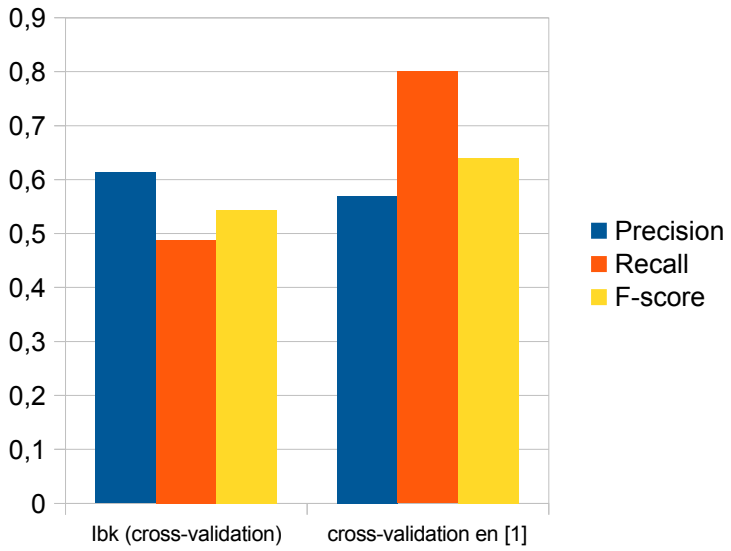


*Ilustración 12: Comparación resultados Ibk (cross-validation) – método kernel aplicado en [1] (cross-validation)*

A continuación se comparan nuestros resultados con los obtenidos mediante el algoritmo SVM aplicado en el proyecto Druida [4]. En la Tabla 12 se muestran los mejores resultados obtenidos en los experimentos realizados en el presente proyecto comparados con los mejores resultados obtenidos en [4]. Ambos experimentos se realizon con la evaluación *cross-validation*, con 10 *folds* en Ibk, y 3 *folds* en SVM*:*

| Experimento | Precisión | Recall | F-score |
|---|---|---|---|
| Ibk (cross-validation) | **0,613** | **0,487** | **0,543** |
| SVM aplicado en [4] (cross-validation) | 0,447 | 0,127 | 0,198 |

*Tabla 12: Comparación resultados Ibk (cross-validation) - SVM aplicado en [4] (cross-validation)*

Según podemos observar entre estos resultados existe una gran diferencia. Los resultados son mayores para las tres medidas con el algoritmo Ibk y evaluación *cross-validation*.

En la siguiente gráfica se muestra la comparación de los resultados obtenidos en este proyecto con Ibk y los obtenidos en [4] con SVM:

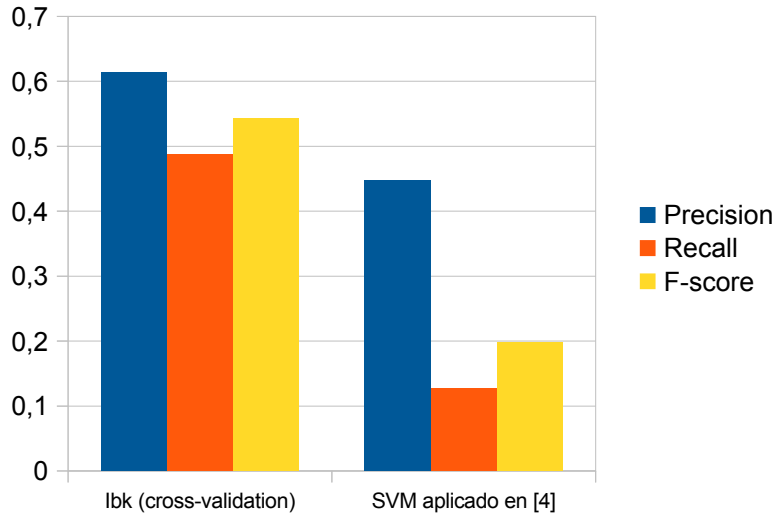


*Ilustración 13: Comparación Ibk-SVM aplicado en [4]*

Por último, en la Ilustración 13 se muestra una comparativa de los mejores resultados obtenidos en [1] con método kernel, en [4] con SVM y en el presente proyecto con Ibk. De esta manera podremos visualizar más claramente las diferencias entre los tres métodos:

