Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Una aplicación de minería de datos para el análisis de la propiedad de terminación de SRTs

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Hermenegildo Fabregat Marcos

Tutor: Mª José Ramírez Quintana y Francisco Javier Piris Ruano

2014/2015

Resumen

Resumen

La minería de datos juega a día de hoy un papel importante en muchos campos de la ciencia debido a la gran cantidad de información con la que se trabaja diariamente y la que se puede deducir de la misma. Las posibilidades que ofrece la minería de datos son muy diversas, siendo además su ámbito de aplicación muy extenso. Uno de los posibles campos y del que versa este trabajo, es la predicción de propiedades de sistemas software. Para ser mas concretos, en este proyecto hemos planteado la posibilidad de predecir la propiedad de terminación de los sistemas de reescritura de términos.

Para ello se ha hecho uso de la base de datos de resultados de la *Termination Competition*, competición que se celebra anualmente desde 2006 y que trata sobre la demostración de la terminación de sistemas de reescritura. En esta base de datos se almacenan en forma de registros los resultados obtenidos por herramientas de terminación que participan en la competición, registros tales como el resultado de la demostración, tiempo empleado para obtener la respuesta y la traza de la demostración.

Palabras clave: sistemas de reescritura de términos, minería de datos, análisis estático de software.

Abstract

Nowadays data mining plays an important role in many science fields due to the huge volume of information that we handle on a daily basis as well as the information which can be obtained from it. Possibilities offered by data mining are very varied and its scope of application is very large. One of the possible fields we handle with in this project is the prediction of software system features. Specifically in this project we have researched the possibility of foreseeing the termination feature of term rewriting systems.

To achieve this goal the database containing the results of the *Termination Competition* taking place annually since 2006 has been used. Such database stores the results obtained by termination tools taking part in the competition, such as the result of the demonstration, time employed to obtain the result and the trace of such a demonstration.

Key words: term rewriting system, data mining, static software analysis.

Índice general

1. Introducción **1**

2. Preliminares y Conceptos básicos **5**
2.1. Sistemas de reescritura de términos 5
2.1.1. Propiedad de terminación 6
2.1.2. Sistemas automáticos para la demostración de la terminación 7
2.2. Minería de datos 9
2.2.1. Técnicas de minería de datos 10
2.2.2. Medidas de evaluación 15
2.2.3. Problemas de clasificación no balanceados 17

3. Metodología y Plan de Trabajo **19**
3.1. Metodología 19
3.2. Plan de Trabajo 21

4. Preparación de los datos **23**
4.1. Tarea 1: Extracción de los resultados de la competición 24
4.2. Tarea 2: Transformación a base de datos. 25
4.3. Tareas 3 y 4: Obtención de TRS y Conversión de .xml a .trs. 26
4.4. Tarea 5: Obtención de propiedades. 27
4.5. Tarea 6: Transformación a base de datos. 28
4.6. Análisis exploratorio de los datos 28
4.7. Relación de lenguajes y librerías usadas 30

5. Estudio experimental: Generación de los modelos **31**
5.1. Enfoque supervisado 32
5.2. Enfoque semi-supervisado. 36

6. Conclusión **39**

A. Código preparación de datos **41**
A.1. Tratamiento inicial de la base de datos. 41
A.2. Obtención de propiedades. 42
A.3. Inserción de propiedades en base de datos. 44
A.4. Arreglo de datos faltantes y inconsistencias. 45
A.5. Experimento R: Enfoque supervisado. 45
A.6. Experimento R: Enfoque semi-supervisado. 47

Índice de figuras

2.1. Ejemplo de TRS simple. 5
2.2. Ejemplo de TRS no terminante. 7
2.3. Interfaz web AProVE. 8
2.4. Interfaz web MU-TERM. 9
2.5. Proceso de descubrimiento de nuevo conocimiento. 10
2.6. Ejemplo de árbol de decisión. 12
2.7. Ejemplo de red neuronal. 13
2.8. Ejemplo de k-NN. 14
2.9. Ejemplo de SVM. 14
2.10. Validación cruzada. 16
2.11. *Tomek Links.* 17

3.1. Metodología seguida durante el estudio. 19
3.2. Plan de trabajo. 21

4.1. Diagrama de preparación de los datos. 23
4.2. Resultados herramientas. 26
4.3. Ejemplo de sistema de reescritura expresado en formato *TRS*. 26
4.4. Vista minable de la base de datos. 28
4.5. Estudio de la propiedad '*Linear*'. 29
4.6. Estudio de la propiedad '*Left Linearity*'. 29
4.7. Estudio de la propiedad '*Ground*'. 29
4.8. Estudio de la propiedad '*Duplicating*'. 29
4.9. Estudio de la propiedad '*Erasing*'. 29
4.10. Estudio de la propiedad '*Collapsing*'. 29

5.1. Balanceado de tres clases: *over-sampling*. 33
5.2. Balanceado de tres clases: *under-sampling*. 34
5.3. J48 SMOTE-TOMEK Enfoque supervisado - En forma de reglas. 35
5.4. Balanceado de dos clases y re-etiquetado del conjunto *MAYBE*. 37

Lista de tablas

2.1. Matriz de confusión 15

4.1. Herramientas de demostración. 24
4.2. Desglose de las categorías de TRS usadas en la competición. 25
4.3. Número de instancias. 28

5.1. Resultados enfoque supervisado, sin balanceado de clases. 32
5.2. Enfoque supervisado - J48: Matriz de confusión. 33
5.3. Resultados enfoque supervisado, estudio completo. 34
5.4. Enfoque supervisado - J48 SMOTE-TOMEK: Matriz de confusión. 36
5.5. Enfoque supervisado - J48 SMOTE-ENN: Matriz de confusión. 36
5.6. Resultados enfoque semi-supervisado, estudio completo. 37
5.7. Enfoque semi-supervisado dos clases - J48 SMOTE-TOMEK: Matriz de confusión. 38
5.8. Enfoque semi-supervisado re-etiquetado - J48 SMOTE-TOMEK: Matriz de confusión. 38

Capítulo 1

Introducción

A día de hoy existen multitud de actividades de la vida cotidiana que constatan cómo la tecnología ha mejorado nuestra calidad de vida y junto a ella nuestra forma de ver las cosas que la componen. Vivimos en una sociedad masificada tecnológicamente, donde los objetos más simples se han convertido en pequeños sistemas informáticos conectados con un ente más grande y complejo. Con el tiempo hemos adquirido unas rutinas que no conciben la posibilidad de un funcionamiento incorrecto de los sistemas informáticos. Esta forma de ver la tecnología, dependiendo siempre del contexto puede conllevar desde un simple reinicio del sistema en cuestión, hasta perder grandes sumas de dinero y en el peor de los casos (entornos críticos) podemos llegar a hablar incluso de la perdida de vidas. Con esto no se intenta decir que nadie revisa los programas o sistemas, ni mucho menos, es más de no ser revisados estaríamos hablando bajo otro punto de vista, pero a lo mejor sí que es cierto que no son revisados con la exhaustividad necesaria para asegurar sin ninguna duda su correcto funcionamiento a lo largo del tiempo y en cualquier contexto. Con motivo de poder dotar de una confianza bien fundada a los programas es necesario analizarlos con la exhaustividad antes mencionada. Existe un área de la informática encargada de este aspecto, es decir del desarrollo de métodos y técnicas de análisis de software. En general, hay dos enfoques posibles: el *análisis estático* y el *análisis dinámico*. La principal diferencia entre ambos es que el análisis dinámico necesita la ejecución de los programas para analizarlos y el estático no [11].

Una de las propiedades mas importantes dentro del análisis estático es la terminación de los programas. Este problema, también conocido como el problema de la parada, puede ser definido de la siguiente forma:

Haciendo uso de una cantidad de tiempo finita, determinar para cualquier programa si siempre terminara su ejecución o podría darse el caso de mantenerse en ejecución eternamente.

En resumen, determinar esta propiedad es un problema muy interesante a la vez que complejo que proporcionaría una vez resuelto, la capacidad de poder afirmar que un programa va a terminar en algún momento sin necesidad de ejecutarlo.

Los resultados de los estudios de Turing [28] fueron muy importantes para la de-

mostración de la indecidibilidad del problema de la terminación[1], siendo sus resultados portada de muchos trabajos y cursos en el area de la lógica y la teoría de computadores. Sin embargo, la popularización del trabajo de Turing tuvo cómo efecto negativo que se mantuviera la idea de que es imposible determinar la terminación de ningún programa, siendo cierto que no es posible determinar la terminación de todos los programas pero si de algunos [7]. Ratificando la proposición anterior, existen muchos trabajos sobre la terminación de los programas donde se demuestra la terminación de sistemas bajo ciertas condiciones. Por este motivo, frases como 'Pero eso es como el problema de la terminación' han sido utilizadas en discusiones donde se comentaban soluciones parciales a problemas planteados como indecidibles.

Una forma de probar la terminación de los programas escritos en un lenguaje de programación declarativo cómo Haskell, Maude o Prolog consiste en convertirlo en un sistema de reescritura de términos (TRS, *Term Rewriting System*) [20] y usar técnicas y herramientas automáticas para la demostración de la terminación sobre ellos. Estas técnicas y herramientas devuelven 'quizás' cuando, después de realizar todas las pruebas, no han sido capaces de llegar a una conclusión acerca de si un sistema de reescritura de términos es o no terminante [19][16].

La reescritura de términos es una rama teórica de la computación que está basada en la lógica ecuacional. En la tercera y cuarta década del siglo 20, las técnicas de reescritura fueron usadas en matemáticas y lógica matemática como un *framework* para analizar computacionalmente propiedades como la confluencia y la terminación de sistemas λ-*calculus* y de lógica combinatoria. λ-*calculus* es probablemente el sistema de reescritura de términos mas conocido, éste tuvo un rol crucial para formalizar la noción de computabilidad.

Los primeros intentos de demostración de la terminación de sistemas de reescritura datan del año 1970 y no es hasta el año 1997 cuando se implementa una demostración basada en pares de dependencias, la cual significa un gran avance ya que logra demostrar la terminación de muchos mas sistemas de reescritura que ninguna otra técnica usada hasta ese momento. Este hecho animó a muchos investigadores a seguir trabajando en este campo. En el año 2003, Albert Rubio promovió un evento especial para comparar las distintas herramientas diseñadas hasta el momento. Motivados todos los investigadores de este campo por los resultados obtenidos de la comparación de las herramientas, un año mas tarde se decidió crear la *Termination Competition* donde cada año varios grupos de investigadores de distintos países comparan sus *demostradores* e intentan demostrar la terminación sobre la mayor cantidad de programas como les sea posible [21]. Un apartado importante de la competición lo constituyen los sistemas de reescritura de términos.

Como consecuencia de esta competición se ha ido acumulando una gran cantidad de información que quizás podría aportar un enfoque distinto a lo ya estudiado; estamos en una época donde el gran volumen de información con el que se trabaja diariamente nos permite analizar y comprender los datos desde un punto de vista mas complejo. La ciencia encargada de este proceso es la minería de datos.

La minería de datos es un campo que aporta métodos y técnicas para el tratamiento

[1]Existe cierta controversia sobre la demostración de la indecidibilidad de la terminación por parte de Turing. Técnicamente el no la demostró, pero si que es consecuencia directa de sus resultados.

y análisis de grandes conjuntos de datos. Este campo tiene conexión con diversas áreas cómo son la recuperación de la información, la estadística y la inteligencia artificial; ligada sobre todo a este último campo pues es de el de donde toma muchas de las técnicas de aprendizaje automático que usa.

El objetivo de éste proyecto no es otro que estudiar y analizar la terminación de los sistemas de reescritura de términos de forma automática, aplicando técnicas de minería de datos sobre la información generada en las distintas ediciones de la *Termination Competition*. El éxito de la minería de datos reside en parte en la calidad de los datos sobre los que se aplican las técnicas que generan los modelos o patrones. De hecho, se entiende que la recopilación y procesamiento de los datos es la etapa que mas tiempo consume. En este proyecto ademas nos enfrentamos al problema de decidir de qué forma podemos representar los sistemas de reescritura de términos, los cuales necesitan de un formato adecuado para poder aplicar las técnicas de aprendizaje automático. Hasta nuestro conocimiento no hay ningún estudio similar al nuestro y los primeros resultados del mismo se exponen en el articulo *Analysing the Termination of Term Rewriting Systems using Data Mining* pendiente de publicación.

Esta memoria se ha organizado de la siguiente forma. El capítulo dos introduce los conceptos previos necesarios para comprender el ámbito de este proyecto, siendo estos la minería de datos, los sistemas de reescritura de términos y la propiedad de terminación. El capítulo tres describe tanto la metodología como el plan de trabajo que se ha seguido. El capítulo cuatro muestra con detalle el proceso de obtención y preparación de los datos para poder ser analizados mediante la aplicación de técnicas de aprendizaje automático. Además, en este capítulo se incluye un estudio de los datos obtenidos con el fin de realizar un análisis a priori de los datos. El siguiente capítulo presenta el estudio experimental realizado y los resultados obtenidos. Finalmente, el último capítulo resume las contribuciones del proyecto y expone el trabajo futuro.

Capítulo 2

Preliminares y Conceptos básicos

En este capítulo se incluyen los conceptos básicos sobre sistemas de reescritura de términos, terminación y minería de datos; los cuales se utilizaran durante el resto de la memoria. Para una información mas detallada acerca de los sistemas de reescritura de términos y sobre técnicas de minería de datos se puede consultar *Handbook of automated reasoning* [26] y *Introducción a la minería de datos* [15], respectivamente.

2.1. Sistemas de reescritura de términos

En general, la reescritura de términos se aplica en cualquier contexto donde se requieren métodos eficientes para el razonamiento con ecuaciones, desde el diseño de lenguajes de programación hasta el análisis e implementación de especificaciones abstractas e. g. tipos de datos abstractos especificados ecuacionalmente [16][2].

Los sistemas de reescritura de términos son interesantes ya que, su simple sintaxis y clara semántica facilita enormemente su análisis matemático y además ellos constituyen un modelo computacional Turing Completo el cual queda muy ligado a la programación funcional. Por otra parte, también proporcionan de un medio natural para implementar procesos computacionales y en un principio incluso computación paralela.

$$
\begin{aligned}
&\mathrm{add}(0, \mathrm{x}) \to \mathrm{x},\\
&\mathrm{add}(\mathrm{s}(\mathrm{x}), \mathrm{y}) \to \mathrm{s}(\mathrm{add}(\mathrm{x}, \mathrm{y})),\\
&\mathrm{prod}(0, \mathrm{x}) \to 0,\\
&\mathrm{prod}(\mathrm{s}(\mathrm{x}), \mathrm{y}) \to \mathrm{add}(\mathrm{y}, \mathrm{prod}(\mathrm{x}, \mathrm{y})),\\
&\mathrm{fact}(0) \to \mathrm{s}(0),\\
&\mathrm{fact}(\mathrm{s}(\mathrm{x})) \to \mathrm{prod}(\mathrm{s}(\mathrm{x}), \mathrm{fact}(\mathrm{x})).
\end{aligned}
$$

Figura 2.1: Ejemplo de TRS simple.

Un ejemplo de sistema de reescritura de términos se muestra en la figura 2.1. Este TRS define la operación de suma (add), producto (prod) y factorial (fact) de números naturales expresadas en notación de Peano (notación en la que los números se construyen

a partir de la constante 0 y de la función unaria s que representa a la función sucesor). Como se puede observar, una regla de reescritura es una ecuación orientada de izquierda a derecha, es decir, la ecuación $l = r$ se corresponde a la regla de reescritura $l \to r$, siendo l y r términos. Por consiguiente, es fácil transformar un conjunto de ecuaciones en un sistema de reescritura.

Mas formalmente, un sistema de reescritura de términos es un par (Σ, R) donde R es un conjunto de reglas de reescritura y Σ es la *signature*, es decir el conjunto de símbolos de función $(f_1, f_2, ...)$ junto con su correspondiente aridad (información que establece el número de parámetros asociados a cada símbolo de función). Además, una regla de reescritura es un par ordenado (l, r), representado como $l \to r$, donde l y r son términos tales que l no es una variable, y todas las variables que ocurren en r también ocurren en l.

Una ejecución consiste en una secuencia encadenada de reducciones a partir de una expresión inicial (un término), donde una reducción se define cómo la sustitución en el término a reducir de una instancia de una parte izquierda de una regla por la correspondiente instancia de su parte derecha. Cuando una expresión no se puede reducir mas se dice que está en *forma normal*.

A la hora de leer un sistema de reescritura de términos y de empezar a realizar reducciones hay que tener claro un punto muy importante, la estrategia a seguir. Lenguajes como Haskell con evaluación perezosa nos dan una buena visión de cuan importante es por donde se ha de empezar a reducir un término para no dar lugar a secuencias infinitas de computación. La evaluación perezosa se corresponde en terminología de TRS a la estrategia *Outermost*, donde primero evaluamos el sub-término reducible mas externo. Por otro lado, *Innermost* es la estrategia de reducción donde primero se reducen lo miembros mas internos del término, un punto de vista mas clásico que genera una traza mas fácil de leer. Por último, también existe la estrategia *Full*, la cual no es una mezcla de las dos anteriores si no la aplicación de las dos obteniendo una traza de ejecución en forma de árbol, es decir, el árbol tendría en las hojas todas las reescrituras posibles para el término inicial aplicando todas la reglas que hicieran *matching*[1] en cualquier sub-término del término evaluado.

