**UNIVERSIDAD DE CÓRDOBA**
**ESCUELA POLITÉCNICA SUPERIOR**
**GRADO EN INGENIERÍA INFORMÁTICA**

TRABAJO FIN DE GRADO

# Framework en Python para Problemas de Clasificación Ordinal

Autor: Iván Bonaque Muñoz

---

Directores:
Pedro Antonio Gutiérrez Peña
Javier Sánchez Monedero

# Framework en Python para Problemas de Clasificación Ordinal

Firmado en Córdoba, -

Directores:

Fdo: Dr. Pedro Antonio Gutiérrez Peña  Fdo: D. Javier Sánchez Monedero

Autor:

Fdo: Iván Bonaque Muñoz

**Pedro Antonio Gutiérrez Peña**, Profesor titular de Universidad del Departamento de Informática y Analisis Numérico en la Escuela Politécnica Superior de la Universidad de Córdoba e investigador del Grupo de Investigación AYRNA.

## Informa:

Que el presente Trabajo Fin de Grado titulado *Framework en Python para Problemas de Clasificación Ordinal*, constituye la memoria presentada por **Iván Bonaque Muñoz** para aspirar al título de Graduado en Ingeniería Informática, y ha sido realizado bajo mi dirección en la Escuela Politécnica Superior de la Universidad de Córdoba reuniendo, a mi juicio, las condiciones necesarias exigidas en este tipo de trabajos.

Y para que conste, se expide y firma el presente certificado en Córdoba, -.

El Director:

Fdo: Prof. Dr. D. Pedro Antonio Gutierréz Peña

**Javier Sánchez Monedero**, Investigador asociado a la Universidad de Cardiff e investigador del Grupo de Investigación AYRNA.

## Informa:

Que el presente Trabajo Fin de Grado titulado *Framework en Python para Problemas de Clasificación Ordinal*, constituye la memoria presentada por **Iván Bonaque Muñoz** para aspirar al título de Graduado en Ingeniería Informática, y ha sido realizado bajo mi dirección en la Escuela Politécnica Superior de la Universidad de Córdoba reuniendo, a mi juicio, las condiciones necesarias exigidas en este tipo de trabajos.

Y para que conste, se expide y firma el presente certificado en Córdoba, -.

El Director:

Fdo: D. Javier Sánchez Monedero

# Agradecimientos

# Índice General

# Índice de Figuras

# Índice de Tablas

# Parte I

# Introducción

# Capítulo 1

# Introducción

Este primer capítulo pretende ilustrar al lector, de forma general, acerca del problema que se tratará en el presente Trabajo, así como de los objetivos a alcanzar mediante la realización del mismo.

El cerebro humano, debido a su estructura y funcionamiento, es capaz de hacer parecer triviales tareas difícilmente trasladables al marco de la computación, como puede ser el reconocimiento de clases de objetos mediante la vista. Este tipo de tareas nos es posible realizarlas gracias a nuestra capacidad innata de abstracción. Sin embargo, conforme aumenta la complejidad y el tamaño de la información con la que tratamos, nuestra facultad se ve superada. Es aquí donde sale a relucir la utilidad del aprendizaje automático, o *machine learning*.

El aprendizaje automático es una subrama de la inteligencia artificial, que tiene por misión generar algoritmos y modelos matemáticos para que las computadoras sean capaces de realizar tareas hasta ahora asociadas a la capacidad de abstracción humana, como inferir estructuras subyacentes en los datos, o recordar y predecir patrones. La información con la que tratan consiste en una serie de patrones, siendo cada uno de ellos una abstracción de un elemento real, determinado por valores concretos para una serie de características. La figura 1.1 trata de ilustrar los elementos que forman parte

de este campo al más alto nivel de abstracción.

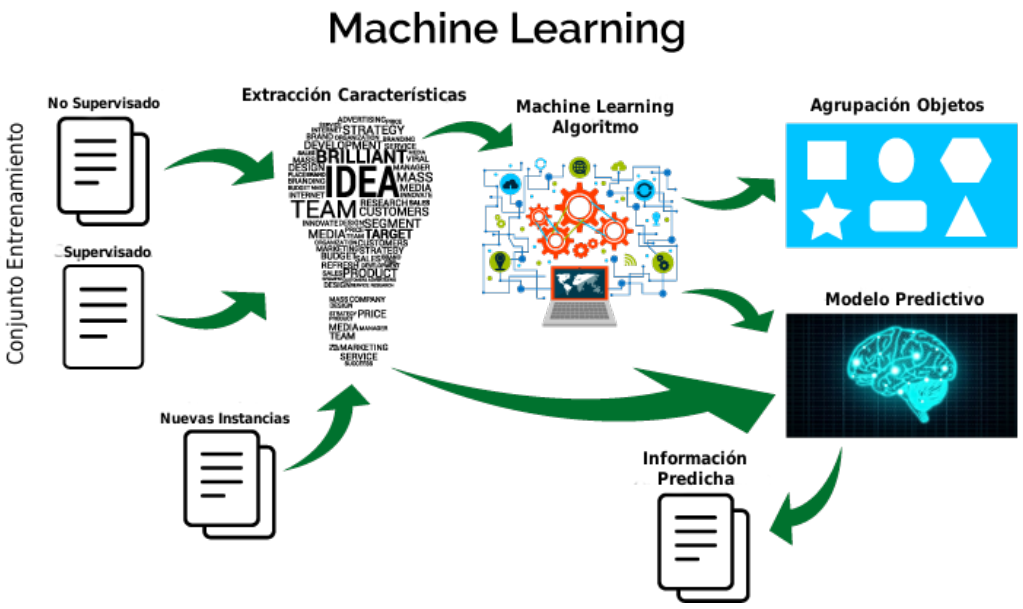


Figura 1.1: Introducción al *machine learning*

Existen múltiples algortimos de aprendizaje automático, pero en función de su finalidad y del tipo de datos con los que tratan, podemos clasificarlos de la siguiente manera [1]:

- Aprendizaje supervisado: Los algoritmos de este tipo intentan aprender la relación que existe entre los valores de las variables independientes de unos ítems de datos y el valor de una variable dependiente, suministrada previamente por un experto. Una vez que el algortimo haya construido una función para aproximar los valores de la variable independiente (fase de entrenamiento), se utilizará para predecir la variable dependiente en instancias para las que desconocemos su valor (fase de test o generalización).