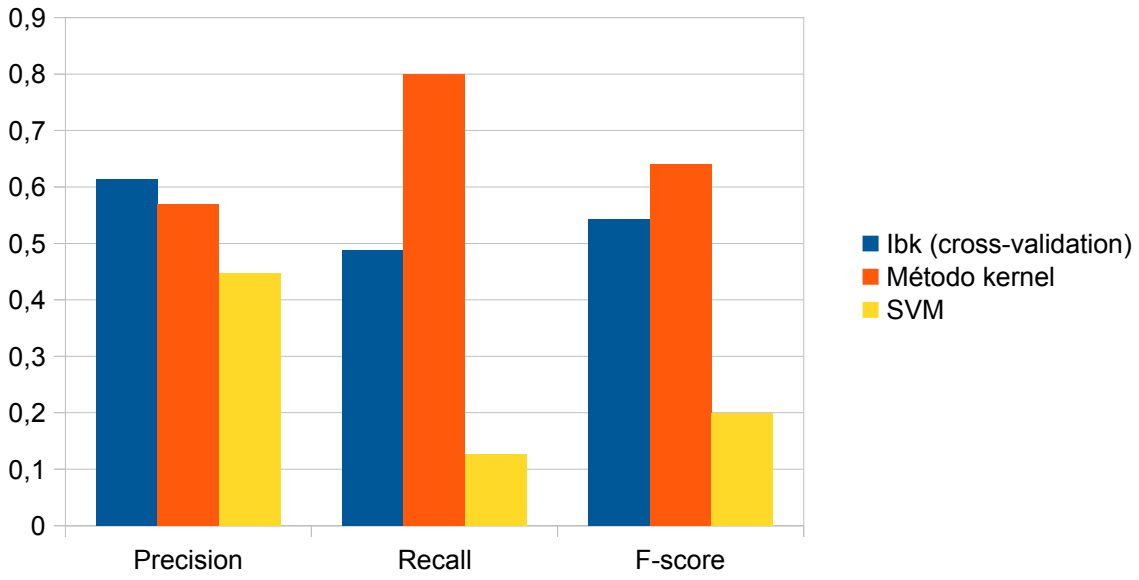


*Ilustración 14: Comparación Ibk-Método kernel [1]-SVM [4]*

En la Ilustración 14 podemos ver que el método kernel aplicado en [1] obtiene en general unos valores superiores respecto a los demás métodos. La única excepción es en la precisión, donde el experimento con Ibk con la evaluación *cross-validation* es el que tiene mejor resultado. En todas las medidas, SVM obtiene unos resultados muy bajos en comparación con el resto de métodos.

# Conclusiones y líneas de trabajo futuro

## ¿Qué objetivos se han logrado?

Si revisamos los objetivos listados al principio de la memoria, vemos que se han cumplido todos. En primer lugar se ha realizado un estudio previo, comprendiendo el proceso que se llevó a cabo en [1] y [4], ya que el presente proyecto se realiza a partir del trabajo realizado en ambos proyectos, utilizando el corpus DrugDDI y tomando como base el proyecto Druida.

En cuanto a la extracción de características, se ha realizado un procesamiento avanzado del corpus y se han obtenido diferentes tipos de atributos:

- Información básica de cada fármaco: palabra que lo representa, información morfosintáctica (*Part-of-Speech*), palabra cabecera, y combinaciones de las anteriores
- Secuencia de tokens entre los dos fármacos que participan en la interacción (N-gramas): bigramas y trigramas
- Infromación del camino entre los dos fármacos de la relación: palabras, PoS, información sintáctica y longitud del camino
- Contextos de cada fármaco: *before-between*, *between* y *fore-between*

Estos nuevos atributos han servido para entrenar diferentes clasificadores. Antes de ejecutar los experimentos se han aplicado algoritmos de selección de atributos sobre nuestros conjuntos de datos. Mediante estos algoritmos se filtran los atributos dejando sólo aquellos más relevantes para la clasificación. En concreto se han aplicado *Gain Ratio* y *Correlation based Feature Selection*, ambos elegidos en base al artículo [6].