2.1.1. Propiedad de terminación

Un sistema de reescritura de términos se considera terminante cuando, dada una expresión cualquiera, su secuencia de ejecución es finita. La terminación es un concepto importante en la reescritura de términos pues permite garantizar que las computaciones son finitas [2]. Esto quiere decir que un sistema de reescritura de términos es terminante si aplicando un número finito de pasos de reducción se puede encontrar la forma normal de una expresión dada. Aunque visto el ejemplo expuesto en la figura 2.1, este problema puede parecer fácil de resolver, la evaluación de esta propiedad resulta un proceso no tan trivial cuando por ejemplo tratamos con sistemas donde intervienen muchas reglas. Un ejemplo de un sistema no terminante puede ser el mostrado en la figura 2.2, que expresa

[1] Cuando un término es una instancia de una parte izquierda de una regla, decimos que término y la regla se emparejan (*matching*).

la conmutatividad de la suma.

$$\mathrm{u} + \mathrm{v} \rightarrow \mathrm{v} + \mathrm{u}.$$

Figura 2.2: Ejemplo de TRS no terminante.

Aunque en las últimas décadas se han desarrollado muchas técnicas y métodos para probar la terminación de los sistemas de reescritura de términos, aún a día de hoy, este problema sigue siendo un tema de discusión abierto.
Un criterio fundamental en la determinación de la terminación es disponer de un orden bien fundado sobre los términos. Criterio definido en primera instancia por Manna y Ness, y redefinido por Lankford [9]:

Teorema *well-founded ordering* o teorema de Lankford. Un TRS (Σ, R) es terminante si y solo si hay una relación de orden $>$ sobre los términos tal que $l > r$ para todo $l \rightarrow r \in R$.

Este teorema ha motivado diversos enfoques para demostrar la terminación de los TRS desde el año 1995 hasta la actualidad. No es objeto de este proyecto entrar en detalle en conceptos, propiedades o demostraciones relacionadas con los TRS, tán sólo apuntar sobre la propiedad de terminación que aunque existen numerosas reglas o características de los sistemas de reescritura que són fácilmente decidibles, ésta no es una de ellas. Un documento muy completo que detalla muchas de las propiedades de los TRS junto a algunas técnicas de demostración de terminación y del cual se ha extraído la información de la cual se compone este apartado es *Termination and confluence properties of structured rewrite systems* [14].

2.1.2. Sistemas automáticos para la demostración de la terminación

Como hemos mencionado, a lo largo de los años se han desarrollado muchas técnicas para la demostración de la terminación bajo ciertas condiciones y el enfoque por el cual se optó a partir del año 2004, estimulados en gran parte por el planteamiento de la *Termination Competition*, fue la búsqueda de un proceso automático de demostración de la terminación. Hasta la fecha son muchas las herramientas que han desarrollado diversos grupos de investigación, los cuales participan cada año en la *Termination Competition*, mostrando los avances conseguidos hasta la fecha. Estos avances incluyen mejoras en las implementaciones siempre intentando incrementar la eficiencia y eficacia de estos sistemas. Algunas de las técnicas implementadas por estas herramientas son:

- Interpretación polinomial [7].
- *Well-founded ordering* [27].
- Etiquetado semántico [31].
- Pares de dependencias [1].

En resumen, una herramienta para la demostración automática de la terminación de sistemas de reescritura de términos es la implementación de las distintas técnicas de demostración de terminación desarrolladas. A estas herramientas también se las denomina *demostradores* debido a su enfoque automático.

A continuación se muestran varias de las herramientas que participan en la competición junto a la descripción aportada desde su página web oficial[2] [3].

AProVE [13] Desarrollada por RWTH Aachen en Alemania, AProVE es una herramienta para la demostración de terminación y complejidad tanto de sistemas de reescritura de términos como de numerosas variaciones de los mismos. AProVE maneja multitud de formalismos además de sistemas de reescritura, como por ejemplo, programas imperativos (Java Bytecode), programas funcionales (Haskell 98) y programas lógicos (Prolog). En el año 2014 el grupo de desarrollo de esta herramienta recibió el primer premio en la competición convirtiéndose en un referente de la misma. Tiene disponible en su web oficial una versión online (figura 2.3).

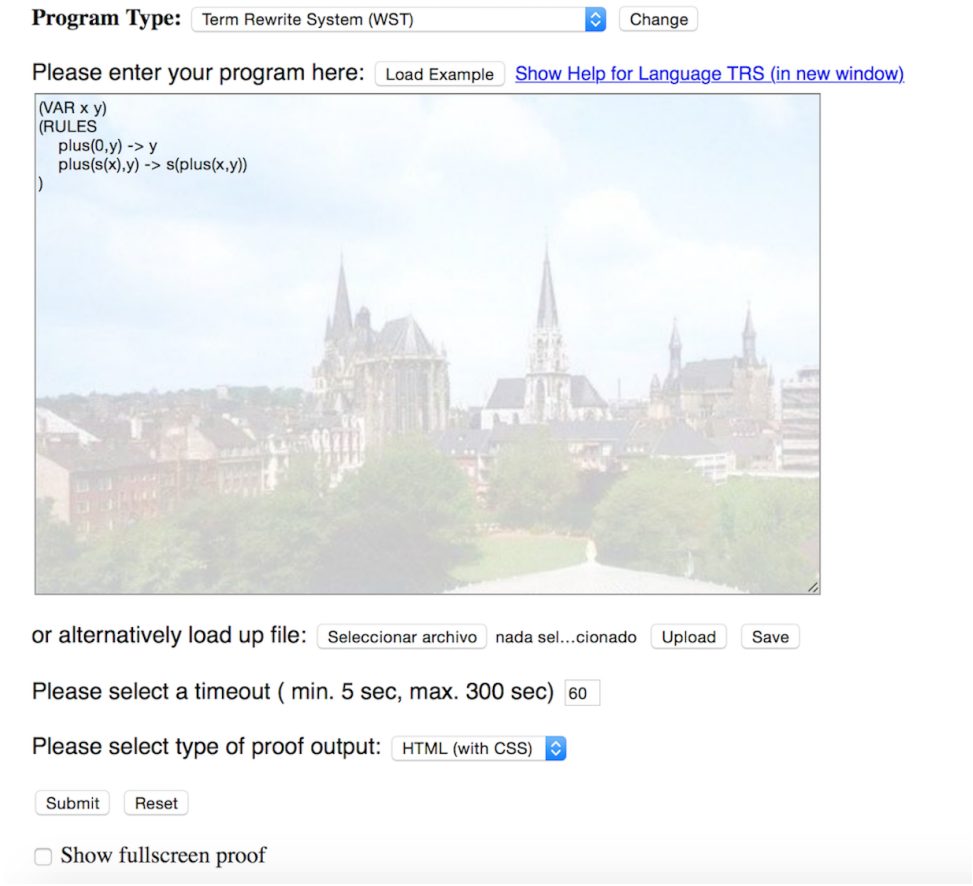


Figura 2.3: Interfaz web AProVE.

MU-TERM Es una herramienta desarrollada por el grupo de Extensiones de la Programación Lógica del Departamento de Sistemas Informáticos y Computación de la Universidad Politécnica de Valencia. En un principio fue concebida para demostrar

[2] Mu-Term: http://zenon.dsic.upv.es/muterm/
[3] AProVE: http://aprove.informatik.rwth-aachen.de

la terminación de sistemas de reescritura dependientes del contexto (CSR) pero a día de hoy trabaja con muchos otros tipos de sistemas de reescritura. MU-TERM 5.0 esta compuesto de 47 módulos programados en Haskell con mas de 19000 líneas de código. Al igual que para AProVE, además de versiones compiladas para distintos sistemas operativos, los desarrolladores de MU-TERM ofrecen una interfaz web (figura 2.4) completamente funcional. La versión actual de MU-TERM es la 5.13, lanzada en Julio de 2014.

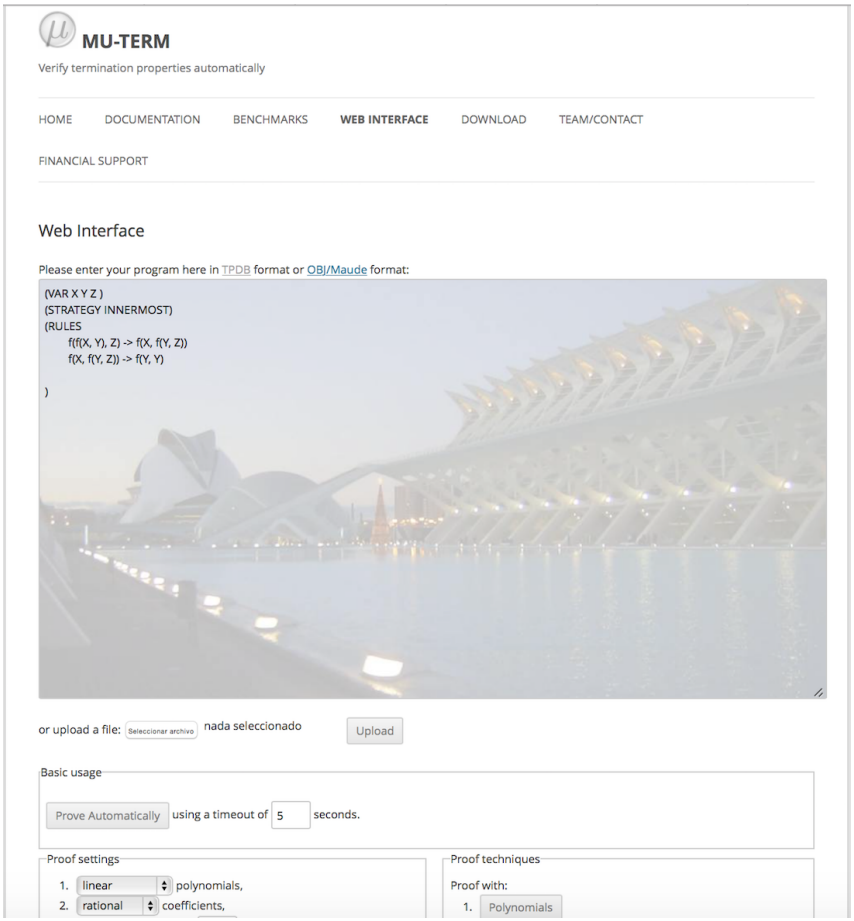


Figura 2.4: Interfaz web MU-TERM.

2.2. Minería de datos

La mayor parte de la información tanto de esta sección como de sub-secciones de la misma ha sido extraída del libro *Introducción a la minería de datos* [15], cualquier otra fuente quedara referenciada de igual modo.

La minería de datos puede definirse inicialmente como un proceso de descubrimiento de nuevas y significativas relaciones o tendencias además de patrones al examinar grandes cantidades de información. Esta disciplina surge debido a la aparición de nuevas necesidades y por el reconocimiento de un nuevo potencial, el valor de la gran cantidad de información con la que trabajan los sistemas. Una definición de este término, introducida en el libro *Data Mining: Practical machine learning tools and techniques* [30], resume la

minería de datos como el proceso de extraer conocimiento útil y comprensible previamente desconocido, de grandes cantidades de datos almacenados en distintos repositorios y en distintos formatos. Desde sus inicios, la minería de datos se enfrenta a dos retos: en primer lugar, el procesado y análisis de grandes volúmenes de información y, en segundo lugar, la posterior extracción del conocimiento.
Para que el uso de la minería de datos sea efectivo, las técnicas usadas para el procesamiento de la información y la creación de los modelos deben permitir un enfoque automático o al menos semi-automático (asistido). Para llevar a cabo su cometido, la minería de datos necesita que los datos estén expresados en una tabla que se denomina *vista minable*, siguiendo una notación 'atributo - valor'.

Existe cierta confusión entre el significado de minería de datos y KDD (*Knowledge Discovery in Databases*), la minería de datos es sólo una etapa del proceso KDD. Un esquema general del proceso de KDD se muestra en la figura 2.5. En este esquema distinguimos las siguientes etapas:

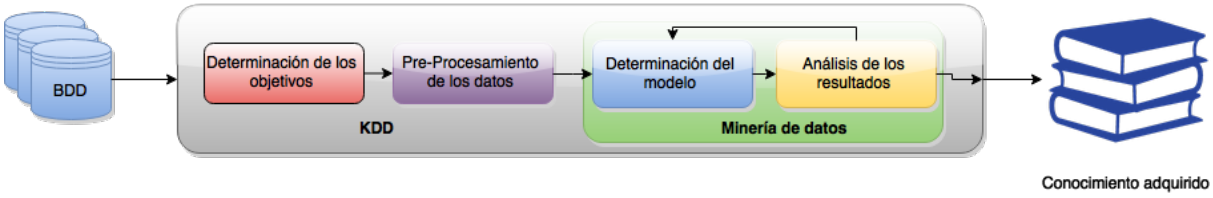
Determinación de los objetivos. Es la etapa de puesta en común entre los deseos del cliente y la opinión del especialista.

Pre-procesamiento de los datos. Se refiere a la selección, limpieza, mejora y transformación de la información contenida en las bases de datos.

Determinación del modelo. Esta es la etapa en la que se centra la minería de datos. Se entiende por modelo el resultado de la aplicación de diferentes algoritmos para la generación de nuevo conocimiento, algunos de ellos provenientes de áreas como la inteligencia artificial.

Análisis de los resultados. Verificación y análisis mediante métodos gráficos y estadísticos de la coherencia de los resultados obtenidos de la aplicación del modelo. A partir de este análisis se determina si el nuevo conocimiento es relevante o no.

Figura 2.5: Proceso de descubrimiento de nuevo conocimiento.



2.2.1. Técnicas de minería de datos

Dentro de la minería de datos hemos de distinguir varios tipos de tareas, cada una de las cuales puede considerarse como un tipo de problema a ser resuelto por un algoritmo de minería de datos. La distintas tareas pueden ser predictivas o descriptivas. Mientras que las tareas predictivas tratan de predecir un valor de clase para una instancia mediante la concordancia con el resto de instancias, las descriptivas tratan de describir los datos en base a similitudes entre las distintas instancias.

- Tareas predictivas

  La clasificación. Es quizás la tarea mas utilizada. En ella, cada instancia (o registro de la base de datos) pertenece a una clase, la cual se indica mediante el valor de un atributo el cual llamamos la clase de la instancia. Este atributo puede tomar diferentes valores discretos. El objetivo es predecir la clase de nuevas instancias a partir del resto de atributos.

  La regresión. Es también una tarea predictiva que consiste en aprender una función real que asigna a cada instancia un valor real. La principal diferencia respecto a la clasificación es que el valor a predecir es numérico.

- Tareas descriptivas

  El agrupamiento (*clustering*). Es la tarea descriptiva por excelencia y consiste en obtener grupos 'naturales' a partir de los datos maximizando la similitud entre las instancias de un mismo grupo y minimizando la similitud entre las instancias de distintos grupos.

  Las correlaciones. Es una tarea descriptiva que se utiliza para examinar el grado de similitud de los valores de dos variables numéricas.

  Las reglas de asociación. Es también una tarea descriptiva, muy similar a las correlaciones, que tiene como objetivo identificar relaciones no explícitas entre atributos categóricos.

  Cuando todas las instancias que se usan para entrenar los modelos están etiquetadas con un valor se dice que el aprendizaje es supervisado. Sin embargo, esto no siempre es necesario ya que en algunos problemas tener datos etiquetados puede ser muy costoso o simplemente las etiquetadas no son conocidas. Este campo es conocido como aprendizaje semi-supervisado.

Dado que la minería de datos es un campo interdisciplinar, existen diferentes paradigmas detrás de las técnicas utilizadas para este sub-proceso: técnicas de inferencia estadística, árboles de decisión, redes neuronales, inducción de reglas, aprendizaje basado en instancias, algoritmos genéticos, aprendizaje bayesiano, programación lógica inductiva y varios tipos de métodos basado en núcleos (*Kernels*), entre otros. Cada una de estas técnicas incluye diferentes algoritmos y variaciones de los mismos, así como otro tipo de restricciones que hacen que la efectividad del algoritmo dependa del dominio de la aplicación, no existiendo lo que podríamos llamar el método universal aplicable a todo tipo de aplicación. A continuación se describen los algoritmos de aprendizaje usados en este proyecto.

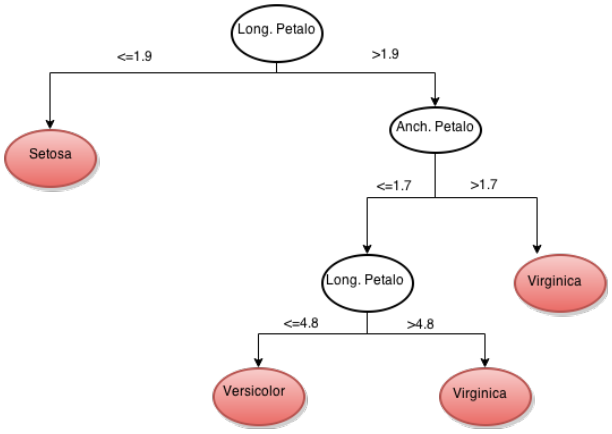
Árbol de decisión

De todos los métodos de aprendizaje predictivos, los sistemas de aprendizaje basados en árboles de decisión son quizás los métodos mas fáciles de utilizar y de entender. Un árbol de decisión es un conjunto de condiciones organizadas en una estructura jerárquica, de manera que la decisión final a tomar se puede determinar siguiendo las condiciones que se cumplen desde la raíz del árbol hasta alguna de sus hojas. Estas condiciones se corresponden con *tests* sobre los atributos. La tarea de aprendizaje para la cual los

árboles de decisión se adecuan mejor es la clasificación. De hecho, clasificar es determinar entre varias clases a qué clase pertenece una instancia, y la estructura de condición y ramificación de un árbol de decisión es idónea para este problema y además facilita mucho el análisis a posteriori. Debido al hecho de que la clasificación trata con clases o etiquetas disjuntas, un árbol de decisión conducirá a un ejemplo hasta una única hoja, asignando por tanto, una única clase al ejemplo. Para ello, las particiones existentes en el árbol deben ser también disjuntas. Es decir, cada instancia cumple o no cumple una condición, siendo además esta propiedad exhaustiva, significando que una condición entre las expresadas debe cumplirse.