  Los dos casos más estudiados dentro de este tipo de problemas son, o bien aquellos en los que las salidas están restringidas a una serie de valores concretos, pudiendo ser numéricos o nominales, o donde la

salida es un rango de números reales. Estos casos son los de clasificación y regresión, respectivamente.

- Aprendizaje no supervisado: en este caso, los algoritmos únicamente disponen de las características de cada elemento del conjunto de datos, no conociendo el valor de la variable de salida. Estos tipos de problemas suelen consistir en la búsqueda de estructuras dentro de los datos suministrados. En este tipo de aprendizaje se engloba el *clustering* (o análisis clúster), entre otros.

A lo largo de este Trabajo, nos centraremos en un subapartado del aprendizaje supervisado, denominado clasificación ordinal, o regresión ordinal. Los problemas a tratar en este tipo de clasificación tienen una peculiaridad inherente: la existencia de un orden lógico entre las diferentes clases presentes en los datos (valores que puede tomar la variable dependiente). Esto implica que no será lo mismo cometer un error al clasificar un patrón en una clase contigua a la que verdaderamente le corresponde, que hacerlo en una muy alejada en la escala ordinal.

Un ejemplo con el que ilustrar esta clase de problemas es el siguiente: tenemos una base de datos con una gran cantidad de fotografías del rostro de diferentes personas, y queremos construir un clasificador que sea capaz de discernir el grupo de edad al que pertenece cada una de ellas (niño, joven, adulto, anciano), donde cada grupo viene representado por una horquilla de edades. Parece lógico afirmar que, si el algoritmo se equivoca clasificando a un niño como si fuese anciano, habrá incurrido en un error más notable que si clasifica incorrectamente a un joven como adulto. La imagen 1.2 muestra un posible problema de este tipo, donde el modelo debería de clasificar a cada uno de los individuos dentro de una horquilla de edades en base a sus rostros.

Figura 1.2: Ejemplo de imagenes a clasificar

Podemos ver que la clasificación ordinal comparte similitudes tanto con la clasificación nominal, ya que hace uso de etiquetas discretas, como con la regresión, puesto que existe un orden entre las posibles salidas. Estas semejanzas pueden a su vez tomarse como diferencias de un tipo de aprendizaje con respecto del otro. Por todo ello, es conveniente tratar el estudio de este tipo de problemas de forma separada a los anteriormente citados.

Por otra parte, aunque conocemos los modelos teóricos en base a los cuales se construyen los diferentes algoritmos de clasificación, es difícil conocer a priori el comportamiento que estos tendrán al tratar con los conjuntos de datos sobre los que se van a aplicar, teniendo en cuenta además, que en función de los parámetros internos de cada clasificador, los resultados pueden variar ostensiblemente. Por ello, es de gran necesidad, sobre todo en el mundo académico, la comparación de diferentes algoritmos para un mismo problema, con el fin de comprobar que clasificador se adapta mejor a ese problema determinado.

Esta comparación de diferentes clasificadores para distintos conjuntos de datos, puede mostrarse altamente tediosa de no contar con un *software* que ayude a la automatización de dichos experimentos. Para llevar a cabo esta automatización, existen herramientas para tratar con una importante cantidad de algoritmos de aprendizaje supervisado y no supervisado, como *Weka*

[2], sin embargo, para el caso concreto de la clasificación ordinal, no existe un *software* de automatización fácilmente disponible.

Con el propósito de poner fin a este problema de evaluación de métodos, el grupo de investigación *AYRNA* [3] desarrolló un framework para la automatización de experimentos con algoritmos de regresión ordinal, denominado *ORCA* [4]. Este *software* se encuentra codificado en los entornos de programación MATLAB [5] y Octave [6], los cuales, hoy en día, disponen de una menor popularidad para el tratamiento de tareas de Ciencias de la Computación que otros como Python o R. Hablando concretamente de Python, se han desarrollado para él importantes bibliotecas de *software* libre para tratar problemas de computación científica, como son `scikit-learn` [7], `pandas` [8] y `numpy` [9].

El objetivo a conseguir con la realización del presente Trabajo es el de crear un *framework* escrito en el lenguaje de programación Python, que incluya la funcionalidad proporcionada por ORCA, haciendo compatible todo el código con las bibliotecas *software* antes nombradas.

# Capítulo 2

# Definición del Problema

Durante el siguiente capítulo se tratará de dar respuesta a la pregunta: *¿Qué se consigue con la realización del Trabajo de Fin de Grado?*. Para efectuar esta tarea, identificaremos las necesidades que han llevado a la ejecución de este, así como los objetivos que se deben alcanzar con su desarrollo. Para un mejor estudio dividiremos el capítulo en dos subapartados:

- Identificación del problema real: el problema se define desde el punto de vista del usuario final.
- Identificación del problema técnico: el problema se define desde el punto de vista del desarrollador del Trabajo.