A continuación, se han probado diferentes algoritmos de aprendizaje automático con la herramienta Weka para ver cuáles eran los que mejor resultado daban para utilizarlos en nuestros experimentos. Los elegidos han sido *Näive Bayes*, *LogitBoost*, *Hyperpipes* e *Ibk*. Con estos algoritmos se han realizado experimentos con diferentes técnicas de evaluación: *cross-validation* (utilizando 10 *folds*) y *training-test*. En el primer caso, el algoritmo que mejor resultado dio fue **Ibk**, con una medida-F de 0,543, mientras que con el método *trainnig-test* **Näive Bayes** obtuvo mejor medida-F: 0,447. Debido a estos resultados, se eligió *Ibk* para la posterior comparación, en concreto sus medidas obtenidas con *cross-validation*.

Una vez ejecutados los clasificadores y obtenidos los resultados, éstos se han comparado con los obtenidos mediante método kernel en [1], y con los obtenidos con el algoritmo SVM en [4]. En todos los experimentos se han comparado las medidas precisión, cobertura y medida-F.

Tal como se ha podido ver en la Tabla 10, los resultados con el método kernel aplicado en [1] son mejores que los obtenidos con técnicas de aprendizaje automático supervisado, habiendo evaluado ambos experimentos con *training-test*. La precisión de Ibk es mayor que la del método kernel, siendo sus valores 0,678 y 0,50 respectivamente. En cuanto a la cobertura y medida-F, los valores del método kernel son más altos. La cobertura en el método kernel es de 0,71 frente al valor 0,342 de Ibk, mientras que la medida-F es de 0,58 en el método kernel y 0,455 en Ibk.

En la Tabla 11 se vuelven a comparar Ibk y método kernel, pero esta vez evaluándolos con *cross-validation* con 10 *folds*. Los valores del método kernel vuelven a ser más altos que los de Ibk. Tal como ocurrió en la evaluación *training-test*, la precisión es más alta en Ibk que en el método kernel, con unos valores de 0,613 y 0,57 respectivamente. Sin embargo el valor de la cobertura es mucho más alto en el método kernel que en Ibk, siendo sus valores 0,8 y 0,487. Por último, la medida-F del método kernel, 0,640 es superior a la de Ibk, 0,543. En base a estos valores, se ha podido destacar que los valores de cobertura y medida-F han sido mejores con la evaluación *cross-validation* que con *training-test*.

En la Tabla 12 se han comparado los resultados de Ibk con los de SVM aplicado en [4], habiendo sido evaluados ambos con *cross-validation* con 10 *folds* en el caso de Ibk, y con 3 *folds* en SVM. En esta comparación los valores de Ibk han sido mejores que los de SVM, habiéndose obtenido unos valores de precisión de 0,613 y 0,447, una cobertura de 0,487 frente a 0,127, y una medida-F de 0,543 frente a 0,198.

Por último, en la Ilustración 14 se ha representado una comparativa de los tres métodos en los que se ve claramente que los resultados de [1] son más altos que los del resto de métodos. También se puede observar que los resultados de [4] son muy bajos comparados con el resto.

En el apartado siguiente se listan algunas mejoras y posibles líneas de trabajo futuro para poder mejorar el trabajo realizado en este proyecto.

## Trabajo futuro

Hay diferentes líneas de trabajo por las que se puede seguir investigando para mejorar los resultados obtenidos en el presente proyecto, o para ahondar en el motivo de los resultados.

En primer lugar se podría profundizar en el estudio de los algoritmos de clasificación. En este proyecto se han realizado experimentos con algunos clasificadores y se han mostrado los mejores resultados. Si se investiga profundamente cómo funcionan los algoritmos y con qué tipos de

características son realmente eficaces, podríamos mejorar su uso y posiblemente los resultados. Ya que en el proyecto hemos utilizado selectores de atributos basándonos en la literatura, también se podría mejorar la selección de caracterísiticas estudiando más profundamente cómo funcionan los selectores. Al igual que los clasificadores, requieren ciertas caracterísiticas para realizar eficazmente su labor.