La figura 2.6 muestra un ejemplo de árbol de decisión para un problema de clasificación de flores tipo iris[4]. Este problema consta de tres clases, *setosa*, *virgínica* y *versicolor*. Como se puede apreciar, las instancias del problema se clasificarán en función de los *tests* que se producen en los nodos del árbol hasta llegar a un nodo hoja. Una de las ventajas de este algoritmo es que el árbol puede representarse cómo un conjunto de reglas, una por cada rama del árbol.

Figura 2.6: Ejemplo de árbol de decisión.



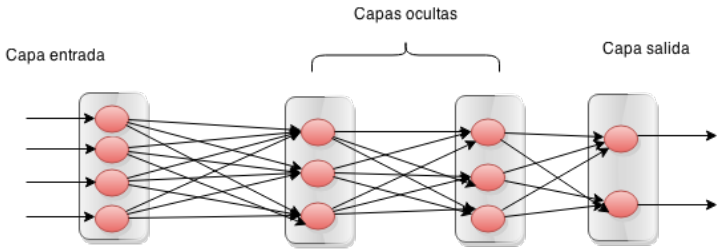
Los algoritmos de aprendizaje basados en árboles de decisión obtienen un modelo que es completo y consistente con respecto a la evidencia. Es decir, el modelo cubre todos los ejemplos vistos y todos de manera correcta. Esto puede parecer óptimo a primera vista, pero se vuelve demasiado ingenuo en la realidad. Intentar ajustar demasiado hace que el modelo sea poco general y se adapte mal a instancias no vistas. De hecho, esto queda especialmente patente cuando la evidencia contiene ruido (errores en los atributos o incluso en las clases), ya que el modelo intentará ajustarse a los errores y esto perjudicará al comportamiento global del modelo aprendido. El proceso mediante el cual se consigue evitar el sobre-ajuste en los modelos obtenidos mediante árboles de decisión se denomina poda. La poda funciona como un límite, quedando de tal forma que los nodos que están por debajo del límite de poda se eliminan, ya que se consideran demasiado específicos o, dicho de una manera mas informal, se consideran demasiado *adhoc*.

[4]https://archive.ics.uci.edu/ml/datasets/Iris

Redes neuronales artificiales

Las redes neuronales artificiales o RNA, son un método de aprendizaje cuya finalidad inicial es la de emular los procesadores biológicos de información. Podemos decir que las RNAs son sistemas de procesamiento de la información adaptativos, distribuidos y paralelos, que desarrollan su funcionalidad en respuesta a la información disponible para la red. Las RNAs parten de la presunción de que la capacidad humana de procesar información se debe a la naturaleza biológica de nuestro cerebro i. e. son capaces de aprender de la experiencia, de generalizar casos anteriores a casos nuevos, de abstraer características esenciales de la información disponible y son tolerantes a fallos (en el sentido de que las RNAs aprenden a reconocer patrones con ruido, distorsionados o incompletos). Las redes neuronales artificiales con capas ocultas se aplican especialmente cuando los comportamientos no lineales son importantes. En la figura 2.7 se muestra un ejemplo de red neuronal con cuatro entradas, dos capas ocultas cada una de ellas con tres neuronas y una capa de salida con dos neuronas de salida.

Figura 2.7: Ejemplo de red neuronal.

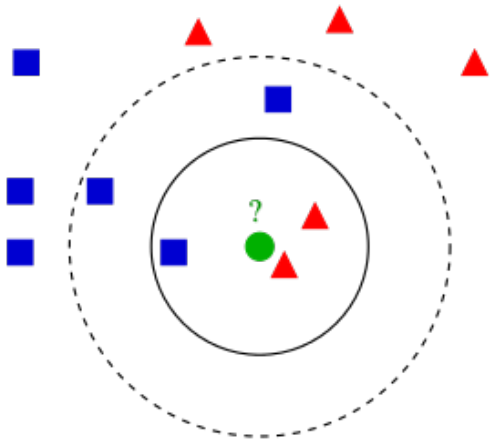


k-NN (*K Nearest Neighbours)*

Los métodos de aprendizaje automático basados en casos y vecindad tratan de resolver un problema a partir de información extraída de un conjunto de ejemplos existentes previamente. Los ejemplos son los que aportaran la información necesaria para poder predecir el comportamiento de un nuevo dato no perteneciente al conjunto utilizado para la generación del modelo. Un ejemplo de método de aprendizaje basado en vecindad es la regla del vecino mas próximo (1-NN), que consiste en clasificar cada ejemplo no visto con el valor de la clase mas próxima de acuerdo a una función de distancia. Esta regla, conocida como 1-NN, tiene bastantes problemas, ya que ignora la densidad o la región donde se encuentra el ejemplo. Una variante de este método son los k vecinos mas próximos o k-NN, en el que se asigna la clase a la que pertenece la instancia en función de la distancia entre los k vecinos mas próximos.

En resumen, k-NN estima el valor de la función de densidad de probabilidad o directamente la probabilidad a posteriori de que un elemento x pertenezca a la clase C_j a partir de la información proporcionada por el conjunto de ejemplos mas próximos. La figura 2.8 muestra un ejemplo gráfico de clasificación en función de los vecinos mas cercanos. La nueva instancia (el círculo) se clasificará cómo triángulo si tenemos en cuenta los tres vecinos mas próximos o cómo un cuadrado si usamos los cinco vecinos mas próximos.

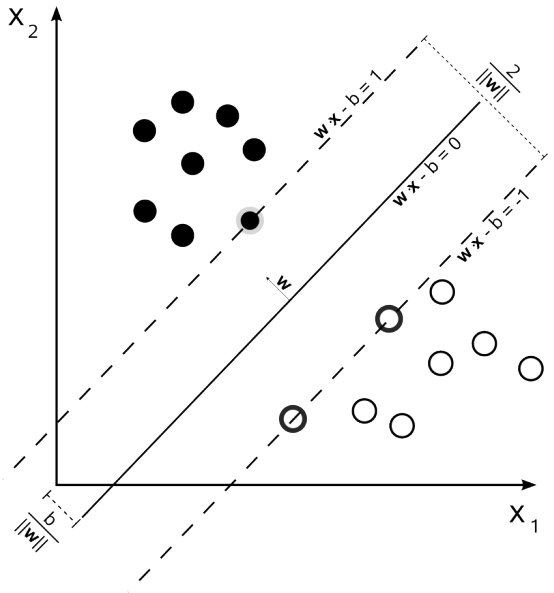
Figura 2.8: Ejemplo de k-NN.



Máquinas de vectores soporte

Las máquinas de vectores soporte o SVM, pertenecen a la familia de los clasificadores lineales puesto que inducen separadores lineales o hiperplanos en *espacios de características* de muy alta dimensionalidad, introducidos por funciones núcleo o *kernel* (ver figura 2.9). El aprendizaje de separadores no lineales con SVM se consigue mediante una transformación no lineal del espacio de atributos de entrada en un *espacio de características* de dimensionalidad mucho mayor y donde sí es posible separar linealmente los ejemplos. El uso de las denominadas funciones núcleo, que calculan el producto escalar de dos vectores en el *espacio de características*, permite trabajar de manera eficiente en el *espacio de características* sin necesidad de calcular explícitamente las transformaciones de los ejemplos de aprendizaje. Es un requisito básico para aplicar con éxito SVM elegir correctamente la función núcleo adecuada, la cual debe reflejar el conocimiento a priori sobre el problema [4][15].

Figura 2.9: Ejemplo de SVM.



2.2.2. Medidas de evaluación

La evaluación del modelo no deja de ser una parte mas del proceso de KDD. Su valía reside en que nos ayuda a encontrar el mejor modelo capaz de representar nuestros datos y el que mejor se adaptará a datos futuros. Existen muchas formas de analizar los clasificadores. Las medidas mas usadas hacen uso de la matriz de confusión, que muestra la cantidad de clasificaciones correctas y la cantidad de incorrectas obtenidas por el modelo de clasificación a evaluar. La matriz es N*N, donde N es el número de posibles valores de clase. El rendimiento es normalmente evaluado usando los datos provenientes de la matriz. La siguiente tabla muestra una matriz de confusión de tamaño 2*2 para dos clases (Positiva y Negativa).

Tabla 2.1: Matriz de confusión

Matriz de confusión		**Real**		**Precisión**
		Positiva	**Negativa**	
Estimado	**Positiva**	True Positive (TP)	False Positive (FP)	TP/(TP+FP)
	Negativa	False Negative (FN)	True Negative (TN)	TN/(FP+TN)
		TP rate	**RN rate**	(TP+TN) /
		TP/(TP+FN)	TN/(FP+TN)	(TP+FP+TN+FN)

Algunas de las medidas mas usadas que se pueden obtener a partir de la matriz de confusión y que tienen mayor importancia con respecto al estudio de problemas de clasificación son las que se describen a continuación.

Precisión. Es la medida mas utilizada para la evaluación de los modelos de predicción y por tanto también la mas criticada. Esta se define como el porcentaje de aciertos en la predicción sobre el total de predicciones realizadas. Una de la ventajas que tiene es su simplicidad, aunque también resulta un inconveniente por no tener en cuenta otros factores además del número de aciertos. Esta medida no tiene en cuenta hechos tales como una distribución no balanceada de las clases, es decir, el entrenamiento del modelo con mas muestras de una clase que de otras.

Kappa. El Coeficiente kappa de Cohen es una medida de evaluación que intenta ajustarse al comportamiento del modelo ante el azar. En general se cree que es una medida mas robusta que la precisión, ya que esta tiene en cuenta el acierto que ocurre por el azar. Muchos investigadores consideran Kappa como una medida demasiado conservadora por el hecho de que se puede llegar a subestimar el acierto para la clase mayoritaria, es decir, la clase predominante.

AUC (*Area Under the ROC Curve*). El análisis ROC (*Receiver Operating Characteristic*) se utiliza normalmente en problemas binarios y consiste en representar cada clasificador en un espacio bi-dimensional limitado por los puntos (0,0) y (1,1). El eje Y es el ratio de positivos y el eje X el ratio de los falsos positivos. Para evaluar un clasificador en el espacio ROC se usa el área bajo la superficie convexa o AUC, basada en una curva generada variando el umbral de decisión, desde predecir todas las instancias como negativas hasta predecir todas como positivas. La curva ROC es

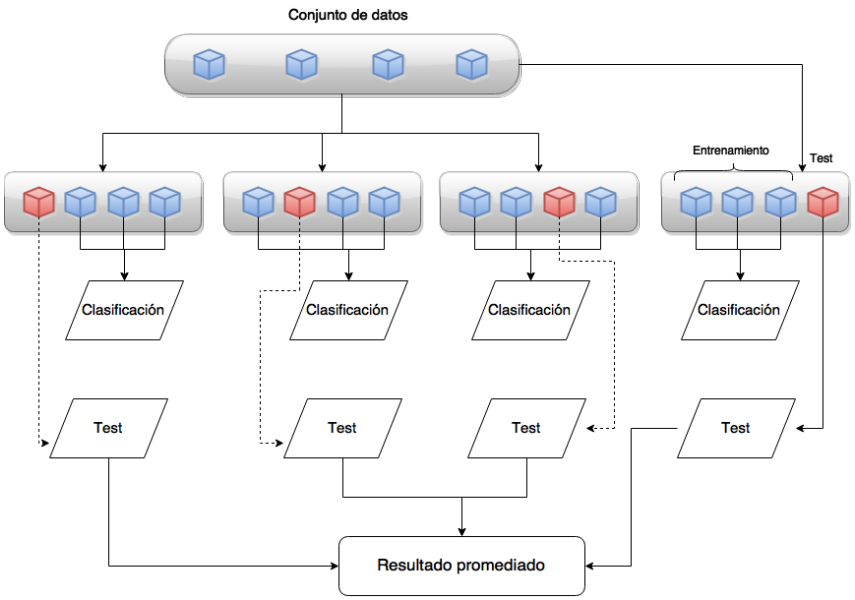
independiente de la distribución de las clases en el conjunto de aprendizaje. El análisis ROC se relaciona de forma directa y natural con el análisis de coste/beneficio en toma de decisiones diagnósticas.

Estas medidas expuestas nos dan una apreciación de cuan bueno es nuestro modelo, pero aun así queda por determinar que conjunto de datos se utilizará para obtener estas medidas de evaluación. Existen muchas metodologías para la realización de una correcta experimentación, las mas conocidas son *Training and Test* y *Validación Cruzada*.

Training and Test. Esta metodología de experimentación es la mas extendida debido a su simplicidad. Partiendo de un conjunto de datos $\mathcal{D}$, consiste en dividir $\mathcal{D}$ en dos sub-conjuntos disjuntos $\mathcal{D}_{test} \in \mathcal{D}$ y $\mathcal{D}_{training} \in \mathcal{D}$. $\mathcal{D}_{training}$ se usara para entrenar el modelo y el conjunto $\mathcal{D}_{test}$ para evaluarlo teniendo en cuenta las medidas antes descritas.

Validación cruzada La anterior metodología tiene varios inconvenientes, entre ellos la poca generalización de los resultados en el caso de que el conjunto de datos no sea lo suficientemente grande, es decir si realizamos el aprendizaje con un pequeño conjunto de datos, los resultados pueden depender de cómo hemos realizado la partición de D. La validación cruzada funciona de manera similar al anterior método pero dividiendo los datos en $\mathcal{K}$ sub-conjuntos para luego, aprender $\mathcal{K}$ modelos, dejando siempre un sub-conjunto de $\mathcal{K}$ fuera del entrenamiento, el cual sera utilizado como conjunto de *Test*. De esta forma se garantiza que son independientes tanto los datos de *Training* como los datos de *Test*. En la figura 2.10 se muestra el proceso de validación cruzada.

Figura 2.10: Validación cruzada.



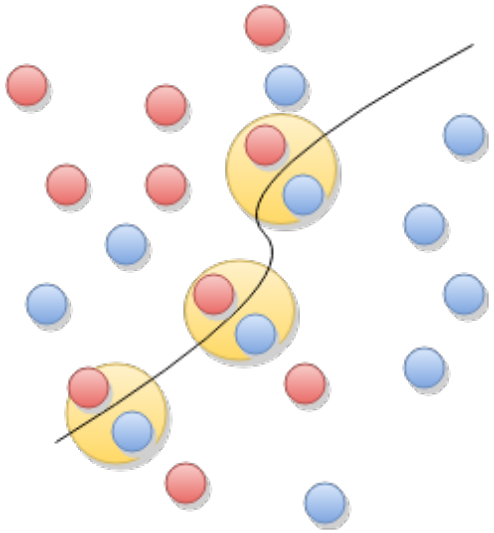
2.2.3. Problemas de clasificación no balanceados

Los métodos de aprendizaje comentados son sensibles a la distribución de las clases. Por este motivo la precisión obtenida (resultado de la evaluación de los modelos) puede quedar comprometida si no estamos ante una distribución balanceada. Esta situación es bastante habitual cuando una de las clases (la mas minoritaria) representa casos especiales, bastante raros pero muy importantes para la resolución de nuestro problema. Existen métodos para mitigar el desbalanceado de las clases, los cuales se pueden dividir en dos enfoques: métodos de *over-sampling* y métodos de *under-sampling*. Los enfoques de *over-sampling* aumentan el número de instancias de las clases minoritarias, mientras que los de *under-sampling* reducen el número de instancias de la clase mayoritaria.

Una técnica sencilla para incrementar el tamaño de la clase minoritaria es balancear la distribución mediante el replicado aleatorio de ejemplos de la clase. Chawla propuso *Synthetic Minority Over-sampling Technique* o SMOTE, un enfoque de *over-sampling* en el que la clase minoritaria incrementa su tamaño mediante un método de creación de instancias artificiales. La técnica consiste en incrementar el tamaño de la clase minoritaria cogiendo cada ejemplo de la clase e introduciendo ejemplos sintéticos creados a partir de la evaluación de los k vecinos mas cercanos [5][12].

Por otro lado, una técnica sencilla de *under-sampling* es eliminar de forma aleatoria instancias de la clase mayoritaria. Esta técnica, aunque simple, puede ocasionar que se eliminen ejemplos que resulten importantes para la definición de las regiones de clasificación. Un método de *under-sampling* que ofrece mejores resultados es eliminar aquellas instancias de la clase mayoritaria que son *Tomek Links* [3]. Suponiendo un conjunto de datos E tal que $(e_1, ..., e_n \in E)$ y donde cada elemento e_i contiene un atributo determinando de clase, con valores $(+, -)$. Se consideran *Tomek Links* aquellas instancias etiquetadas como $(+)$ cuya distancia con la instancia mas próxima de clase $(-)$ es mínima. Un ejemplo de aplicación lo vemos en la figura 2.11. Eliminar estas observaciones supondría tener una mejor separación entre las regiones de las clases [8].

Figura 2.11: *Tomek Links.*



Capítulo 3

Metodología y Plan de Trabajo

3.1. Metodología

En este trabajo hemos aplicado la metodología CRISP-DM (*Cross Industry Standard Process for Data Mining*[29]), ya que es la mas utilizada en el desarrollo de proyectos de minería de datos. La metodología CRISP-DM está descrita en términos de un modelo de proceso jerárquico consistente en un conjunto de tareas repartidas en tres niveles de abstracción.