## 2.1. Identificación del Problema Real

Como se introdujo en el capítulo previo, el objetivo primordial es el de portar la funcionalidad de un *framework* previamente desarrollado hacia un lenguaje de programación más abierto y actual. Además de la creación del sistema *software*, se desarrollará un algoritmo de clasificación ordinal que se integrará con el mismo. Concretamente, el problema real planteado por el

grupo de investigación AYRNA consiste en los siguientes puntos:

1. Desarrollo de un *framework* implementado en el lenguaje de programación Python, que permita la experimentación con diversos algoritmos de clasificación ordinal, conjuntos de datos y métricas de evaluación del rendimiento, además de almacenar la información de importancia generada, de forma tal que se puedan ejecutar estos experimentos sin mayor esfuerzo por parte del usuario que el de definir un fichero de configuración con una sintaxis sencilla. Podemos dividir este punto en varios subproblemas:

   - Desarrollo de un formato sencillo de configuración para indicarle al *framework* las variables con las que modificar el comportamiento de los experimentos que lleve a cabo.
   - Desarrollo de la parte principal del *framework*, encargada de interpretar los ficheros de configuración y ejecutar el experimento propiamente dicho. Debe implementar el proceso de validación cruzada integrándose con la biblioteca `scikit-learn`, la cual proporciona una serie de utilidades que facilitan enormemente la creación del sistema.
   - Desarrollo de un módulo *software* encargado de almacenar en ficheros la información relevante del experimento, consistente en: métricas que evalúan la aptitud de un modelo para clasificar los patrones pertenecientes a una base de datos, modelos obtenidos durante la validación cruzada y predicciones obtenidas por esos modelos.

2. Codificación de un algoritmo de clasificación ordinal, basado en la descomposición del problema ordinal original en varios subproblemas de clasificación binaria [10]. El clasificador ha de ser compatible con `scikit-learn`.

## 2.2. Identificación del Problema Técnico

Una vez definido el problema real que se busca solucionar, es necesario precisarlo desde una visión más técnica. Para ello, haremos uso de la metodología PDS (*Product Design Specification*), que nos permite hacer un análisis de los principales condicionantes técnicos mediante una serie de cuestiones fundamentales.

### 2.2.1. Funcionamiento

Como el entorno de experimentación a desarrollar se encuentra basado en uno ya existente (ORCA), es necesario que este contenga la funcionalidad necesaria para que pueda actuar como él. Nuestro programa deberá ser capaz de leer bases de datos, almacenadas en un formato específico, y aplicarles a las mismas una serie de algoritmos de clasificación, no siendo necesario que estos sean de regresión ordinal. A su vez, se podrá configurar el comportamiento de cada algoritmo dando valores a los distintos parámetros que estos posean, elemento indispensable para realizar la fase de validación cruzada, donde se seleccionará el valor de cada parámetro que mejores resultados obtenga para el conjunto de datos de entrenamiento.

Durante la ejecución, se irán obteniendo distintas medidas que evalúan la efectividad de cada clasificador para cada uno de los conjuntos de datos. Estos valores serán almacenados para la visualización por parte del usuario junto con otros elementos como los modelos de datos y las predicciones de las etiquetas de clase obtenidas.

Los pasos más generales del funcionamiento de la herramienta son:

- ***Introducción de Datos***: El usuario proporcionará las bases de datos, que podrán o no estar fragmentadas en particiones, mediante ficheros de texto en formato CSV [11]. Todas las bases de datos a usar en un experimento concreto deben estar dentro de una carpeta raíz indicada

en el fichero de configuración.

- ***Definir Configuración***: El usuario debe indicar múltiples factores a través de ficheros de configuración para que el experimento pueda desarrollarse, tales como los conjuntos de datos a utilizar, las diferentes métricas de evaluación del rendimiento, los clasificadores y los distintos valores que pueden tomar sus parámetros, etc.
- ***Ejecución***: Se aplicarán las distintas configuraciones de algoritmos indicadas sobre los conjuntos de datos, obteniendo una serie de métricas previamente escogidas.
- ***Almacenamiento de Resultados***: La herramienta *software* almacenará los resultados en la carpeta seleccionada por el usuario.

### 2.2.2. Entorno

El entorno de programación que se va a utilizar durante el desarrollo de la aplicación está meramente constituido por el editor de texto por defecto de la distribución Debian GNU/Linux, *Gedit*. En él se visualizará y modificará el código escrito en el lenguaje de programación Python. Para la ejecución del *framework* será necesario un intérprete de este lenguaje de programación. El sistema operativo que utilizaremos durante nustro Trabajo es *Debian GNU/Linux*, concretamente la versión 9.8 (stretch), aunque para corroborar el correcto comportamiento multiplataforma del programa, se comprobará la validez del *software* en el sistema operativo *Windows 10*.

Auxiliarmente, también se hará uso del entorno de programación impuesto por MATLAB/Octave, con el objeto de poder ejecutar ORCA, y comprobar el funcionamiento de dicho *framework*.

Los usuarios finales a los que se encuentra dirigida esta aplicación conforman un grupo relativamente reducido, puesto que deben disponer de conocimientos básicos acerca de la temática de la aplicación (algoritmos de clasificación, métricas, etc.), junto con unos mínimos conocimientos informáticos

para ponerlo en marcha, ya que han de utilizar la terminal del sistema, conocer las reglas de los ficheros de configuración, etc.

### 2.2.3. Vida Esperada

La estimación de la vida útil del *software* es difícil de determinar, puesto que este campo de investigación se encuentra en constante evolución, ideándose y codificándose continuamente nuevos algoritmos o métricas que pueden ser mejorados y/o añadidos al *framework*.

Por tanto, suponemos que el presente Trabajo puede servir de base para futuros cambios que añadan funcionalidad al mismo, haciéndolo progresivamente más completo.