Por otro lado se podrían analizar con mayor detalle los resultados obtenidos. Las curvas ROC y de precisión-recall pueden ayudarnos a entender mejor los resultados. En las curvas ROC hay que tener en cuenta dos métricas de rendimiento: el ratio de falsos positivos que es el porcentaje de ejemplos negativos clasificados erróneamente como positivos, y el ratio de verdaderos positivos, que mide el porcentaje de ejemplos positivos etiquetados correctamente. Las curvas ROC se generan trazando el ratio de verdaderos positivos en el eje vertical, y el ratio de falsos positivos en el eje horizontal. La curva ROC respresenta el rendimiento de un clasificador sin tener en cuenta la distribución de las clases ni los costes de errores. La representación se realiza de una manera intuitiva y robusta. De todas formas, hay que saber interpretarla con precaución ya que puede dar unos resultados muy optimistas del rendimiento de los clasificadores con conjuntos de datos muy desviados.

También se pueden analizar mejor los resultados mediante las curvas de precission-recall. Éstas son una alternativa a la curva ROC cuando hay una desviación grande en la distribución de las clases. En estas curvas el recall se representa en el eje horizontal y la precisión en el eje vertical. En nuestro caso, ya que los datos están muy desbalanceados, el análisis de estas curvas podría representar de mejor manera nuestros resultados.

Otra línea de trabajo futuro podría ser utilizar aplicar las pruebas de significación estadística de McNemar. Estas pruebas te permiten comparar diferentes clasificadores y determina si entre ellos difieren significantemente. De esta manera se puede determinar estadísticamente qué algoritmo es mejor.

Ya que en este proyecto se ve claramente que hay un gran desbalance en los datos, podría ser conveniente hacer más experimentos balanceando los datos. Tal como vimos en los experimentos, en los conjuntos de datos que se evalúan con validación cruzada la proporción de DDIs era de 11,62% frente a un 88,38% de instancias con fármacos que no interaccionan entre sí. En el caso de los conjuntos de datos de entrenamiento, el porcentaje de DDIs era de un 11,05% frente a un 88,95%. Por útlimo en los conjuntos de datos de evaluación, un 13,94% eran instancias de fármacos que interaccionaban mientras que un 86,06% correspondía a instancias sin interacción. En nuestro caso, la clase menos representada (DDI) es la de mayor importancia en nuestro problema. Los hechos de que el número de instancias con interacciones es muy bajo y de que muchos de los algoritmos de aprendizaje automático no pueden aprender de manera muy precisa con datos desbalanceados, pueden dar lugar a que los resultados no sean tan buenos

como deseáramos. Realizar experimentos con conjuntos de datos más balanceados podría ser una buena línea de trabajo a seguir ya que la proporción de las clases en las instancias podrían tener cierto impacto en los resultados. En la actualidad existen diversas técnicas para solucionar el problema del desbalance. Algunas de ellas con el *sampling*, donde se altera la distribución original de los datos, otros métodos consisten en adaptar las técnicas existentes de clasificación al problema del desbalance, tales como los métodos sensibles al coste (*cost-sensitive methods*), o incluso se pueden crear combinaciones de las técnicas de *sampling* (*hybrid sampling*).

Por último se podría realizar un análisis de los errores para ver qué partes de nuestro programa se podrían mejorar. Teniendo en cuenta el número de falsos positivos y negativos, deberemos buscar los motivos de estos errores. Una vez conocidas estas causas, podremos detectar los puntos débiles de nuestro programa, ya sea en los datos, o en los modelos de predicción.

# Bibliografía


1. **Segura Bedmar, Isabel**. Application of Information Extraction techniques to pharmacological domain: Extracting drug-drug interactions. Universidad Carlos III de Madrid, 2010.

2. **Sarawagi**. Information extraction. *Foundations and Trends® in Databases*, Vol. 1, pp. 261-377, 2007.

3. **Giuliano, A. Lavelli, and L. Romano**. Exploiting shallow linguistic information for relation extraction from biomedical literature. *Proceedings of the Eleventh Conference of the European Chapter of the Association for Computational Linguistics (EACL-2006)*, pp 5 - 7, 2006.

4. **Víctor Manuel Méndez González**. Un sistema para la extracción de interacciones farmacológicas basado en Support Vector Machines (SVM). Universidad Carlos III de Madrid, 2010.