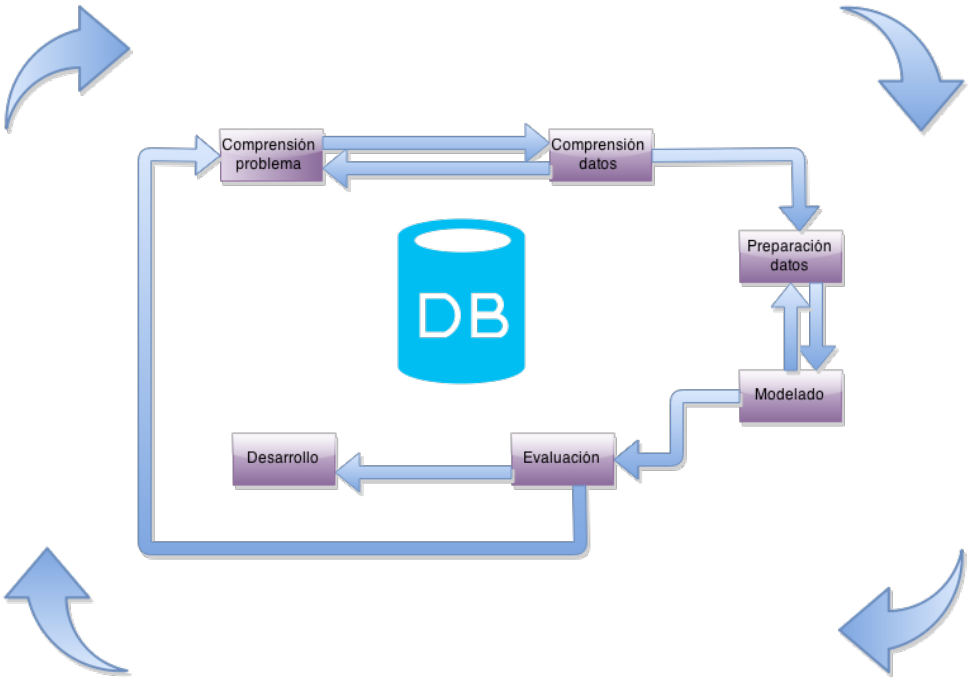


Figura 3.1: Metodología seguida durante el estudio.

En el primer nivel, el proceso de minería de datos se organiza en un número determinado de fases, constando cada fase de varias tareas genéricas de segundo nivel. El siguiente

nivel es donde se detallan cómo se deberán de realizar las acciones del nivel anterior ante situaciones especificas, e. g. como tarea genérica podríamos tener la normalización de los datos y como situaciones específicas de aplicación la normalización de datos numéricos o categóricos. Y en el último nivel, se encuentra el registro o informe de todo el proceso realizado. Este informe contiene todos los resultados obtenidos mediante procesos de minería de datos [23]. La figura 3.1 muestra las fases definidas, y que se han seguido durante el estudio, haciendo uso de la metodología CRISP-DM, siendo cada una de ellas objeto de exposición y detalle en siguientes puntos.
La descripción de fases y tareas como pasos discretos realizados en un orden específico representa una secuencia idealizada de eventos. En la práctica, muchas de las tareas pueden ser realizadas en un orden diferente, y esto a menudo ha significado volver a hacer tareas anteriores repetidamente y repetir ciertas acciones. Esta metodología nos ha otorgado un nivel de flexibilidad alto dentro de una correcta experimentación. A continuación, de forma resumida y genérica, se incluye una breve definición de las fases de CRISP-DM.

Comprensión del problema En esta fase se especifican de forma detallada los objetivos principales del proyecto y las tareas en las cuales se va a dividir la resolución del problema, siempre teniendo en cuenta una de las características clave de esta metodología, la flexibilidad. De esta fase depende que no se desperdicien recursos debido a una mala especificación del dominio del problema a resolver.

Comprensión de los datos Saber con qué datos vamos a trabajar, sus posibles atributos, las distribuciones de los mismos y estudiar las distintas fuentes de información posibles, son los objetivos principales de esta fase. En este punto se verifica la calidad de los datos con el fin de hacer un primera criba o selección.

Preparación de los datos La preparación de los datos se puede definir como la aplicación de técnicas para la selección, limpieza y estructuración homogénea de los datos, ademas de técnicas para la unión de distintas fuentes de información y la obtención de datos derivados.

Modelado En este punto se determina qué tipo de modelo o modelos se utilizarán, cuáles son sus parámetros óptimos y se crearán el o los informes con las medidas de evaluación seleccionadas para su estudio.

Evaluación Una vez obtenido el modelo final, queda determinar en qué medida se puede mejorar, es decir, determinar cuáles serán los siguientes pasos, si es que los hubiera, para en base a los informes generados en anteriores fases e iteraciones, obtener una mejora sobre la solución.

3.2. Plan de Trabajo

Una vez descritas las fases en las cuales se divide la metodología CRISP-DM, es necesario concretar el plan de trabajo (figura 3.2) seguido en este TFG y cómo se corresponden cada una de las fases de la metodología con el mismo.

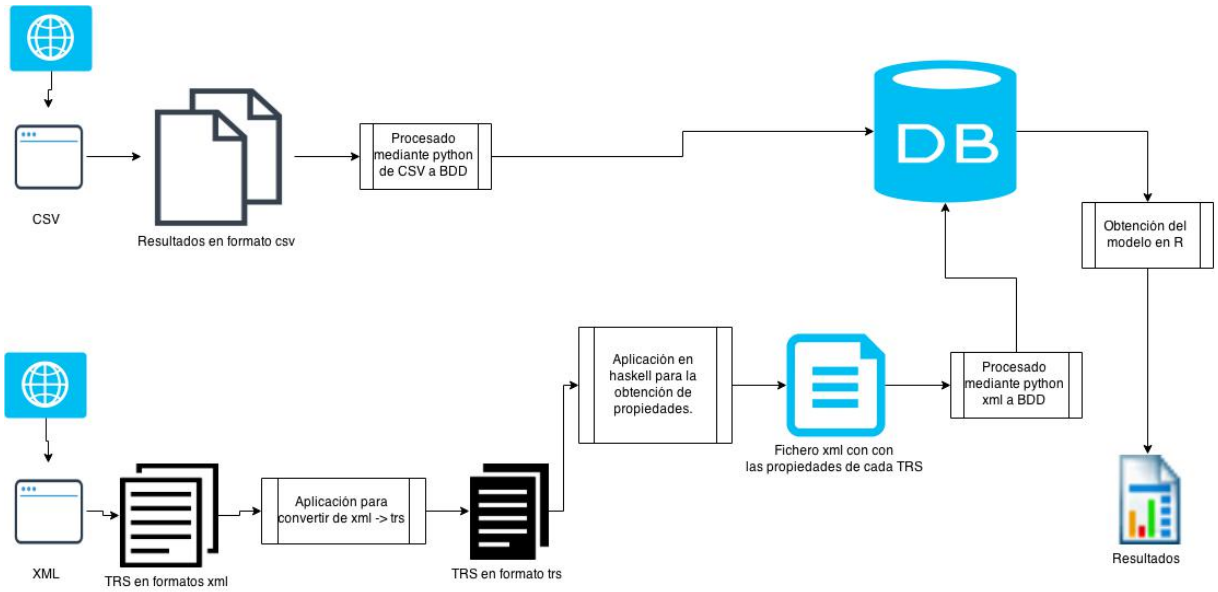


Figura 3.2: Plan de trabajo.

Comprensión del problema Esta fase ha resultado inmediata en este estudio. Hemos abordado un problema de clasificación cuyo objetivo es determinar si un TRS es o no terminante, sin necesidad de realizar ninguna demostración explícita.

Comprensión de los datos Para el estudio se ha hecho uso de la base de datos de resultados de la *Termination Competition* en la categoría de TRS en las ediciones desde el año 2008 hasta el año 2013. La información que ha sido obtenida de esta base de datos ha sido los tiempos que ha tardado cada demostrador en determinar si un TRS es o no terminante, en el caso de que haya sido posible (información no utilizada en el actual estudio pero que sería útil en otros estudios futuros), y el resultado de dicha demostración. Es decir, hasta aquí, tenemos los datos referentes a si un TRS es terminante o no según un conjunto de demostradores.

Preparación de los datos La definición formal que realiza el estándar para esta fase es la realización del conjunto de actividades necesarias para la creación de la vista minable, tareas como la selección y transformación del dominio de los datos. En este punto hemos tocado muchos aspectos básicos del problema algunos tales cómo el formato de ficheros con el cual nos convenía trabajar. Al final de esta fase, el conjunto de datos resultante es lo que ha formado nuestra vista minable.

Modelado Esta es la fase principal en la que se han generado todos los modelos predictivos, buscando minimizar el error cometido en la clasificación con las distintas configuraciones aplicadas a cada modelo.

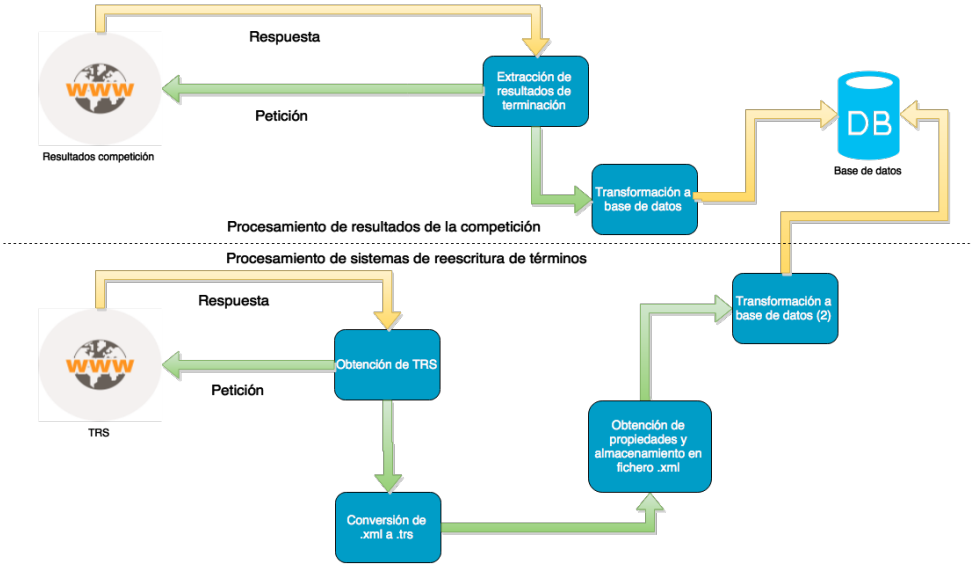
Evaluación Una vez han sido generados los modelos, estos se han evaluado con respecto a varias medidas de evaluación con el fin de seleccionar los mejores.

Capítulo 4

Preparación de los datos

Como se muestra en la figura 4.1, atendiendo a la descripción formal de la fase de preparación de los datos descrita en el capitulo anterior, este proceso ha quedado desglosado en una serie de tareas que quedaran descritas en forma de secciones a lo largo del capítulo[1].

Figura 4.1: Diagrama de preparación de los datos.



Como se puede observar en la anterior figura, las tareas realizadas para la preparación de los datos son las siguientes.

- Tarea 1: Extracción de resultados de terminación.
- Tarea 2 y 6: Transformación a base de datos.
- Tarea 3: Obtención de TRS.
- Tarea 4: Conversión de .xml a .trs.

[1]El código el cual ha servido para completar estas tareas se encuentra en el apéndice A.

- Tarea 5: Obtención de propiedades y almacenamiento en fichero .xml.

Al final del capítulo se expondrá un breve análisis de los datos resultantes de este proceso y además se describirán brevemente los lenguajes de programación y librerías usadas. A continuación se detallan cada una de las tareas antes expuestas.

4.1. Tarea 1: Extracción de los resultados de la competición

El repositorio de información usado ha sido el formado por los resultados de la *Termination Competition* desde el año 2008 hasta el año 2013, ambos inclusive[2]. La competición, dentro de la categoría de sistema de reescritura de términos, se divide en otras sub-categorías, tal y como se muestra en la tabla 4.2. En este trabajo esta división no se tiene en cuenta, pero en futuros proyectos seria interesante ver los resultados de terminación de los sistemas de reescritura en función de la categoría a la que pertenecen.

Herramienta	Versión
APoVE-COLOR	1.0, 1.8
APoVE-CERT	1.0, 1.8
APoVE-A3PAT	1.0,2010-0.2, 2011-0.1, 2011-0.2
APoVE	1.2, 1.5, 1.6, 1.8, 2010-0.2, 2011-0.1, TC2012, TC2013
APoVE-CeTA	1.8.1, 2010-0.3, 2011-0.1, TC2012, TC2013
TTT2	0.1, o0.2, 0.9, 2010, 2010x, 2013_1
TTT2Cet	2009z, 2010, 2011, 2011fixed, 2011a, 2013_, 2013_1
mu-tem	5.05, 5.07, 5.08
Jamox	2008, 2009_
JamoxGoesOut	2008
cime3	0.3
cime3finde	14dec16h26, 13jul201011h44
cime3eta	13jul201017h40
Wanda	wanda2.1.fixed, 2.1f
matchox	0.2.11
TafO	0.1.1
VMTL	1.4
NaTT	2013

Tabla 4.1: Herramientas de demostración.

Para el desarrollo de este trabajo, se han tenido en cuenta todas las herramientas de demostración que han participado en las ediciones seleccionadas de la competición así como todas las versiones de las mismas, tal y como se muestra en la tabla 4.1.

[2]Los datos han sido descargados en formato CSV, de la web oficial de la competición. http://www.termination-portal.org/

En resumen, la estructura de los datos obtenidos de este repositorio consiste en una tripleta (R, T_d, S_d) donde T_d se corresponde al tiempo que ha tardado el demostrador d en determinar el resultado S_d para el sistema de reescritura de términos R.

`Standard`
`Relative`
`Innermost`
`Equational`
`Context sensitive`
`Outermost`
`Standard Certifying`
`Standard`
`Relative`
`Conditional`
`Relative Certifying`

Tabla 4.2: Desglose de las categorías de TRS usadas en la competición.

4.2. Tarea 2: Transformación a base de datos.

Para el desarrollo de esta tarea se ha desarrollado un script en Python (apéndice A.1), el cual se encarga de recorrer de manera recursiva la estructura de directorios donde se almacenan los ficheros con los resultados de la competición, estos ficheros son leídos e insertados en la base de datos. El dominio de los resultados de la demostración de los TRS que se almacenan en la base de datos es el siguiente:

YES: la herramienta ha podido determinar que el sistema *es terminante*.

NO: la herramienta ha podido determinar que el sistema *es no terminante*.

MAYBE: la herramienta no ha podido determinar la terminación del sistema después de la ejecución de todas las pruebas.

TIMEOUT: la herramienta ha excedido el tiempo límite establecido por la competición.

Vacío: La herramienta no ha competido en la demostración del TRS.

Para este estudio hemos unificado los resultados *MAYBE* y *TIMEOUT* ya que desde el punto de vista de la terminación, ambos casos corresponden a una indeterminación (la herramienta no ha sido capaz de determinar si es o no terminante), por lo tanto vamos a considerar únicamente tres etiquetas de clase.

A continuación, se crea una columna *'resultado final'* con la etiqueta de clase que resulta de la comparación de todos los resultados obtenidos por las diferentes herramientas para cada TRS. Hay que destacar que la columna 'resultado final' ha sido difícil de definir, ya que en la base de datos existen diferentes resultados para un mismo TRS en función del demostrador. Por ejemplo, para el TRS de la fila 7 de la figura 4.2, AProVE obtiene como

resultado *YES* y AProVE-CERT obtiene *MAYBE*. Estas discrepancias deben tratarse previamente para evitar inconsistencias.

Figura 4.2: Resultados herramientas.

	tpfile text	AProVE (1 text	AProVE (1 text	Jambox (2 text	Jambox (2 text	AProVE-C text	AProVE-C text	cime3 (0.3 text	cime3 (0.3 text	AProVE-C text	AProVE-C text	matchbox text	matchbox text	AProVE-A text	AProVE-A text	T t
7	tpdb-8.0/T	YES	2122	YES	1324	MAYBE	10657	YES	189	MAYBE	10397	TIMEOUT	60242	MAYBE	1958	YI
8	tpdb-8.0/T	NO	2792	MAYBE	2075	MAYBE	4104	MAYBE	308	MAYBE	3813	MAYBE	14455	MAYBE	1086	NO
9	tpdb-8.0/T	YES	2174	YES	918	YES	4755	YES	3355	YES	3650	YES	5113	YES	3077	YI
10	tpdb-8.0/T	YES	2951	YES	1639	YES	3250	TIMEOUT	60265	YES	3657	YES	729	YES	7750	YI

Para solucionar esto hemos asumido en primer lugar, que dado el conjunto de demostradores $\mathcal{D}$ y el conjunto de sistemas de reescritura $\mathcal{R}$, en nuestra base de datos no podía quedar registrado ningún $R \in \mathcal{R}$ que obtuviera mediante un demostrador $\mathcal{D}_j \in \mathcal{D}$ un resultado *YES* y para un demostrador $\mathcal{D}_r \in \mathcal{D}$, siendo $j \neq r$, un resultado *NO*. Para ello se han realizado una serie de pruebas con el fin de identificar estos casos, para luego mediante un *script SQL* borrarlos (apéndice A.4). En segundo lugar, hemos asumido que dado un R cualquiera, cuando un $\mathcal{D}_j \in \mathcal{D}$ obtiene *YES* o *NO* y un demostrador $\mathcal{D}_r \in \mathcal{D}$, siendo $j \neq r$, obtiene *MAYBE* en la demostración de R, el 'resultado final' de la demostración es aquel que representa la respuesta mas definida.