### 2.2.4. Ciclo de Mantenimiento

Como se comentó en el punto anterior, la vida útil del programa estará asociada a las futuras revisiones y mejoras que se hagan sobre el mismo, que podrán ser planteadas para futuros Trabajos de Fin de Grado.

La ausencia de una planificación de mantenimiento se debe a las distintas circunstancias que rodean nuestro Trabajo.

### 2.2.5. Competencia

Weka [2] es un *software* que permite realizar tareas de preprocesamiento sobre bases de datos y experimentar con una gran variedad de algortimos de aprendizaje automático, contando con una interfaz gráfica para facilitar su manejo a usuarios menos expertos. El principal inconveniente del que dispone es que únicamente dispone de un clasificador que contemple el orden entre las distintas clases.

Por otra parte nos encontramos con ORCA [10], *framework* implementado en MATLAB del cual nace la iniciativa que impulsa el desarrollo de esta aplicación, que cuenta con una gran cantidad de métodos para tratar con problemas de regresión ordinal, pero cuenta con el inconveniente de no poder disponer de las funcionalidades que ofrecen las distintas bibliotecas de aprendizaje automático con las que cuenta Python.

Asimismo, existe una amplia variedad de algoritmos de clasificación ordinal desarrollados previamente. De entre esta variedad seleccionaremos un conjunto de ellos con los que contrastar el rendimiento de nuestro clasificador.

### 2.2.6. Aspecto Externo

La aplicación será presentada al usuario en un CD-ROM, debido a que es un medio de almacenamiento con unas características idóneas: una capacidad de almacenamiento más que suficiente, durabilidad, resistencia, coste reducido y dimensiones reducidas.

Junto al soporte óptico se entregará el Manual de Usuario, redactado de forma clara y concisa, que tratará de explicar al potencial usuario final de esta aplicación los pormenores de su utilización.

Dentro del CD-ROM se incluirán, en formato PDF, tanto el manual de usuario como el manual técnico.

Asimismo, la aplicación se encontrará alojada en un repositorio en *GitHub* [12], servicio web de almacenamiento en la nube que hace uso de la herramienta de control de versiones *Git* [13], para facilitar el desarrollo colaborativo y evitar posibles pérdidas de información.

### 2.2.7. Estandarización

En relación a la estandarización durante la fase de codificación, se procurará seguir el estándar estudiado durante la carrera a la hora de definir correctamente cada una de las clases de objetos y sus distintas propiedades.

En cuanto al código en sí, se ha adecuado en la medida de lo posible a la guía de estilo de Python, PEP 8 [14]. En ella, se indica la importancia de la facilidad de comprensión del código, así como la de mantenerse consistente con la forma en la que se programa a lo largo del desarrollo. Algunas de las directrices incluidas en la guía son:

- Los nombres de todos los elementos deben ser tan descriptivos como sea posible.
- Los nombres de los paquetes, métodos, funciones y variables deben utilizar minúsculas y guiones bajos.
- Los nombres de clases usan la convención *CapWords*, donde la primera letra de cada palabra que compone el nombre debe estar en mayúsculas.
- Los nombres de métodos y de instancias declarados como no públicos (dentro de una clase) deben estar precedidos de un guión bajo.
- Las constantes se indican mediante letras mayúsculas y guiones bajos.
- La longitud máxima de las líneas de código debería ser inferior a 99 caracteres, mientras que los comentarios de texto no deben exceder las 72.
- No hacer uso de espacios en blanco en determinadas situaciones.

Cabe comentar que, por compatibilidad con la biblioteca `scikit-learn`, los clasificadores tendrán una convención especifica: los nombres de los atributos de las clases principales de estos clasificadores, finalizarán con un guión bajo.

### 2.2.8. Calidad y Fiabilidad

La calidad del Trabajo será controlada por el autor y los directores del mismo. Tras haberse finalizado, será evaluado por el Tribunal designado para este propósito. En la medida de lo posible, se intentarán seguir los estándares de Ingeniería del Software para favorecer la mayoría de los factores que afectan a la calidad de una aplicación.

En lo que respecta a la fiabilidad, la aplicación se diseñará para que, en caso de que el usuario introduzca parámetros inadecuados, esta le informe del error cometido mediante un mensaje en la terminal. Asimismo, cuando se intente leer datos desde un fichero, se notificará en caso de que no se encuentre dicho fichero, o del uso de un formato inadecuado (ya sea en la nomenclatura de los archivos de entrada, o en la forma de representar los datos). En el volcado de los resultados en sus respectivos ficheros, se avisará en caso de error al escribir sobre el sistema.

### 2.2.9. Programa de Tareas

De manera general, el desarrollo se dividirá en las siguientes etapas:

1. ***Fase de preparación:*** En esta primera fase se realizará un estudio teórico, tanto del problema en sí, como de los principios generales y básicos necesarios para abordar el mismo, que incluyen los conceptos de regresión ordinal, así como algunos algoritmos asociados a este tipo de problemas. Además, se estudiará el lenguaje de programación Python y las bibliotecas `scikit-learn`, `numpy` y `pandas`, necesarias para el desarrollo de la aplicación.

2. ***Fase de diseño:*** Una vez analizados y estudiados los requisitos y objetivos del sistema, se procederá a diseñarlo para su posterior codificación. Se analizará la funcionalidad del sistema, el diseño de los datos, el flujo de control que estos seguirán, etc.

3. ***Fase de implementación:*** En esta fase se llevará a cabo la programación del *framework*, tomando como referencia a ORCA en cuanto a la estructura general, además del desarrollo de un algoritmo de clasificación ordinal que se integrará dentro del *framework*. Para crear la aplicación se utilizará el lenguaje de alto nivel Python. Los flujos de entrada y salida se realizarán mediante ficheros de texto mayortariamente.