5. **Bunescu and R. Mooney**. Subsequence kernels for relation extraction. Advances in Neural Information Processing Systems, 2006.

6. **Roxana Danger, Isabel Segura-Bedmar, Paloma Martínez, Paolo Rosso**. A comparison of machine learning techniques for detection of drug target articles. *Journal of Biomedical Informatics*, 2010.

# Presupuesto

A continuación se detalla el coste presupuestado para el presente proyecto.

## Cálculo de costes

### Descripción del proyecto

**Autor**: Beatriz Nombela Escobar

**Departamento**: Informática

**Título**: Aplicación de técnicas de aprendizaje automático para la extracción de información en textos farmacológicos

**Duración**: la fecha de inicio del proyecto es el **20 de septiembre de 2010** y la fecha de finalización es el **15 de julio de 2011**. En total se han invertido **380 horas** en la realización del presente proyecto.

### Costes de personal

El proyecto se ha realizado por una persona, la cual ha desarrollado todas las funcionalidades llevadas a cabo en el proyecto. A continuación se muestra el desglose de personal:

| Fase | Coste/hora | Total horas | Total coste |
|---|---|---|---|
| Análisis | € 25,00 | 40 | € 1.000,00 |
| Diseño | € 35,00 | 40 | € 1.400,00 |
| Codificación | € 45,00 | 100 | € 4.500,00 |
| Pruebas | € 30,00 | 80 | € 3.000,00 |
| Experimentación | € 30,00 | 50 | € 2.400,00 |
| Documentación | € 45,00 | 70 | € 3.150,00 |
| **Total** | | **380** | **€ 15. 450,00** |

*Tabla 13: Costes de personal*

### Costes de equipos

En este proyecto ha sido necesario un equipo durante la duración total del mismo. Aparte se ha utilizado otro equipo para la realización de experimentos. Este último equipo ha estado disponible durante un total de 6 meses.

| Concepto | Unidades | Total coste |
|---|---|---|
| Equipo general | 1 | € 400,00 |
| Equipo experimentación | 1 | € 150,00 |
| **Total** | | **€ 550,00** |

*Tabla 14: Costes de equipo*

### Costes de software

El coste del sistema operativo del equipo que se ha utilizado durante toda la duración del proyecto viene incluido en la Tabla 11, pues al estar incluído en el equipo sus costes se contabilizan en los Costes de equipo.

El sistema operativo del equipo de experimentación es una distribución Linux, que al ser gratuita no conlleva ningún coste.

El software utilizado en las pruebas y experimentación es Weka, y al ser una herramienta de código abierto tampoco conlleva ningún coste.

Para el desarrollo de la documentación se ha utlizado la suite gratuita LibreOffice.

En conclusión, no existen costes de software asociados al presente proyecto ya que todos los programas son de código abierto y por lo tanto gratuitos.

| Concepto | Unidades | Total coste |
|---|---|---|
| S.O. Equipo de experimentación | 1 | € 0,00 |
| Weka | 1 | € 0,00 |
| Suite LibreOffice | 1 | € 0,00 |
| **Total** | | **€ 0,00** |

*Tabla 15: Costes de software*

### Costes de material fungible

A continuación se muestran los costes asociados al material fungible de papel y cartuchos de tinta de impresora, y otros gastos no contemplados en los conceptos calculados previamente:

| Concepto | Total coste |
|---|---|
| Papel | € 23,00 |
| Tinta impresora | € 13,00 |
| Otros gastos | € 56,00 |
| **Total** | **€ 92,00** |

*Tabla 16: Costes de material fungible*

## Presupuesto

A continuación se listan todos los conceptos definidos anteriormente junto con el coste total del presupuesto:

| Concepto | Total coste |
|---|---|
| Coste de personal | € 15.450,00 |
| Coste de equipos | € 550,00 |
| Costes de software | € 0,00 |
| Costes de material fungible | € 92,00 |
| **Total** | **€ 16.092,00** |

*Tabla 17: Presupuesto total*

Por lo tanto el coste total presupuestado asciende a **DICESÉIS MIL NOVENTA Y DOS EUROS** (16.092,00 €), sin incluir el Impuesto sobre el Valor Añadido (I.V.A.).