4.3. Tareas 3 y 4: Obtención de TRS y Conversión de .xml a .trs.

Con motivo de una valoración mas completa de las herramientas de demostración, en cada edición el conjunto de TRS a demostrar cambia, bien porque los participantes proponen nuevos sistemas de reescritura o bien por que se decide eliminar antiguos. La versión del conjunto de TRS utilizado para este trabajo es la 8.0[3] ya que es la que contiene todos los TRS de las ediciones seleccionadas en la tarea 1.

Debido a que la librería que se usará en la tarea 5 para la obtención de las propiedades de los sistemas de reescritura solo trabaja con ficheros con extensión .trs y los ficheros con los sistemas de reescritura se presentan en un formato XML, se ha realizado una conversión de formato xml a formato trs (la figura 4.3 muestra un ejemplo de un TRS en formato trs).

Figura 4.3: Ejemplo de sistema de reescritura expresado en formato *TRS*.

```
(VAR x y )
(STRATEGY INNERMOST)
(RULES
  f(g(x), s(0), y) → f(y, y, g(x))
  g(s(x)) → s(g(x))
  g(0) → 0
)
```

[3]http://termcomp.uibk.ac.at/status/downloads/tpdb-8.0.tar.gz

4.4. Tarea 5: Obtención de propiedades.

Una vez se ha almacenado en la base de datos la información concerniente a la terminación de los TRS, el siguiente paso es construir la vista minable. En ella, cada fila representará un TRS que vendrá descrito mediante una serie de propiedades (los campos o atributos). Todas las propiedades de los sistemas de reescritura de términos usadas cumplen la característica de poder ser obtenidas sin necesidad de realizar ninguna acción relacionada con la demostración de la terminación.

Dado un sistema de reescritura de términos R, las propiedades obtenidas mediante el código escrito en Haskell del apéndice A.2, son las que se describen a continuación.

Número de símbolos. Número de símbolos de función diferentes que aparecen en R. El dominio de la salida de esta propiedad es $\mathbb{N}$.

Ground. R es *ground* si no contiene variables. El dominio de la salida tanto de esta como de las siguientes propiedades es $True, False$.

Left/Right Ground. Cuando para cada regla $l \to r \in$, el término l/r es *ground*.

Duplicating. R es *duplicating* si existe alguna variable que para alguna regla $l \to r \in$R aparece mas veces en r que en l.

Linear. Un término es lineal si cada una de sus variables ocurre sólo una vez. R es lineal si para toda regla $l \to r \in$R, l y r son lineales.

Left/Right Linearity. R es lineal por la izquierda/derecha si para toda regla $l \to r \in$R, el término l/r es lineal.

Collapsing. R es *collapsing* si existe una regla en R de la forma $l \to x$ siendo x una variable.

Shallow. R es *shallow* si todas las variables ocurren como mucho en la profundidad uno, en ambos lados de cada regla.

Recursive. Se hace uso de una noción básica de recursividad. Dada una regla $l \to r \in$R, es recursiva si el símbolo mas externo de l aparece en r. R es recursivo si contiene alguna regla que sea recursiva.

Erasing. R es *erasing* si existe una regla donde alguna variable ocurre en el lado izquierdo y no ocurre en lado derecho.

El programa del apéndice A.2 recorre todos los directorios donde están los sistemas de reescritura de términos, los evalúa dando valores a las propiedades antes mencionadas y almacena esta nueva información en un fichero XML.

4.5. Tarea 6: Transformación a base de datos.

Esta es la última tarea del proceso de preparación de los datos. Una vez creado el fichero XML de la tarea anterior, éste es leído e insertado en la base de datos, mediante el código del apéndice A.3, quedando por lo tanto un TRS definido por el conjunto de propiedades y etiquetado en función de su terminación. La figura 4.4 muestra la vista minable resultado de todo el proceso de preparación de los datos. Con esta vista minable se crearan los modelos de aprendizaje automático, los cuales se detallarán en el siguiente capítulo.

Figura 4.4: Vista minable de la base de datos.

	numsimb integer	depth integer	isLeftLinear integer	isLinear integer	isRightLinear integer	isCollapsing integer	isDuplicating integer	isGround integer	isLeftGround integer	isRightGround integer	isErasing integer	isCreating integer	isShallow integer	isRecursive integer	Res text
1	9	3	1	1	1	1	0	0	0	0	1	0	1	1	YES
2	33	5	1	0	0	1	1	0	0	0	1	0	0	1	MAYB
3	73	7	0	0	0	0	1	0	0	0	1	0	0	1	YES
4	10	6	1	1	1	0	0	0	0	0	0	0	0	1	MAYB
5	57	4	1	0	0	1	1	0	0	0	1	0	0	1	MAYB
6	74	8	0	0	0	0	1	0	0	0	0	0	0	1	YES
7	21	4	1	0	0	1	1	0	0	0	0	0	0	1	YES
8	9	3	1	1	1	0	0	0	0	0	1	0	0	1	NO
9	12	4	1	1	1	0	0	0	0	0	0	0	0	1	YES
10	24	7	1	1	1	0	0	0	0	0	0	0	0	1	YES

4.6. Análisis exploratorio de los datos

A continuación se detallan algunos aspectos del análisis de los datos efectuado.

Clase	#Instancias
YES	1514
NO	342
MAYBE	300

Tabla 4.3: Número de instancias.

En la tabla (tabla 4.3) se muestra la distribución de las instancias que conforman la vista minable. Como se puede observar hay 5 veces mas instancias de la clase *YES* que de las otras clases. Este hecho con seguridad se hará patente en los resultados de la evaluación de los distintos modelos.

Las figuras 4.5 y 4.6 muestran los histogramas de las propiedades *'linear'* y *'left linearity'* para cada clase (*YES, NO, MAYBE*) se puede deducir que un alto porcentaje de los sistemas de reescritura que forman el estudio son 'Lineales por la izquierda' y otro alto porcentaje no son 'Lineales' (en ambos casos mas del 60 %); así que recordando la definición de 'Linealidad', tenemos que en comparación con la propiedad de 'Lineales por la izquierda' hay muchos menos sistemas que sean 'Lineales por la derecha'.

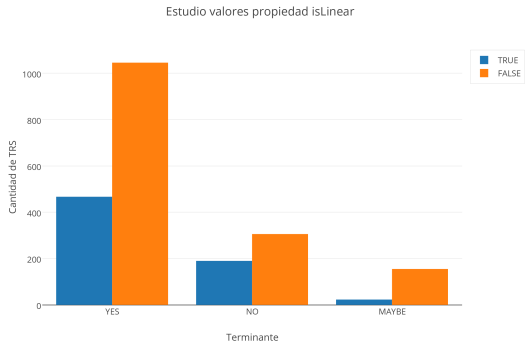


Figura 4.5: Estudio de la propiedad '*Linear*'.

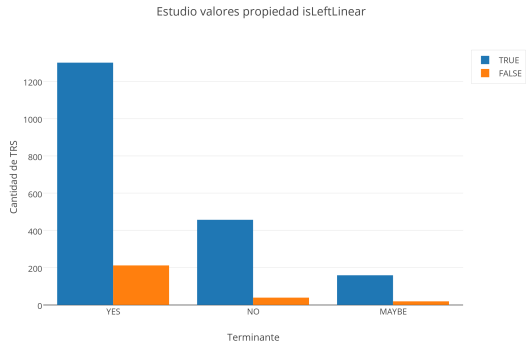


Figura 4.6: Estudio de la propiedad '*Left Linearity*'.

Por otro lado, tenemos que casi la totalidad de las instancias son no '*Ground*', como muestra la figura 4.7. Y por último, para propiedades como *'Duplicating'*, *'Erasing'* y *'Collapsing'*, como se puede ver en las figuras 4.8, 4.9 y 4.10, las instancias se reparten con una misma distribución entre las distintas clases para los valores *'True'* y *'False'*, significando que estas propiedades por si solas son las que menos aportan al modelo.

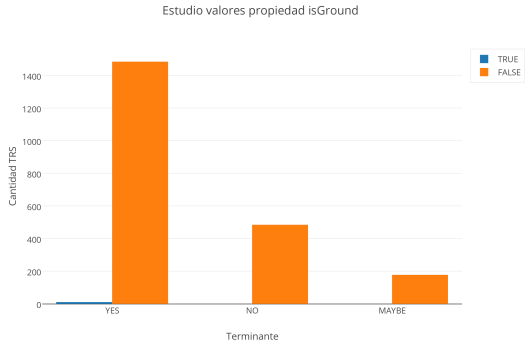


Figura 4.7: Estudio de la propiedad '*Ground*'.

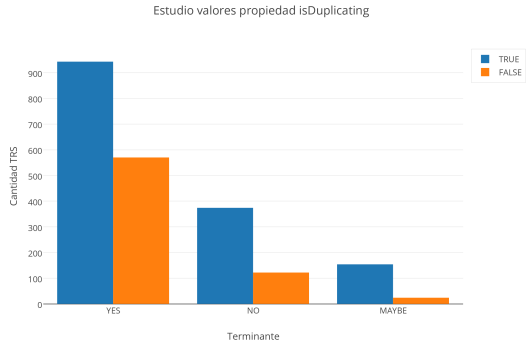


Figura 4.8: Estudio de la propiedad '*Duplicating*'.

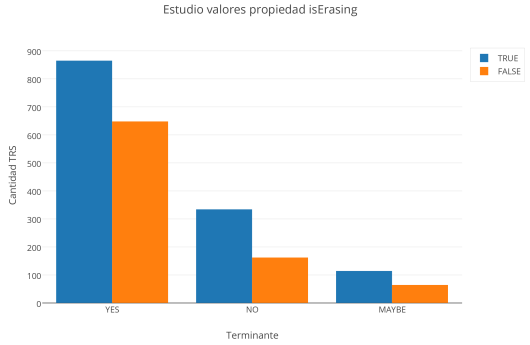


Figura 4.9: Estudio de la propiedad '*Erasing*'.

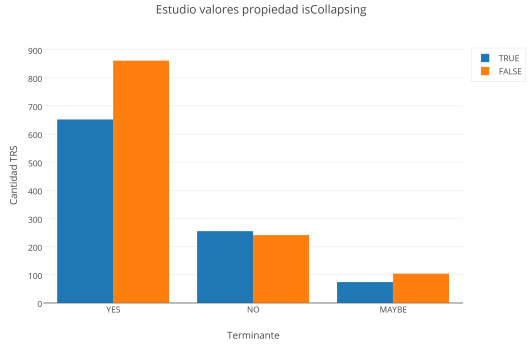


Figura 4.10: Estudio de la propiedad '*Collapsing*'.

La conclusión que podemos obtener en base a estos datos es que ninguna de las propiedades son lo suficientemente discriminantes para determinar por sí solas la terminación de un TRS. En el siguiente capítulo mostraremos como los modelos obtenidos a partir de todas las propiedades permiten predecir la terminación con tasas de acierto aceptables.

4.7. Relación de lenguajes y librerías usadas

Python 2.7.6. Se trata de un lenguaje de programación multiplataforma, disponible para la gran mayoría de sistemas operativos y también multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Este lenguaje de programación favorece la creación de código legible. La versatilidad que ofrece nos ha facilitado en gran medida la implementación varias de las tareas 1, 2 y 6.

Librerías usadas:

csv La librería *csv* implementa clases para la lectura y escritura de archivos en formato *CSV*, formato en el cual hemos obtenido los resultados de la competición.

psycopg2 2.6 Para el almacenamiento de los datos hemos utilizado una base de datos *PostgreSQL* y la librería *psycopg2 2.6* ofrece funciones para trabajar con bases de datos de este tipo en Python.

xml.dom Esta librería ofrece funciones para el tratamiento de fichero .xml en Pyton. Ha sido necesaria esta librería para la lectura del fichero .xml generado en la tarea 5, el cual contiene todas las propiedades de todos los TRS que forman la base de datos.

***The Glorious Glasgow Haskell Compilation System, version* 7.8.3** Es el intérprete de Haskell, siendo este el lenguaje de programación usado para obtener las propiedades de los sistemas de reescritura de términos en la tarea 5.

Librerías usadas:

term-rewriting Esta librería ha sido desarrollada por participantes y organizadores de la competición de terminación y también investigadores del campo de los sistemas de reescritura ajenos a ella. *Term-rewriting* es una librería escrita en Haskell la cual provee de una funcionalidad básica para trabajar con sistemas de reescritura de primer orden[10].

xsltproc Es un programa usado en este proyecto para convertir los sistemas de reescritura en formato *XML* a formato *TRS*. Es una forma sencilla de usar la librería *libxslt*.

Capítulo 5

Estudio experimental: Generación de los modelos

Este capítulo consta de una descripción detallada de todos los experimentos que se han realizado. Una versión preliminar de los experimentos incluidos en este capítulo se han publicado en el artículo *Analysing the Termination of Term Rewriting Systems using Data Mining*. Antes de comenzar con la experimentación hay algunos puntos básicos que es meritorio tener en cuenta.

Para el desarrollo de estos modelos se ha usado R cómo lenguaje de programación. Este lenguaje nace como resultado de la implementación *GNU* del premiado lenguaje *S*. R y *S-Plus* —versión comercial de *S*— son probablemente los dos lenguajes de programación más utilizados en investigación por la comunidad estadística y de *data science*, siendo además muy populares en el campo de la investigación biomédica, de la bioinformática y de las matemáticas financieras. Un listado detallado de las librerías de R que se han utilizado es el siguiente:

Caret **[17]** Es un completo paquete de funciones que entre otros objetivos tiene el preprocesado y particionado de los datos, la selección de atributos y la definición de modelos. Sirve tanto para tareas de clasificación como para tareas de regresión. Caret se ha usado en este proyecto para crear todos los modelos de aprendizaje, aunque también se podrían haber considerado otros paquetes como RWeka, es mas algunos modelos de Caret cómo el J48, son implementaciones de RWeka.

unbalanced **[24]** Paquete R con implementaciones de muchas técnicas para el balanceado de clases.

pROC **[25]** Paquete R que se ha usado para el cálculo del AUC en problemas de clasificación con mas de dos clases.

DBI **[22]** Interfaz para la comunicación entre R y sistemas de bases de datos relacionales. Todas las clases que incluye son virtuales por lo tanto es necesario cargar una librería que las implemente.

RPostgreSQL **[6]** Este paquete ofrece un *driver* compilado para el tratamiento de bases de datos *PostgreSQL* desde R.

Respecto a la metodología experimental, se ha optado por utilizar validación cruzada para obtener y validar los modelos, motivados sobre todo por la poca cantidad de ejemplos que disponemos. Usaremos 10 pliegues y para obtener una medida mas realista, nos hemos decantado por repetir el proceso de validación cruzada 10 veces, lo que hace un total de 100 ejecuciones. Por lo tanto, los resultados que se muestran en este capítulo son el promedio obtenido en las 100 iteraciones.

Las técnicas que se han utilizado para la obtención de los modelos han sido árboles de decisión (J48), máquinas de vectores soporte (svmLinear), redes neuronales (nnet) y k-NN. Caret ofrece un proceso de ajuste de los modelos con el fin de obtener la configuración óptima de los mismos, en este capítulo se hará un estudio de los experimentos probados y los resultados presentados serán los de las configuraciones óptimas encontradas. Las medidas que se han utilizado para comparar los modelos han sido Kappa, AUC y precisión. El código para la generación de los modelos se adjunta en los apéndices A.5 y A.6.

Hemos efectuado dos tipos de experimentos. El primero sigue una aproximación supervisada en lo que hemos considerado un problema de clasificación con tres clases. El segundo experimento sigue una aproximación semi-supervisada, en la que parte de los datos se considerarán sin valor de etiqueta de clase, mientras que el resto de instancias si están etiquetadas con dos posibles valores (*YES* y *NO*). En este segundo experimento se estudiarán tanto clasificadores binarios como clasificadores binarios influenciados por el re-etiquetada de la clase *MAYBE*.

5.1. Enfoque supervisado

En este enfoque queda patente la complejidad del problema de terminación al representar los tres posibles valores de clase a los que se acogen los TRS. Debido a que no se tiene constancia de estudios previos, el proceso experimental ha sido guiado en función de los resultados obtenidos en este punto y las conclusiones obtenidas en el capítulo anterior. Los primeros resultados obtenidos se muestran en la tabla 5.1 y en la matriz de confusión presentada en la tabla 5.2. Para una lectura mas sencilla, los mejores resultados obtenidos se muestran en negrita.

Técnica	Precisión	SD Precisión	Kappa	SD Kappa	AUC
J48	**0.7374**	**0.0198**	**0.2757**	**0.0612**	**0.5704**
svmLinear	0.7336	0.0119	0.1867	0.0496	0.5
nnet	0.7301	0.0176	0.2392	0.0585	0.5
k-NN	0.6782	0.0225	0.1503	0.0595	0.5946

Tabla 5.1: Resultados enfoque supervisado, sin balanceado de clases.