4. ***Fase de pruebas:*** Etapa durante la cual se validará el correcto funcionamiento de nuestro programa. Esta fase será paralela a la de implementación, puesto que nos permitirá comprobar que las alteraciones realizadas sobre el código del programa no producen resultados indeseados.

5. ***Fase de documentación:*** Comprende la realización de la memoria final del Trabajo de Fin de Grado, así como del manual de usuario.

6. ***Fase de aplicación:*** En esta última fase del Trabajo, se compararán los resultados obtenidos (diferentes métricas de rendimiento de clasificadores) por el clasificador desarrollado, frente a los resultados obtenidos por otros algoritmos de regresión ordinal. Mientras que los resultados del algoritmo a desarrollar se obtendrán mediante el uso del *framework*, nos basaremos en el trabajo contenido en el artículo de investigación [10] para el contraste de resultados.

### 2.2.10. Pruebas

En la fase de pruebas se pretende aportar una visión de las estrategias seguidas en la certificación del *software* y de los resultados obtenidos en las mismas. La estrategia de pruebas a seguir abarca:

- Pruebas de unidad a nivel de módulos.
- Pruebas de integración para comprobar el ensamblado de los módulos.

- Pruebas de validación para probar el software como un conjunto.
- Pruebas de sistema para asegurar el cumplimiento de los objetivos.
- Pruebas de aceptación donde el usuario pueda verificar si el producto se ajusta los requisitos.

Para comprobar el correcto funcionamiento del *software*, se contrastarán los resultados obtenidos por el clasificador desarrollado frente a múltiples algoritmos de regresión ordinal, con respecto a diferentes conjuntos de datos. Esto, a su vez, servirá para comprobar que el programa encargado de gestionar los experimentos responde adecuadamente.

Solo una vez que la aplicación cumpla todos los objetivos para los cuales fue diseñada y no muestre errores en los casos de prueba generados, podremos considerar que la misma es válida.

### 2.2.11. Seguridad

La temática de la aplicación realizada no permite la ejecución de operaciones incorrectas que salgan de sus objetivos determinados.

La privacidad no es un requerimiento, puesto que el tipo de datos que manejará la aplicación, en principio, no son susceptibles de contener información sensible para el usuario o terceras personas.

# Capítulo 3

# Objetivos

El propósito general que se busca conseguir llevando a cabo este Trabajo es el de construir un *framework* para la automatización de experimentos con diferentes configuraciones de algoritmos de clasificación, bases de datos y métricas de evaluación. Además de desarrollar un clasificador para problemas de regresión ordinal.

## 3.1. Objetivos de Formación

Los objetivos de formación del presente Trabajo son:

1. Estudio en profundidad del lenguaje de programación Python.
2. Estudio sobre el manejo de las herramientas para la cienca computacional que proporcionan bibliotecas como `scikit-learn`, `numpy` y `pandas`.
3. Estudio en profundidad sobre el manejo de `sacred` [15], biblioteca de apoyo para la automatización de experimentos computacionales.
4. Estudio en profundidad de la metodología de la programación orientada a objetos.

5. Realizar un estudio teórico de las áreas relacionadas con el Trabajo, incluyendo conceptos de la regresión ordinal y de las distintas métricas para evaluar el rendimiento de un clasificador.

6. Estudio de mecanismos para validar la funcionalidad del *framework* y el rendimiento del clasificador desarrollado sobre varios conjuntos de datos reales, basándonos para ello en los resultados obtenidos previamente por el citado artículo de investigación [10].

7. Profundizar en el aprendizaje de todo el proceso relacionado con la creación de una aplicación y las herramientas de apoyo necesarias, incluyendo la creación del manual de usuario.

## 3.2. Objetivos Operacionales

Los objetivos a cumplir por la aplicación durante su funcionamiento son:

1. Adaptación de un algortimo de regresión ordinal basado en la descomposición de un problema ordinal en múltiples subproblemas de clasificación nominal, implementado de forma que sea compatible con `scikit-learn`. El funcionamiento de este clasificador se encuentra detallado en la sección 4.4

2. Desarrollar un *framework* para facilitar la experimentación con algortimos de clasificación y BBDD (acrónimo de *bases de datos*). Su ejecución constará de distintas fases:

   *a*) Leer la configuración del experimento desde un fichero con una sintaxis relativamente sencilla.

   *b*) Llevar a cabo un proceso de validación cruzada para seleccionar los valores óptimos para unos hiperparámetros (aquella combinación que obtenga los mejores resultados clasificando las particiones de validación). Tanto los hiperparámetros, como los valores a validar

deberán ser especificados por el usuario en el fichero de configuración.

*c*) Obtener las métricas de rendimiento para la fase de entrenamiento y de generalización con el modelo que haya obtenido los mejores resultados durante la fase de validación.

*d*) Almacenar los resultados obtenidos en un formato determinado, que sea fácil de interpretar y manejar.

# Capítulo 4

# Antecedentes

En este capítulo se ilustrará al lector acerca del punto de partida, tanto teórico como tecnológico, desde el cual se da luz verde al desarrollo del Trabajo Fin de Grado tratado en este documento. Este conocimiento nos servirá para especificar los requisitos del sistema que buscamos desarrollar.

## 4.1. ORCA

ORCA (*Ordinal Regression and Classification Algorithms*) es un *framework* desarrollado por el grupo de investigación AYRNA [1], perteneciente al Departamento de Informática y Análisis Numérico de la Universidad de Córdoba.

Esta herramienta, cuyo funcionamiento general se trata de replicar en nuestro *software*, implementa una amplia variedad de métodos de clasificación ordinal y de métricas de evaluación de clasificadores. Se puede utilizar mediante la definición de ficheros de configuración que automatizan la experimentación de algoritmos de clasificación ordinal sobre conjuntos de datos, almacenándose la información resultante de la ejecución del experimento.