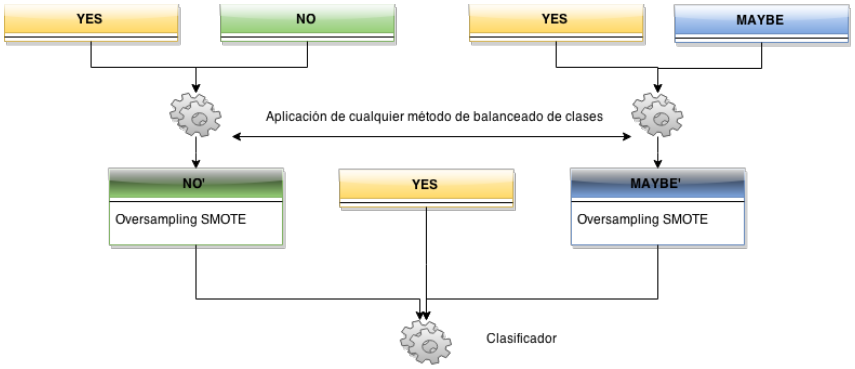
Muchos valores de AUC están en torno al 0.5 lo que significa que tienen un comportamiento muy próximo al de un clasificador al azar. Como se observa (tabla 5.2), J48 predice para la mayoría de los TRS la clase *YES*, de esta forma obtiene unos resultados de precisión altos pero a costa de no representar correctamente las demás clases.

Estimado/Real	*YES*	*NO*	*MAYBE*
YES	1488	323	140
NO	25	171	26
MAYBE	0	2	12

Tabla 5.2: Enfoque supervisado - J48: Matriz de confusión.

Cómo se ha visto en la tabla 4.3 estamos ante un problema de clasificación con la distribución de clases des-balanceada. Está demostrado que la precisión no es una medida correcta para representar problemas de este tipo (ver artículo [18]), por este motivo en estudios siguientes se omitirá esta medida y se hará uso de las medidas Kappa y AUC como medidas de estudio. Por otra parte, aunque existen técnicas de aprendizaje las cuales sufren poca penalización ante el des-balanceado de clases, se ha decidido balancear las clases sintéticamente mediante técnicas especificas de balanceado de clases.

Figura 5.1: Balanceado de tres clases: *over-sampling*.

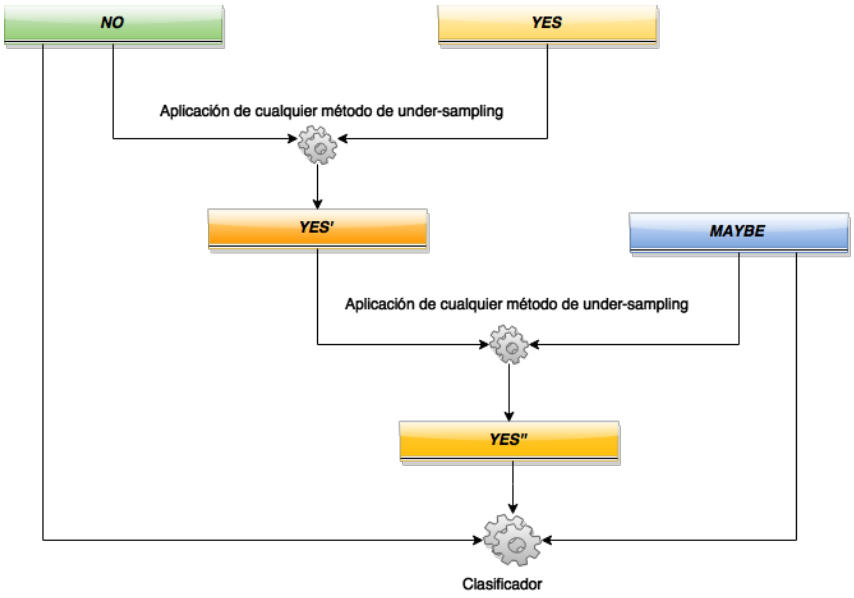


Las técnicas que se han seleccionado para balancear las clases son *SMOTE*, *TOMEK* y *ENN*, siendo *SMOTE* de entre las usadas la técnica mas agresiva, en cuanto a la modificación de los datos del problema, hecho que se retomará en secciones siguientes. Para poder aplicar estas técnicas, implementadas todas ellas en la librería *unbalanced*, se ha tenido que diseñar un algoritmo de aplicación especifico ya que estas implementaciones solo contemplan la existencia de dos clases, la mayoritaria y la minoritaria. La figura 5.1 muestra los pasos seguidos para la aplicación de la técnica de *over-sampling SMOTE*, siendo esta técnica la única aplicada de este tipo. En primer lugar cogemos el total de las instancias y las dividimos en tres subconjuntos, uno por cada clase. En el siguiente paso, realizamos *over-sampling* sobre el par de clases (YES, NO) y sobre $(YES, MAYBE)$, considerando en cada ejecución NO y $MAYBE$ como clase minoritaria y YES como la mayoritaria. Resultado de este paso tenemos los conjuntos NO' y $MAYBE'$, los cuales usamos nuevamente junto al conjunto *YES* para entrenar el clasificador y crear el modelo en cuestión.

Tal y como se muestra en la figura 5.2, un esquema similar de aplicación se ha realizado para poder usar métodos de *under-sampling* en este enfoque. En resumen el proceso de *under-sampling* consta de dos pasos, en cada uno de ellos la clase modificada es la mayoritaria. En el primer paso se reduce la clase *YES* comparándola con la clase *NO* y el resultado es *YES'*. En el siguiente paso se reduce de nuevo la clase *YES'* comparándola

con la clase *MAYBE* y como resultado se obtiene un conjunto de clases balanceadas.

Figura 5.2: Balanceado de tres clases: *under-sampling*.



Decir también, que se han usado mezclas de técnicas de *over-sampling* y *under-sampling*.

Técnica	Kappa	SD Kappa	AUC
J48	0.2757	0.0612	0.5704
J48 SMOTE	0.5340	0.0340	0.8263
J48 TOMEK	0.3522	0.0609	0.6636
J48 SMOTE-TOMEK	**0.6036**	**0.0427**	**0.8538**
J48 SMOTE-ENN	**0.5702**	**0.0388**	**0.8418**
svmLinear	0.1867	0.0498	0.5000
svmLinear SMOTE	0.2095	0.0306	0.5524
svmLinear TOMEK	0.1882	0.0602	0.4962
svmLinear SMOTE-TOMEK	0.2454	0.0418	0.5815
svmLinear SMOTE-ENN	0.2261	0.0389	0.5822
nnet	0.2392	0.0585	0.5000
nnet SMOTE	0.3385	0.0534	0.6519
nnet TOMEK	0.3116	0.0658	0.6067
nnet SMOTE-TOMEK	0.3644	0.0552	0.6909
nnet SMOTE-ENN	0.3635	0.0624	0.6259
k-NN	0.1503	0.0595	0.5946
k-NN SMOTE	0.3093	0.0323	0.668
k-NN TOMEK	0.2065	0.0525	0.6443
k-NN SMOTE-TOMEK	0.3611	0.0419	0.7441
k-NN SMOTE-ENN	0.3544	0.0421	0.7085

Tabla 5.3: Resultados enfoque supervisado, estudio completo.

En la tabla 5.3 se muestran los resultados de este enfoque para las medidas Kappa y AUC. Como se puede ver con el fin de tener una referencia, se muestran los resultados para los clasificadores con balanceado de clases y para los originales, esta forma de mostrar los resultados se mantendrá durante toda la experimentación. Los modelos que ofrecen los resultados mas interesantes son J48 SMOTE-TOMEK y J48 SMOTE-ENN. Resulta interesante ver como salvo en el caso de J48, ningún modelo obtiene resultados interesantes. En el caso de SVM es necesario señalar que los resultados obtenidos son bastante malos aún habiendo aplicado técnicas de balanceado.

Los modelos de árbol de decisión resultan sobre todo los mas interesantes, ademas que ofrecen en este caso los mejores resultados estos nos ofrecen un modelo en forma de reglas el cual se puede utilizar para hacer un sistema de ayuda a la toma decisiones, motivación inicial de este proyecto.

```
isCreating <= 0
| isDuplicating <= 0
| | isLeftLinear <= 0.320939: YES (104.0/13.0)
| | isLeftLinear > 0.320939
| | | isErasing <= 0.005681
| | | | numsimb <= 10.873198
...
— | | | numsimb > 10.873198
...
— | | isErasing > 0.005681
— | | | isErasing <= 0.989432
...
— | | | isErasing > 0.989432
...
| isDuplicating > 0
| | isLinear <= 0
| | | numsimb <= 29.918565
| | | | depth <= 4.017497
...
| | | | depth > 4.017497
...
| | | numsimb > 29.918565
| | | | numsimb <= 117
...
| | | | numsimb > 117
...
| | isLinear > 0: MAYBE (139.0)
isCreating > 0: NO (276.0)
```

Figura 5.3: J48 SMOTE-TOMEK Enfoque supervisado - En forma de reglas.

En la figura 5.3 se muestra el modelo resultante (J48 SMOTE-TOMEK) en forma de reglas. No se muestra entero debido a que es demasiado extenso para ponerlo completo. Es interesante ver que en las ramas del árbol, aún siendo entero el dominio de las

variables del problema, se realizan tests con números decimales; esto es debido a que el algoritmo *SMOTE* genera datos independientes del dominio y ajustados a la vecindad de las instancias. Debido a esto, es interesante fijarse en las matrices de confusión resultantes de clasificar los datos iniciales (sin balancear), debido a que es posible que se produzca sobre-ajuste.

A continuación se muestran las matrices de confusión de ambos modelos obtenidas tras la clasificación de los datos iniciales del problema. Ambos modelos obtienen resultados similares en la clasificación de la clase *MAYBE* pero difieren en los resultados de las demás clases. Aunque es mejor modelo el J48 SMOTE-TOMEK, el J48 SMOTE-ENN acierta en mas instancias, de tal forma que se hace patente en este caso que la medida de precisión eliminada en anteriores pasos, no es una medida idónea para la evaluación de estos modelos ya que tiene tan solo en cuenta el numero de aciertos respecto al total.

Estimado/Real	*YES*	*NO*	*MAYBE*
YES	1067	55	49
NO	436	437	93
MAYBE	10	4	36

Tabla 5.4: Enfoque supervisado - J48 SMOTE-TOMEK: Matriz de confusión.

Estimado/Real	*YES*	*NO*	*MAYBE*
YES	1254	108	79
NO	239	381	60
MAYBE	20	7	39

Tabla 5.5: Enfoque supervisado - J48 SMOTE-ENN: Matriz de confusión.

Cómo se observa y a diferencia de los mostrado en la tabla 5.2 ya no tenemos clasificadores que responden *YES* para casi todo, si no que tenemos unos modelos que responden con bastante exactitud a la complejidad del problema de la terminación. Aún así las matrices de confusión nos muestran que las propiedades que hemos utilizado para definir los TRS no son lo suficiente discriminantes para diferenciar sin error entre las tres clases.

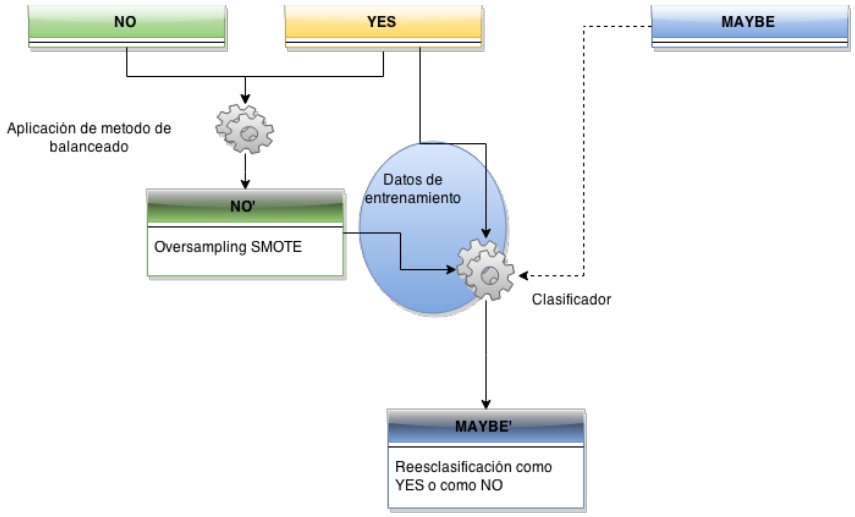
5.2. Enfoque semi-supervisado.

Este enfoque nace del significado de la clase *MAYBE*. Hemos considerado las instancias de esta clase como instancias no etiquetadas del resto de clases, es decir instancias sin valor de clase pendientes de etiquetado.

Se ha realizado un método de dos pasos para realizar el experimento. El primero es la creación de un modelo de clasificación binario, entrenado con las clases *YES* y *NO*. A este modelo se le pasaran las instancias sin etiquetar del conjunto *MAYBE* y se les asignará un nuevo valor de clase. Este paso es el que se muestra en la figura 5.4 y el que da solución al problema de la *Indeterminación del significado de* Maybe y al de el *Balanceado de tres clases*. El segundo paso, es la creación de un nuevo modelo de clasificación binario,

entrenado como en el paso anterior con los conjuntos *YES* y *NO*, pero esta vez añadiendo las instancias re-etiquetadas del conjunto *MAYBE*.

Figura 5.4: Balanceado de dos clases y re-etiquetado del conjunto *MAYBE*.



Para un correcto estudio de este enfoque a continuación se muestran los resultados obtenidos en ambos pasos de forma que será posible determinar cuanto beneficio aporta el re-etiquetado del conjunto *MAYBE*.

	Dos clases			Re-etiquetado *MAYBE*		
Técnica	Kappa	SD Kappa	AUC	Kappa	SD Kappa	AUC
J48	0.3421	0.0655	0.6755	0.4034	0.0767	0.6881
J48 SMOTE	0.5561	0.0383	0.8335	**0.6041**	**0.0413**	**0.8496**
J48 TOMEK	0.4083	0.0751	0.0706	0.4619	0.0627	0.7328
J48 SMOTE-TOMEK	**0.6091**	**0.0430**	**0.8562**	**0.6776**	**0.0362**	**0.8925**
J48 SMOTE-ENN	**0.5845**	**0.0461**	**0.8525**	**0.6572**	**0.0368**	**0.8832**
svmLinear	0.2536	0.0693	0.5927	0.2587	0.0690	0.5927
svmLinear SMOTE	0.2590	0.0317	0.6181	0.2937	0.0405	0.6372
svmLinear TOMEK	0.2551	0.0729	0.6156	0.2576	0.0684	0.5923
svmLinear SMOTE-TOMEK	0.2809	0.0392	0.6378	0.3291	0.0402	0.6568
svmLinear SMOTE-ENN	0.2660	0.0435	0.6291	0.3250	0.0434	0.6557
nnet	0.3253	0.0730	0.6514	0.3678	0.0675	0.6636
nnet SMOTE	0.4184	0.0455	0.7015	0.4765	0.0508	0.7481
nnet TOMEK	0.3802	0.0762	0.6856	0.4154	0.0696	0.7236
nnet SMOTE-TOMEK	0.4599	0.0545	0.7434	0.5021	0.0469	0.7626
nnet SMOTE-ENN	0.4379	0.0498	0.7270	0.4806	0.0499	0.7295
k-NN	0.1834	0.0534	0.6579	0.2644	0.0692	0.6719
k-NN SMOTE	0.4165	0.0483	0.7935	0.4618	0.0494	0.7968
k-NN TOMEK	0.2340	0.0615	0.6822	0.3193	0.0629	0.7110
k-NN SMOTE-TOMEK	0.4809	0.0492	0.8272	**0.5298**	**0.0447**	**0.8272**
k-NN SMOTE-ENN	0.4363	0.0494	0.7980	0.5080	0.0418	0.8101

Tabla 5.6: Resultados enfoque semi-supervisado, estudio completo.

A la vista de los resultados expuestos en la tabla 5.6, queda claro que es beneficioso el paso de re-etiquetado de la clase *MAYBE*. Sería interesante si hubiera algún otro repositorio de TRS, probar como se comporta este enfoque ante mas instancias de esta clase. A continuación, se muestran las matrices de confusión de los mejores modelos de ambos pasos del experimento (J48 SMOTE-TOMEK, ambos).

Estimado/Real	*YES*	*NO*
YES	1155	84
NO	358	412

Tabla 5.7: Enfoque semi-supervisado dos clases - J48 SMOTE-TOMEK: Matriz de confusión.

Estimado/Real	*YES*	*NO*
YES	1216	78
NO	297	418

Tabla 5.8: Enfoque semi-supervisado re-etiquetado - J48 SMOTE-TOMEK: Matriz de confusión.

Viendo las matrices queda confirmado que cuando re-etiquetamos las instancias de la clase *MAYBE* se produce una mejor clasificación de las instancias del resto de clases. En conclusión y a la vista de los últimos resultados, el enfoque de re-etiquetado de las instancias de la clase *MAYBE* es el que mejores valores obtiene de entre todos los modelos. Del mimo modo que se hacia patente que los modelos en forma de árbol eran los mejores en anteriores apartados, J48 SMOTE-TOMEK y J48 SMOTE-ENN, son los mejores modelos. En ambos modelos se ha logrado una mejor clasificación de las instancias de la clase *NO* y no se ha empeorado demasiado la clasificación de las instancias *YES*.

Llegados a este punto y siendo este último el enfoque mas simple para este problema desde las perspectiva del número de clases, la conclusión que podemos obtener es que las propiedades de los sistemas de reescritura de términos que hemos utilizado durante todo el proyecto son demasiado genéricas, es decir, no son lo suficiente descriptivas como para ayudar a discernir sin temor a error entre las distintas clases.

Si la colección de datos fuera mayor se podrían hacer otro tipo de pruebas para evaluar la calidad de los modelos generados.

Capítulo 6

Conclusión

Este trabajo responde a la idea expuesta al inicio de esta memoria sobre la necesidad de examinar cualquier tipo de software con la mejor de las lupas con el único fin de analizar sus propiedades. Además este trabajo se enmarca dentro del ámbito del análisis y verificación de código, pero se diferencia de otros estudios existentes en que tiene en cuenta el valor añadido de la información.

En concreto, nos hemos planteado estudiar la propiedad de terminación de sistemas de reescritura de términos a partir de los datos de la *Termination Competition* aplicando técnicas de minería de datos, para ello hemos realizado lo siguiente:

- Extraer los datos y prepararlos para la aplicación de técnica de minería de datos.
- Definir el conjunto de propiedades con las que se representaran los TRS.
- Implementar varios experimentos usando diferentes configuraciones (supervisado, semi-supervisado) intentando tener en cuenta las características intrínsecas del problema (clases desbalanceadas).

Los mejores resultados se han obtenido en el último experimento (enfoque semi-supervisado), usando las clases 'terminante' y 'no terminante' y balanceando las clases.

Nacido este proyecto de la idea de obtener un sistema de ayuda para alguna de las herramientas de demostración, los resultados obtenidos dan pie a pensar que es posible la determinación de la terminación de los programas con simples propiedades de los mismos haciendo uso de técnicas de minería de datos. Este estudio aporta a la investigación de la terminación de los programas unos primeros resultados de la experimentación haciendo uso de técnicas de minería de datos. Además, este aporta un completa metodología de trabajo para este contexto de aplicación, siendo el código generado reutilizable y adaptable.
Este trabajo abre un amplio abanico de posibilidades de estudio.

El estudio de los valores de terminación ofrecidos por los demostradores. Se ha visto que para un mismo sistema de reescritura, distintos demostradores ofrecen distintos resultados. Sería un gran aporte un estudio exhaustivo de los datos de la terminación de los sistemas ya vistos que permita saber cual es el correcto.

La inclusión de mas propiedades de los sistemas de reescritura. Las propiedades usadas en este trabajo para definir un sistema de reescritura son muy básicas y es por ello que existe solape entre las clases; solape que se ha podido ver en los análisis expuestos de algunas de las propiedades.

Optimización de los modelos. Para este trabajo se ha usado un conjunto de modelos de clasificación. Seria interesante ver el comportamiento de los datos ante otro tipo de modelos o incluso ante otras configuraciones para los ya usados.

Valga la redundancia del término, pero en conclusión, si bien no se han obtenido resultados perfectos, estos describen una situación idónea de estudio y de posibles mejoras de los mismos. No obstante, seria también interesante adaptar alguno de los modelos aquí expuestos a alguna herramienta de demostración y ver los resultados que se obtienen.

Apéndice A

Código preparación de datos

A.1. Tratamiento inicial de la base de datos.

```
1  # !/usr/bin/python
2  import csv, os, re,psycopg2,sys,string
3  def readCSVtoBDD(file,modificadorCampos):
4      csvfile=open(file, 'rb')
5      spamreader = csv.reader(csvfile, delimiter=',', quotechar='"')
6      a = next(spamreader)
7      letreros,tpdbfile = [],a[0]
8      letreros.append(a[0])
9      for b in xrange(1,len(a)):
10         letreros.append(str(a[b])+' '+modificadorCampos)
11     for i in spamreader:
12         valores=[]
13         valores.append(str(i[0]))
14         for x in xrange(1,len(i)):
15             valores.append(str(i[x]))
16         insertarDatosBDD(letreros,valores,0)
17 def insertarDatosBDD(campos, valores, nComp):
18     try:
19         conn = psycopg2.connect("dbname='trs_tfg' user='sigma' host='localhost' password
               ='3535872'")
20         cursor = conn.cursor()
21         count = cursor.execute("""SELECT tpfile FROM trs WHERE tpfile LIKE '%s'"""%
               valores[nComp])
22         scount = cursor.fetchone()
23         if scount>0:
24             sent = "UPDATE trs SET"
25             for i in xrange(1,len(campos)):
26                 sent = sent +' "'+campos[i]+'"=\''+valores[i]+'\','
27             sent = sent[:-1]+' WHERE tpfile=\''+valores[0]+'\';'
28         else:
29             izq="INSERT INTO trs (tpfile";
30             der=" VALUES ('"+valores[0]+"'";
31             for x in xrange(1,len(campos)):
32                             if not repr(valores[x])=="'':
33                                         izq+=', "'+str(campos[x])+'"'
34                     der+=", '"+str(valores[x])+"'"
35                         sent = izq+")"+der+");"
36             try:
37                 cursor.execute("INSERT INTO proptrs (tpfile) VALUES ('"+valores[0]+"');
                       ")
38                         except Exception,e:
39                 print "No se ha podido ejecutar el insert en la tabla proptrs"
40                 print "Exception: "+str(e)
41         try:
42             cursor.execute(sent)
43         except Exception, e:
44             print "No se ha podido ejecutar el insert/update en la tabla trs"
```

```
45              print "Exception:␣"+str(e)
46          conn.commit()
47          cursor.close ()
48      except Exception, e:
49          print >> sys.stderr, "Exception:␣%s" % str(e)
50          sys.exit(1)
51  def colextra():
52          try:
53                  conn = psycopg2.connect("dbname='trs_tfg'␣user='sigma'␣host='localhost'␣
                        password='3535872'")
54                  cursor = conn.cursor()
55                  cursor.execute("""SELECT␣*␣FROM␣trs""")
56                  a = list(cursor)
57                  for row in a:
58                          yes, mayb, no, res= False, False, False, "NO"
59                          for r in row:
60                                  if r =="YES": yes = True
61                                  if r =="MAYBE": mayb = True
62                                  if r =="TIMEOUT": mayb = True
63                                  if r =="NO": no = True
64                          if mayb: res = "MAYBE"
65                          if no: res= "NO"
66                          if yes: res = "YES"
67                          sent = 'UPDATE␣trs␣SET␣"Res"=\''+res+'\'␣WHERE␣tpfile␣=\''+fich+'
                                \';'
68                          try:
69                                  cursor.execute(sent)
70                          except Exception, e:
71                                  print "No␣se␣ha␣podido␣ejecutar␣el␣update␣en␣la␣tabla␣trs
                                        "
72                                  print "Exception:␣"+str(e)
73                  conn.commit()
74                  cursor.close()
75          except Exception, e:
76              print >> sys.stderr, "Exception:␣%s" % str(e)
77                  sys.exit(1)
78  path="./CSV_RESULTADOS"
79  for root, dirs, files in os.walk(path, topdown=False):
80      for name in files:
81          aux=os.path.join(root, name)
82          year=str(root).split("/")[-1];
83          if (aux.endswith(".csv")):
84              readCSVtoBDD(aux,year)
85  colextra()
```

Código A.1: Código Python para la inserción de los resultados de la competición.

A.2. Obtención de propiedades.

```
1  module Main where
2  import Data.Rewriting.Rules.Ops
3  import Data.Rewriting.Rule.Type
4  import Data.Rewriting.Problem
5  import qualified Data.Rewriting.Term as T
6  import qualified Data.Rewriting.Term.Ops as Tops
7  import qualified Data.Rewriting.Rule.Ops as Rops
8  import Control.Monad ( forM, void)
9  import System.Directory
10 import System.FilePath
11 import Data.List
12
13 main = void $ do
14          x  <- getRecursiveContents "./tpdb-8.0/TRS"
15          putStrLn "<ini>"
16          inicio x
17          putStrLn "</ini>"
18
19 inicio [] = print ""
20 inicio (x:[]) = do
21          putStrLn "<trs>"
```

```
22          obtencion x
23          putStrLn "</trs>"
24  inicio (x:xs) = do
25          putStrLn "<trs>"
26          obtencion x
27          putStrLn "</trs>"
28          inicio xs
29
30  obtencion y = do
31          x <- parseFileIO y
32          putStrLn $ "<File>"++y++"</File>"
33          putStrLn $ "<Depth>"++(show $ maxProfSubterm $ union (getLhss x) (getRhss x)) ++"
                </Depth>"
34          putStrLn $ "<Strategy>"++(show $ getStrategy x)++"</Strategy>"
35          putStrLn $ "<StartTerms>"++(show $ getStartTerms x)++"</StartTerms>"
36          putStrLn $ "<nSymbols>"++(show $ (length (getSymbols x)))++"</nSymbols>"
37          putStrLn $ "<isRecursive>"++(show $ castToInt(isRecursive x))++"</isRecursive>"
38          putStrLn $ "<isShallow>"++(show $ castToInt(isShallowRule (getAllRules x)))++"</
                isShallow>"
39          putStrLn $ "<isLeftLinear>"++(show $ castToInt(isLeftLinear (getAllRules x)))++"
                </isLeftLinear>"
40          putStrLn $ "<isLinear>"++(show $ castToInt(isLinear (getAllRules x)))++"</
                isLinear>"
41          putStrLn $ "<isRightLinear>"++(show $ castToInt(isRightLinear (getAllRules x)))++
                "</isRightLinear>"
42          putStrLn $ "<isCollapsing>"++(show $ castToInt(isCollapsing (getAllRules x)))++"
                </isCollapsing>"
43          putStrLn $ "<isDuplicating>"++(show $ castToInt(isDuplicating (getAllRules x)))++
                "</isDuplicating>"
44          putStrLn $ "<isGround>"++(show $ castToInt(isGround (getAllRules x)))++"</
                isGround>"
45          putStrLn $ "<isLeftGround>"++(show $ castToInt(isLeftGround (getAllRules x)))++"
                </isLeftGround>"
46          putStrLn $ "<isRightGround>"++(show $ castToInt(isRightGround (getAllRules x)))++
                "</isRightGround>"
47          putStrLn $ "<isErasing>"++(show $ castToInt(isErasing (getAllRules x)))++"</
                isErasing>"
48          putStrLn $ "<isCreating>"++(show $ castToInt(isCreating (getAllRules x)))++"</
                isCreating>"
49
50  castToInt:: Bool -> Int
51  castToInt True = 1
52  castToInt False = 0
53
54  getStrategy:: Problem t t1 -> Strategy
55  getStrategy ( Problem a b c d e f g ) = b
56
57  getStartTerms:: Problem t t1 -> StartTerms
58  getStartTerms ( Problem a b c d e f g ) = a
59
60  getTheory:: Problem t t1 -> Maybe [Theory t t1]
61  getTheory ( Problem a b c d e f g ) = c
62
63  getRules:: Problem t t1 -> RulesPair t t1
64  getRules ( Problem a b c d e f g )= d
65
66  getVariables:: Problem t t1 -> [t1]
67  getVariables ( Problem a b c d e f g ) = e
68
69  getSymbols:: Problem t t1 -> [t]
70  getSymbols ( Problem a b c d e f g )  = f
71
72  getComments:: Problem t t1 -> Maybe String
73  getComments ( Problem a b c d e f g ) = g
74
75  getLhss x = lhss $ getAllRules x
76  getRhss x = rhss $ getAllRules x
77  getAllRules x = allRules $ getRules x
78
79  headA [] = []
80  headA (x:xs) =  x
81
82  checkRecursive [] [] = False
```

```
checkRecursive (x:xs) (k:ks) = x `elem` (Tops.funs k) || checkRecursive xs ks
isRecursive x = checkRecursive [y | z<-getLhss x, y <-[headA $ Tops.funs z]] (getRhss x)

maxProf (T.Var _ ) = 1
maxProf (T.Fun _ xs) = 1 + maxProfSubterm xs
maxProfSubterm [] = 0
maxProfSubterm (t:ts) = foldr max (maxProf t) (map maxProf ts)

isShallowRule:: [Rule f v] -> Bool
isShallowRule r =  (isShallowTerms $ lhss r) && (isShallowTerms $ rhss r)

isShallowTerms::[T.Term f v] -> Bool
isShallowTerms [] = True
isShallowTerms (x:xs) = isShallowTerm x && isShallowTerms xs

isShallowTerm::T.Term f v -> Bool
isShallowTerm t
        | Tops.isVar(t) = True
        | otherwise = and ( map (\t -> null (Tops.vars t)) (varsOut $ Tops.properSubterms
            t))
        where varsOut  = foldr  (\t l -> if Tops.isVar(t) then l else t:l) []

getRecursiveContents :: FilePath -> IO [FilePath]
getRecursiveContents topdir = do
        names <- getDirectoryContents topdir
        let properNames = filter (`notElem` [".", ".."]) names
        paths <- forM properNames $ \name -> do
                let path = topdir </> name
                isDirectory <- doesDirectoryExist path
                if isDirectory
                        then getRecursiveContents path
                        else return [path]
        return (concat paths)
```

Código A.2: Código Haskell para la obtención de las propiedades de ficheros .trs y creación de formato .xml

A.3. Inserción de propiedades en base de datos.

```
#!/usr/bin/python
import os, re, psycopg2, sys
from xml.dom import minidom
def obtenerDato(et,trs):
        dato = trs.getElementsByTagName(et)[0]
        return dato.firstChild.data
def insertarDatosBDD(campos, valores, nComp):
    try:
        conn = psycopg2.connect("dbname='trs_tfg' user='sigma' host='localhost' password
            ='3535872'")
        cursor = conn.cursor()
        count = cursor.execute("""SELECT tpfile FROM trs WHERE tpfile LIKE '%s'""" %
            valores[nComp])
        scount = cursor.fetchone()
        if scount>0:
            sent = "UPDATE proptrs SET"
            for i in xrange(1,len(campos)):
                sent = sent +' "'+campos[i]+'"=\''+valores[i]+'\','
            sent = sent[:-1]+' WHERE tpfile=\''+valores[0]+'\';'
                        try:
                                cursor.execute(sent)
                        except Exception, e:
                                print "No se ha podido ejecutar el update en la tabla
                                    proptrs"
                                print "Exception: "+str(e)
                else:
                        print "El TRS: "+str(valores[nComp])+" no ha sido testado en
                            ninguna competicion"
        conn.commit()
        cursor.close ()
    except Exception, e:
```

```
28          print >> sys.stderr, "Exception: %s" % str(e)
29          sys.exit(1)
30
31  campos = ["tpfile","strategy","startTerms","isLinear","isLeftLinear","isRightLinear","
        isGround","isLeftGround","isRightGround","isCreating","isDuplicating","isCollapsing",
        "isErasing","numsimb","depth", "isShallow","isRecursive" ]
32  doc = minidom.parse("datos")
33  ini = doc.getElementsByTagName("ini")
34  trsS = doc.getElementsByTagName("trs")
35  i = 0
36  for trs in trsS:
37          l = []
38          l.append(obtenerDato("File", trs)[2:-4])
39          l.append(obtenerDato("Strategy", trs))
40          l.append(obtenerDato("StartTerms",trs))
41          l.append(obtenerDato("isLinear",trs))
42          l.append(obtenerDato("isLeftLinear",trs))
43          l.append(obtenerDato("isRightLinear",trs))
44          l.append(obtenerDato("isGround", trs))
45          l.append(obtenerDato("isLeftGround", trs))
46          l.append(obtenerDato("isRightGround", trs))
47          l.append(obtenerDato("isCreating", trs))
48          l.append(obtenerDato("isDuplicating",trs))
49          l.append(obtenerDato("isCollapsing", trs))
50          l.append(obtenerDato("isErasing", trs))
51          l.append(obtenerDato("nSymbols", trs))
52          l.append(obtenerDato("Depth",trs))
53          l.append(obtenerDato("isShallow",trs))
54          l.append(obtenerDato("isRecursive",trs).lower())
55          insertarDatosBDD(campos,l,0)
```

Código A.3: Código Python para la inserción de las propiedades en la base de datos.

A.4. Arreglo de datos faltantes y inconsistencias.

```
1  DELETE FROM trs WHERE tpfile IN (SELECT tpfile FROM proptrs WHERE depth IS NULL);
2  DELETE FROM proptrs WHERE depth IS NULL;
3  DELETE FROM trs WHERE tpfile='tpdb-8.0/TRS/Relative_05/rtL-pwl.xml';
4  DELETE FROM trs WHERE tpfile='tpdb-8.0/TRS/Transformed_CSR_04/Ex1_2_Luc02c_L.xml';
5  DELETE FROM trs WHERE tpfile='tpdb-8.0/TRS/Transformed_CSR_04/Ex4_7_77_Bor03_L.xml';
6  DELETE FROM trs WHERE tpfile='tpdb-8.0/TRS/Mixed_relative_TRS/gcd.xml';
7  DELETE FROM trs WHERE tpfile='tpdb-8.0/TRS/Mixed_relative_TRS/rt-rw4.xml';
8  DELETE FROM trs WHERE tpfile='tpdb-8.0/TRS/Relative_05/rt2-8.xml';
9  DELETE FROM proptrs WHERE tpfile='tpdb-8.0/TRS/Relative_05/rtL-pwl.xml';
10 DELETE FROM proptrs WHERE tpfile='tpdb-8.0/TRS/Transformed_CSR_04/Ex1_2_Luc02c_L.xml';
11 DELETE FROM proptrs WHERE tpfile='tpdb-8.0/TRS/Transformed_CSR_04/Ex4_7_77_Bor03_L.xml';
12 DELETE FROM proptrs WHERE tpfile='tpdb-8.0/TRS/Mixed_relative_TRS/gcd.xml';
13 DELETE FROM proptrs WHERE tpfile='tpdb-8.0/TRS/Mixed_relative_TRS/rt-rw4.xml';
14 DELETE FROM proptrs WHERE tpfile='tpdb-8.0/TRS/Relative_05/rt2-8.xml';
```

Código A.4: *Query* SQL para el arreglo de datos faltantes.