[1] Repositorio del proyecto: https://github.com/ayrna/orca

ORCA se encuentra desarrollado en el lenguaje propietario MATLAB, cuyos IDE (Entorno de Desarrollo Integrado) e intérprete cuentan con unas licencias de uso no gratuitas. A pesar de este hecho, el sistema también es compatible con el lenguaje de cómputo numérico GNU Octave, el cual cuenta con una licencia de software libre, en contraposición a MATLAB. Sin embargo, sufre el inconveniente de ser menos flexible y más difícil de utilizar que Python.

## 4.2. Validación Cruzada

Supongamos un problema de clasificación estándar, donde el problema está representado por patrones descritos por una serie de variables de entrada y una etiqueta de clase. Si utilizáramos el conjunto de datos al completo durante la fase de entrenamiento, no podríamos posteriormente comprobar la eficiencia del modelo, puesto que únicamente podríamos utilizar instancias que han sido previamente aprendidas, en mayor o menor medida.

Una solución a esta problemática sería realizar una división en la base de datos en la que utilizásemos, por ejemplo, el 70 % de los patrones para la fase de entrenamiento, en la que se generará el modelo; y el 30 % restante para la fase de validación, donde se comprobará la eficiencia del clasificador con patrones no vistos durante el entrenamiento. Sin embargo, este método denominado *hold-out*, tiene como desventaja la generación de una gran variabilidad en los resultados, ya que estos pueden depender en gran medida de como se haya realizado la partición de los datos.

Como mejora a esta técnica apareció el método de validación cruzada, o *cross-validation*. La validación cruzada es una técnica de evaluación de modelos estadísticos o de aprendizaje automático que permite optimizar estos modelos y a la vez comprobar su rendimiento esperado sobre patrones no vistos previamente, evitando problemas como el sobre-ajuste (*over-fitting*). Existen distintas variantes de este método, pero nosotros nos centraremos

en la validación cruzada de $k$-iteraciones, más conocida como *k-fold cross-validation*.

Este tipo de validación cruzada dividirá el conjunto de datos en $k$ subconjuntos diferentes, usándose uno de ellos como conjunto de prueba (fase de validación), y el resto para construir el modelo (fase de entrenamiento). Manteniéndose los conjuntos inalterados, se repetirá este proceso $k$ veces, tomando en cada iteración a un subconjunto diferente como subconjunto de validación. Este proceso se ilustra en la figura 4.1.

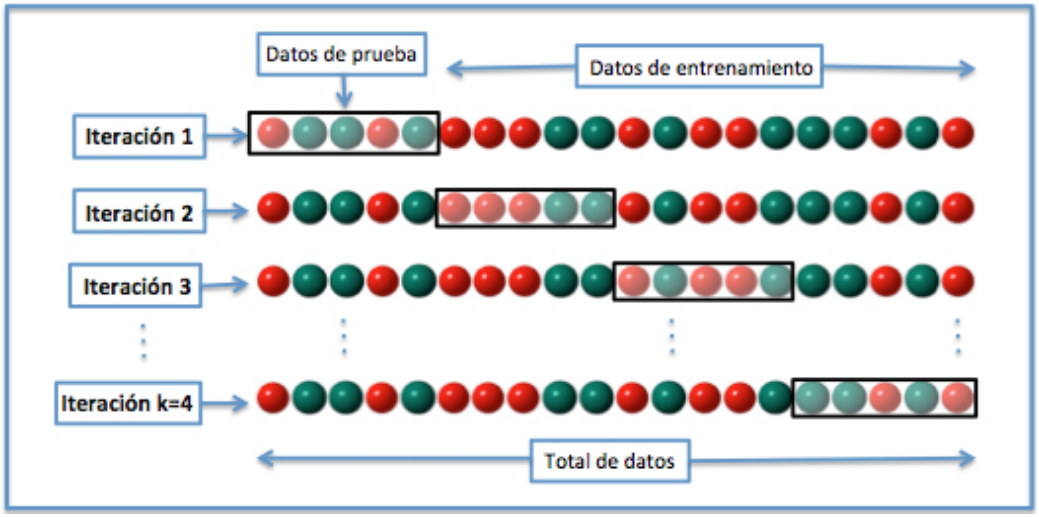


Figura 4.1: Validación cruzada con $k$ iteraciones

En nuestra aplicación haremos un uso mixto de las dos técnicas de validación aquí nombradas. Ya que el programa aceptará que los datos se encuentren previamente divididos en conjuntos de entrenamiento y de test, a la vez que realizará un *k-fold* sobre los datos de entrenamiento para encontrar un modelo lo más óptimo posible.

## 4.3. Regresión Ordinal

En esta sección se pretende expandir el concepto de regresión ordinal previamente introducido en el Capítulo 1. Para este propósito, nos apoyaremos en el estudio [10] previamente realizado por investigadores del grupo AYRNA, en el cual se analiza el estado del arte de la materia en cuestión.

Como se comentó anteriormente, la regresión ordinal es un tipo de aprendizaje supervisado, donde las clases en las que se pueden clasificar los patrones poseen un orden natural entre sí. Este orden constituye información adicional que sería producente tener en cuenta a la hora de construir un modelo de predicción. Asimismo, el coste de los errores cometidos a la hora de clasificar patrones deberá ser relativo a la diferencia, en el orden entre clases, de una y otra etiqueta.