A.5. Experimento R: Enfoque supervisado.

```
1  library(DBI);
2  library(RPostgreSQL);
3  library(caret);
4  library(pROC);
5  library(unbalanced);
6
7  principal<-function(yes,no,maybe,o,metodo2){
8          if(grepl("SMOTE",o)){
9                  trainData2<-rbind(yes,no);
10                 trainData2<-sampling(trainData2,"SMOTE");
11                 no<-subset(trainData2,Res==1);
12                 trainData2<-rbind(yes,maybe);
13                 trainData2<-sampling(trainData2,"SMOTE");
```

```
14                  maybe<-subset(trainData2,Res==1);
15                  maybe$Res<-2;
16                  trainData2<-rbind(yes,no,maybe);
17          }
18          if(grepl("TOMEK",o) || grepl("ENN",o)){
19                  maybe$Res<-1;
20                  trainData2<-rbind(yes,no);
21                  trainData2<-sampling(trainData2,o);
22                  yes2<-subset(trainData2,Res==0);
23                  trainData2<-rbind(yes2,maybe);
24                  trainData2<-sampling(trainData2,o);
25                  yes2<-subset(trainData2,Res==0);
26                  maybe$Res<-2;
27                  trainData2<-rbind(yes2,no,maybe);
28          }
29          if(o=="NORMAL"){
30                  maybe$Res<-2;
31                  trainData2<-rbind(yes,no,maybe);
32          }
33          trainData2$Res<-factor(trainData2$Res);
34          modelo<-creacionModelo(trainData2,metodo=metodo2);
35          return(modelo);
36  }
37
38  creacionModelo<-function(datos,metodo2){
39          ctrl <- trainControl(method="repeatedcv",repeats=10,number=10,savePred=T,
                classProb=T);
40          if(metodo2=="KNN"){
41                  modelo<- train(Res~.,data = datos, method="knn",tuneLength = 8,trControl=
                        ctrl,metric="Kappa");
42          }
43          else if(metodo2=="RNA"){
44                  modelo<- train(Res~.,data = datos,method = "nnet", maxit = 1000,
                        trControl = ctrl, trace=F,metric="Kappa");
45          }
46          else if(metodo2=="TREE"){
47                  modelo<- train(Res~.,data = datos, method="J48", trControl=ctrl,
                        tuneLength = 8,metric="Kappa");
48          }
49          else{
50                  modelo<- train(Res~.,data = datos, method="svmLinear",tuneGrid = data.
                        frame(.C = c(.25, .5, 1)), tuneLength = 8, verbose=TRUE,trControl=
                        ctrl,metric="Kappa");
51          }
52          return(modelo);
53  }
54
55  sampling<-function(datos, metodo = NULL){
56           datos2<-datos;
57           if(!is.null(metodo)){
58                  n<-ncol(datos2);
59                  output<-factor(datos2$Res);
60                  input<-datos2[,-n];
61                  if(metodo=="SMOTE"){
62                          data2<-ubSMOTE(X=input,Y=output);
63                  }
64                  else if(grepl("TOMEK",o)){
65                          data2<-ubTomek(X=input,Y=output);
66                  }
67                  else if(grepl("ENN",o)){
68                          data2<-ubENN(X=input,Y=output);
69                  }
70                  datos2 <- cbind(data2$X,data2$Y);
71           }
72           colnames(datos2)[15]<-"Res";
73           datos2$Res<-as.numeric(as.character(datos2$Res));
74           return(datos2);
75  }
76  oVr<-function(method="cv",k=1,number=1,datas=NULL,labelss=NULL){
77          auc<-c();
78          if(method=="cv"){       number=1;       }
79          for(i in 1:number){
80                  auc[i]<-as.numeric(crv(datas,labelss,k));
```

```
81          }
82          return(format(mean(auc, na.rm=TRUE),digits = 4));
83  }
84  crv<-function(datas=NULL,labelss=NULL,k=1){
85          auc<-c();
86          l<-createFolds(datas, k = 10, list = TRUE, returnTrain = FALSE)
87          for (i in 1:k){
88                  Test    <- datas[l[[i]]];
89                  TestL   <- labelss[l[[i]]];
90                  auc[i]  <- as.numeric(multiclass.roc(Test,as.integer(TestL))$auc);
91          }
92          return(format(mean(auc, na.rm=TRUE),digits = 4));
93  }
94  driverS<-"PostgreSQL";
95  hostS<-"localhost";
96  dbnameS<-"trs_tfg";
97  userS<-"sigma";
98  pass<-"3535872";
99  semilla=3535;
100
101 drv <- dbDriver(driverS);
102 con <- dbConnect(drv,host=hostS, dbname=dbnameS, user=userS, password=pass);
103 crx<-dbGetQuery(con,'SELECT␣proptrs.numsimb,␣proptrs."depth",␣proptrs."isLeftLinear",␣
        proptrs."isLinear",␣proptrs."isRightLinear",␣proptrs."isCollapsing",␣proptrs."
        isDuplicating",␣proptrs."isGround",␣proptrs."isLeftGround",␣␣proptrs."isRightGround",
        ␣proptrs."isErasing",␣proptrs."isCreating",␣proptrs."isShallow",␣proptrs."isRecursive
        ",␣trs."Res"␣FROM␣public.proptrs,␣public.trs␣WHERE␣trs.tpfile␣=␣proptrs.tpfile');
104 crx <- as.data.frame(crx);
105
106 for(i in seq(1:(length(crx[1,])-1))){ crx[,i]<-strtoi(crx[,i]);}
107 yes<-subset(crx,Res=="YES");
108 yes$Res<-0;
109 no<-subset(crx,Res=="NO");
110 no$Res<-1;
111 maybe<-subset(crx,Res=="MAYBE");
112 maybe$Res<-1;
113
114 for(metodo in c("SVM","TREE","RNA", "KNN")){
115         for(o in c("NORMAL","SMOTE","TOMEK","SMOTE-TOMEK","SMOTE-ENN")){
116                         set.seed(semilla);
117                         modelo<-principal(yes,no,maybe,o,metodo);
118                         print(metodo);
119                         print(o);
120                         print("*****************************************");
121                         print("Resultados");
122                         print(modelo);
123                         print("AUC");
124                         res<-predict(modelo, modelo$trainingData);
125                         set.seed(semilla);
126                         print(oVr(method="repeated-cv",k=10,number=10,datas=modelo$
                                trainingData$.outcome, labelss=as.integer(res)));
127                         print("Matriz␣de␣confusion␣datos␣iniciales");
128                         maybe2<-maybe;
129                         maybe2$Res<-2;
130                         trainData2<-rbind(maybe2,no,yes);
131                         t1<- table(predict(modelo,trainData2),truth=trainData2$Res);
132                         resultados<-confusionMatrix(t1);
133                         print(resultados);
134                         print("#####################################");
135         }
136 }
```

Código A.5: Experimento R: Enfoque supervisado.

A.6. Experimento R: Enfoque semi-supervisado.

```
1  library(DBI);
2  library(RPostgreSQL);
3  library(caret);
4  library(unbalanced);
```

```
5  library(pROC);
6  library(cvAUC);
7
8  svm.summary<-function(data, lev = NULL, model = NULL){
9          a<-defaultSummary(data, lev, model)
10         b<-twoClassSummary(data, lev, model)
11         out<-c(a,b)
12         out
13 }
14 principal<-function(datos,over,metodo2){
15         trainData2 <- datos;
16         if(grepl("SMOTE",over)){
17                 trainData2<-sampling(trainData2,"SMOTE");
18         }
19         if(grepl("TOMEK",over)){
20                 trainData2<-sampling(trainData2,"TOMEK");
21         }
22         else if(grepl("ENN",over)){
23                 trainData2<-sampling(trainData2,"ENN");
24         }
25         if(over=="NORMAL"){
26                 trainData2<-sampling(trainData2);
27         }
28         modelo<-creacionModelo(trainData2,metodo=metodo2);
29         return(modelo);
30 }
31 creacionModelo<-function(datos,metodo2){
32         ctrl <- trainControl(method = "repeatedcv", number = 10,repeats=10, savePred=T,
               classProb=T,summaryFunction=svm.summary);
33         if(metodo2=="KNN"){
34                 modelo<- train(Res~.,data = datos, method="knn",trControl=ctrl,tuneLength
                        = 8, metric="Kappa");
35         }
36         else if(metodo2=="RNA"){
37                 modelo<- train(Res~.,data = datos,method = "nnet", maxit = 1000,
                        trControl = ctrl, trace=F,metric="Kappa");
38         }
39         else if(metodo2=="TREE"){
40                 modelo<- train(Res~.,data = datos, method="J48",trControl=ctrl,tuneLength
                        = 8,metric="Kappa");
41         }
42         else{
43                 modelo<- train(Res~.,data = datos, method="svmLinear", verbose=TRUE,
                        tuneGrid = data.frame(.C = c(.25, .5, 1)),tuneLength = 8,trControl=
                        ctrl,metric="Kappa");
44         }
45         return(modelo);
46 }
47 # Definicion de funcion sampling
48 sampling<-function(datos, metodo = NULL){
49          datos2<-datos;
50          if(!is.null(metodo)){
51                 n<-ncol(datos2);
52                 output<-factor(datos2$Res);
53                 input<-datos2[,-n];
54                 if(metodo=="SMOTE"){
55                         data2<-ubSMOTE(X=input,Y=output);
56                 }
57                 else if(metodo=="TOMEK"){
58                         data2<-ubTomek(X=input,Y=output);
59                 }
60                 else if(metodo=="ENN"){
61                         data2<-ubENN(X=input,Y=output);
62                 }
63                 datos2 <- cbind(data2$X,data2$Y);
64          }
65          colnames(datos2)[15]<-"Res";;
66          datos2$Res<-factor(datos2$Res);
67          return(datos2);
68 }
69
70 driverS<-"PostgreSQL";
71 hostS<-"localhost";
```

```
72  dbnameS<-"trs_tfg";
73  userS<-"sigma";
74  pass<-"3535872";
75  semilla=3535;
76
77  drv <- dbDriver(driverS);
78  con <- dbConnect(drv,host=hostS, dbname=dbnameS, user=userS, password=pass);
79  crx<-dbGetQuery(con,'SELECT␣proptrs.numsimb,␣proptrs."depth",␣proptrs."isLeftLinear",␣
        proptrs."isLinear",␣proptrs."isRightLinear",␣proptrs."isCollapsing",␣proptrs."
        isDuplicating",␣proptrs."isGround",␣proptrs."isLeftGround",␣␣proptrs."isRightGround",
        ␣proptrs."isErasing",␣proptrs."isCreating",␣proptrs."isShallow",␣proptrs."isRecursive
        ",␣trs."Res"␣FROM␣public.proptrs,␣public.trs␣WHERE␣trs.tpfile␣=␣proptrs.tpfile');
80  crx <- as.data.frame(crx);
81  dbDisconnect(con);
82  dbUnloadDriver(drv);
83
84  for(i in seq(1:(length(crx[1,])-1))){ crx[,i]<-strtoi(crx[,i]);}
85  yes<-subset(crx,Res=="YES");
86  yes$Res<-0;
87  no<-subset(crx,Res=="NO");
88  no$Res<-1;
89
90  maybe<-subset(crx,Res=="MAYBE");
91
92  for(metodo in c("SVM","TREE","RNA", "KNN")){
93          for(o in c("NORMAL","SMOTE","TOMEK","SMOTE-TOMEK","SMOTE-ENN")){
94                          set.seed(semilla);
95                          maybeM<-maybe;
96                          trainData<-rbind(yes,no);
97                          modelo<-principal(trainData,o,metodo);
98                          print("*****************************************");
99                          print(metodo);
100                         print(o);
101                         print("*****************************************");
102                         print("EVALUACION␣MODELO␣DE␣DOS␣CLASES");
103                         print(modelo);
104
105                         res<-predict(modelo,trainData);
106                         t1<- table(res,truth=trainData$Res);
107                         resultados<-confusionMatrix(t1)
108                         print(resultados);
109
110                         print("AUC");
111                         res<-predict(modelo,modelo$trainingData);
112                         print(cvAUC(as.integer(res),modelo$trainingData$.outcome,folds
                                =10)$cvAUC)
113
114                         print("EVALUACION␣MODELO␣CON␣REETIQUETADO␣DE␣CLASES");
115                         maybePredict<-predict(modelo,maybeM);
116                         maybeM$Res<-maybePredict;
117                         trainData <-rbind(maybeM,no,yes);
118                         modelo<-principal(trainData,o,metodo);
119                         print(modelo);
120                         trainData2<-rbind(no,yes);
121                         res<-predict(modelo,trainData2);
122
123                         t1<- table(res,truth=trainData2$Res);
124                         resultados<-confusionMatrix(t1)
125                         print(resultados);
126
127                         print("AUC");
128                         res<-predict(modelo,modelo$trainingData);
129                         print(cvAUC(as.integer(res),modelo$trainingData$.outcome,folds
                                =10)$cvAUC)
130                         print("####################################");
131         }
132 }
```

Código A.6: Experimento R: Enfoque semi-supervisado.

Bibliografía

[1] T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236(1):133–178, 2000.

[2] F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, New York, NY, USA, 1998. ISBN 0-521-45520-0.

[3] G. E. Batista, R. C. Prati, and M. C. Monard. A study of the behavior of several methods for balancing machine learning training data. *ACM Sigkdd Explorations Newsletter*, 6(1):20–29, 2004.

[4] G. A. Betancourt. Máquinas de soporte vectorial (svm). *Scientia Et Technica*, XI: 67–72, 2005.

[5] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16(1): 321–357, 2002.

[6] J. Conway, D. Eddelbuettel, T. Nishiyama, S. K. Prayaga, and N. Tiffin. *RPostgreSQL: R interface to the PostgreSQL database system*, 2013. URL `http://CRAN.R-project.org/package=RPostgreSQL`. R package version 0.4.

[7] B. Cook. Principles of program termination. *Engineering Methods and Tools for Software Safety and Security*, 22:161, 2009.

[8] A. Dal Pozzolo, O. Caelen, S. Waterschoot, and G. Bontempi. Racing for unbalanced methods selection. In *Intelligent Data Engineering and Automated Learning–IDEAL 2013*, pages 24–31. Springer, 2013.

[9] N. Dershowitz. Termination of rewriting. *Journal of symbolic computation*, 3(1): 69–115, 1987.

[10] B. Felgenhauer, M. Avanzini, and C. Sternagel. A haskell library for term rewriting. *arXiv preprint arXiv:1307.2328*, 2013.

[11] M. A. Feliú Gabaldón. *Logic-based techniques for program analysis and specification synthesis*. PhD thesis, Universidad Politécnica de Valencia, 2013.

[12] V. Ganganwar. An overview of classification algorithms for imbalanced datasets. *International Journal of Emerging Technology and Advanced Engineering*, 2(4):42–47, 2012.

[13] J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Automated termination proofs with aprove. In *Rewriting Techniques and Applications*, pages 210–220. Springer, 2004.

[14] B. Gramlich. *Termination and confluence properties of structured rewrite systems*. Universitat Kaiserslautern. Fachbereich Informatik, 1996.

[15] J. Hernández Orallo, M. J. Ramírez Quintana, and C. Ferri Ramírez. Introducción a la minería de datos. *Editorial Pearson Educación SA, Madrid*, 2004.

[16] J. W. Klop. Term rewriting systems, 1992.

[17] M. Kuhn. Building predictive models in r using the caret package. *Journal of Statistical Software*, 28(5):1–26, 2008.

[18] V. López, A. Fernández, S. García, V. Palade, and F. Herrera. An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics. *Information Sciences*, 250:113–141, 2013.

[19] S. Lucas. Practical use of polynomials over the reals in proofs of termination. In *Proceedings of the 9th ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming*, PPDP '07, pages 39–50, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-769-8. doi: 10.1145/1273920.1273927. URL `http://doi.acm.org/10.1145/1273920.1273927`.

[20] S. Lucas and R. Pena. Rewriting techniques for analysing termination and complexity bounds of safe programs. *Proc. Logic-Based Program Synthesis and Transformation, LOPSTR*, 8:43–57, 2008.

[21] C. Marché and H. Zantema. *The Termination Competition*, volume 4533 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2007. ISBN 978-3-540-73447-5.

[22] R. S. I. G. on Databases. *DBI: R Database Interface*, 2014. URL `http://CRAN.R-project.org/package=DBI`. R package version 0.3.1.

[23] M. Perez Marques. *MINERIA DE DATOS. La Metodologia CRIS-DM de IBM. El lenguaje CLEM e IBM SPSS MODELER*. Createspace, 2014. ISBN 1505536782.

[24] A. D. Pozzolo, O. Caelen, and G. Bontempi. *unbalanced: The package implements different data-driven method for unbalanced datasets*, 2014. URL `http://CRAN.R-project.org/package=unbalanced`. R package version 1.1.

[25] X. Robin, N. Turck, A. Hainard, N. Tiberti, F. Lisacek, J.-C. Sanchez, and M. Müller. proc: an open-source package for r and s+ to analyze and compare roc curves. *BMC Bioinformatics*, 12:77, 2011.

[26] A. J. Robinson and A. Voronkov. *Handbook of automated reasoning*, volume 2. Elsevier, 2001.

[27] D. Singh and A. M. Shuaibu. Well-foundedness for termination of term rewriting systems. *Applied Mathematical Sciences*, 7(66):3291–3302, 2013.

[28] A. M. Turing. On computable numbers, with an application to the entscheidungsproblem. *J. of Math*, 58(345-363):5, 1936.

[29] R. Wirth. Crisp-dm: Towards a standard process model for data mining. In *Proceedings of the Fourth International Conference on the Practical Application of Knowledge Discovery and Data Mining*, pages 29–39, 2000.

[30] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.

[31] K. Woodsend and M. Lapata. Text rewriting improves semantic role labeling. *Journal of Artificial Intelligence Research*, pages 133–164, 2014.