### 4.3.1. Definición del Problema

De manera formal, podemos definir los problemas de regresión ordinal de la siguiente forma:

Dado un conjunto de puntos o patrones pertenecientes a un espacio de variables $k$-dimensional, $x \in \mathcal{X} \subseteq \mathbb{R}^k$, y un conjunto de Q etiquetas o clases, $y \in \mathcal{Y} = \{\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_Q\}$, que categoricen dichos patrones de manera excluyente; el objetivo es el de construir una regla o función, $r : \mathcal{X} \to \mathcal{Y}$, en base a una serie de patrones con etiquetas conocidas, que sea capaz de discernir correctamente la clase para puntos no vistos con anterioridad.

Además, se considera la existencia de un orden natural entre las diferentes etiquetas $\mathcal{C}_1 \prec \mathcal{C}_2 \prec ... \prec \mathcal{C}_Q$, donde $\prec$ representa la relación de orden entre clases. El orden de una etiqueta con respecto al total de las etiquetas de un problema vendrá representado por el operador $\mathcal{O}(.)$, para el cual $\mathcal{O}_q = q$, $q = 1, 2, 3..., Q$. Este operador nos informa acerca de la posición de la categoría en la escala ordinal.

### 4.3.2. Taxonomía de los Métodos de Regresión Ordinal

Existen múltiples aproximaciones para tratar los problemas de regresión ordinal, que pasan, o por utilizar modelos no diseñados originalmente para este propósito, o por el uso de otros más específicos. En este apartado se hará un breve resumen de las distintas categorías, propuestas por texto científico previamente citado [10].

- **Aproximaciones Ingenuas**: Estos métodos deciden simplificar el problema ordinal original, para utilizar algoritmos sofisticados dentro del ámbito del aprendizaje de máquinas. De esta manera, convirtiendo cada clase en un número real (manteniendo un orden ascendente entre clases similar al original), podremos hacer uso de métodos de regresión; si obviamos el orden de las etiquetas, podremos usar algoritmos de clasificación nominal, a los que podemos asociar matrices de costes diferentes para los errores de clasificación, como las que se pueden observar en la tabla 4.1

| Sin costes | Costes Absolutos | Costes Cuadráticos |
|---|---|---|
| $\begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix}$ | $\begin{pmatrix} 0 & 1 & 2 & 3 & 4 \\ 1 & 0 & 1 & 2 & 3 \\ 2 & 1 & 0 & 1 & 2 \\ 3 & 2 & 1 & 0 & 1 \\ 4 & 3 & 2 & 1 & 0 \end{pmatrix}$ | $\begin{pmatrix} 0 & 1 & 4 & 9 & 16 \\ 1 & 0 & 1 & 4 & 9 \\ 4 & 1 & 0 & 1 & 4 \\ 9 & 4 & 1 & 0 & 1 \\ 16 & 9 & 4 & 1 & 0 \end{pmatrix}$ |

Tabla 4.1: Tipos de Matrices de Costes

- **Métodos de Descomposición Ordinal**: Los métodos encuadrados en este grupo descomponen el problema ordinal original en múltiples subproblemas binarios (únicamente existen dos clases), a los cuales aplicarán un único modelo, o generarán un modelo por subproblema, posteriormente combinando las salidas predichas de vuelta al problema original.

- **Métodos de Umbral**: Con frecuencia, dentro del ámbito de la regresión ordinal es factible pensar que una variable continua no observada puede dar explicación al comportamiento ordinal del problema; a este tipo de variables se las denomina *latentes*. Los posibles valores que pueda tomar la variable latente se encontrarán divididos en rangos diferentes, representando cada uno a una clase del problema. Los métodos basados en la existencia de estas variables son los métodos de umbral.

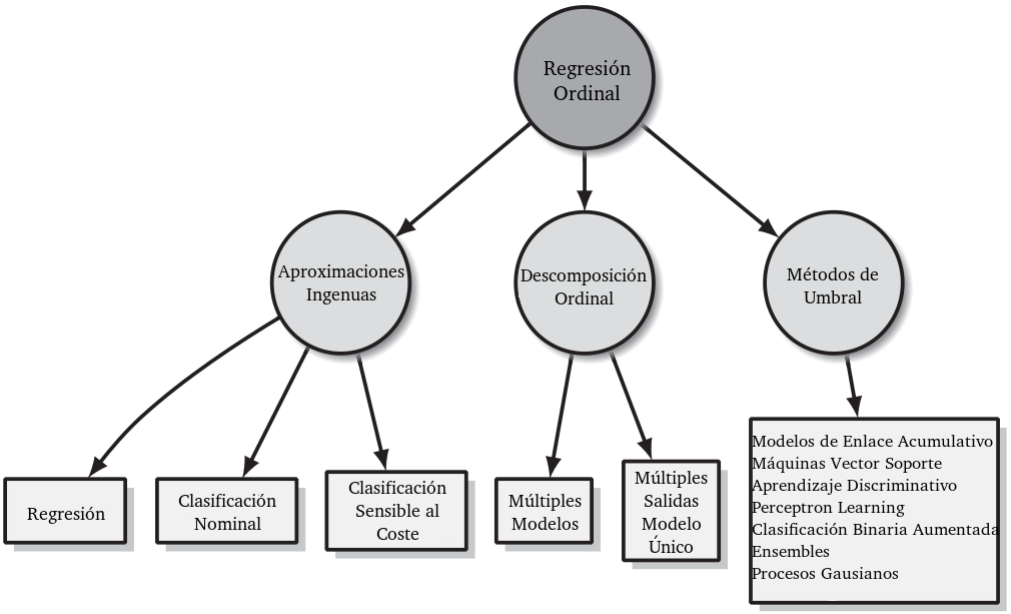


Figura 4.2: Taxonomía de métodos de clasificación ordinal

## 4.4. Métodos de Descomposición Ordinal

En la sección anterior se dió una pequeña definición acerca de los métodos de descomposición ordinal, pero dado que en este Trabajo vamos a codificar un algortimo que implementa este tipo de métodos, creemos necesario hacer un inciso sobre la base teórica en la que se sustenta.

Cuando aplicamos uno de estos métodos, el problema original de $Q$ clases diferentes se dividirá en $Q-1$ subproblemas binarios. En nuestro caso trataremos únicamente con 4 tipos diferentes de decomposición ordinal, ilustrados en la tabla 4.2.

En ella se ilustran los distintos métodos de descomposición ordinal para un problema con 6 clases (filas), que se dividirá en 5 subproblemas diferentes (columnas). Dentro de cada subproblema, se dividirán los ítems del conjunto de datos en función de los signos indicados de la siguiente manera: si el signo es positivo, los patrones de esa clase pasarán a formar parte de la clase positiva; si el signo es negativo, a los de la clase negativa, y si no tiene signo, los patrones de esa clase se ignorarán a la hora de entrenar ese clasificador binario.

| **Descomposición Ordinal** | |
|---|---|
| *OrderedPartitions* | *OneVsNext* |
| $\begin{pmatrix} - & - & - & - & - \\ + & - & - & - & - \\ + & + & - & - & - \\ + & + & + & - & - \\ + & + & + & + & - \\ + & + & + & + & + \end{pmatrix}$ | $\begin{pmatrix} - & & & & \\ + & - & & & \\ & + & - & & \\ & & + & - & \\ & & & + & - \\ & & & & + \end{pmatrix}$ |
| *OneVsFollowers* | *OneVsPrevious* |
| $\begin{pmatrix} - & & & & \\ + & - & & & \\ + & + & - & & \\ + & + & + & - & \\ + & + & + & + & - \\ + & + & + & + & + \end{pmatrix}$ | $\begin{pmatrix} + & + & + & + & + \\ + & + & + & + & - \\ + & + & + & - & \\ + & + & - & & \\ + & - & & & \\ - & & & & \end{pmatrix}$ |

Tabla 4.2: Tipos de Descomposición Ordinal

Una vez realizada la descomposición en problemas binarios y, teniendo en cuenta que el algoritmo a desarrollar se construirá de forma que se genere un modelo diferente por cada subproblema generado, debemos responder a una cuestión: ¿Cómo predecir la etiqueta de un patrón nuevo a partir de $Q-1$

clasificadores binarios?.

En nuestro caso, trataremos con 4 soluciones posibles para esta pregunta, que darán lugar a sendos métodos de decisión. Es necesario comentar que para llevar a cabo esta fase, se hará uso de los códigos generados por la descomposición ordinal como una matriz de códigos ECOC (*Error-Correcting Output Codes*) [16].

1. **Aproximación de Frank y Hall**: En su aproximación [17] utilizan *OrderedPartitions* como método de descomposición, determinando la probabilidad de pertenencia del patrón a cada clase como:

$$p_q = P(y \succ \mathcal{C}_q|x), q = 1, ..., Q-1.$$

Esto se traduce en tres ecuaciones diferentes para el cálculo de probabilidades.

$$P(y = \mathcal{C}_1|x) \approx 1 - p_1, \quad P(y = \mathcal{C}_Q|x) \approx p_{Q-1},$$
$$P(y = \mathcal{C}_q|x) \approx p_{q-1} - p_q, \quad 2 \leq q < Q-1.$$

Salvo que sea necesario mantener las probabilidades de estimación en algún modo, la clase para la cual se predecirá la pertencia del patrón $x$ es aquella que arroje el valor máximo.

2. **Función de Pérdida Exponencial**: Este método de decisión clasifica al patrón en la etiqueta que minimiza la pérdida exponencial,

$$k = \arg \min_{q=1,...,Q} d(M_q, y(x)),$$

donde $M_q$ es la $q$-ésima fila de la matriz de codificación (matriz ECOC), $y(x)$ es un vector que contiene las predicciones de cada clasificador para un patrón $x$, y $d(M_q, y(x))$ es la función de pérdida exponencial:

$$d(M_q, y(x)) = \sum_{i=1}^{Q-1} \exp(M_{qi} \cdot y_i(x)).$$

Los valores del vector $y(x)$ deben encontarse en el rango $[-1, +1]$, mientras que los valores de la matriz de codificación deben ilustrarse con los valores $\{-1, 0, +1\}$.

3. **Función de Pérdida Logarítmica**: Este método, en lugar de minimizar en base a la función de pérdida exponencial, lo hará con respecto a la de pérdida logarítmica, definida como:

$$d(M_q, y(x)) = \sum_{i=1}^{Q-1} \log(1 + \mathrm{e}^{-2 \cdot M_{qi} \cdot f_i(x)}).$$

4. **Función Hinge de Pérdida**: De forma similar a las dos funciones precedentes, el único cambio que se produce es la función de pérdida a utilizar, en este caso conocida como *hinge-loss*, definida como:

$$d(M_q, y(x)) = \sum_{i=1}^{Q-1} max(0, (1 - M_{qi} \cdot f_i(x))).$$

## 4.5. Métricas

Las métricas son una medida del rendimiento de un clasificador, esto es, computan la exactitud con la que los modelos generados predicen la clase correcta para una serie de patrones. Para ello, tienen en cuenta distintos aspectos de los datos obtenidos tras el proceso de clasificación.

Una forma de ilustrar el comportamiento de un clasificador es mediante la matriz de confusión, la cual contrapone las clases reales de los distintos patrones frente a las clases predichas por el algoritmo. En la tabla 4.3 se representa una de matriz de confusión para un problema multiclase.

| Clase Real \ Clase Predicha | 1 | ... | $k$ | ... | $Q$ | |
|---|---|---|---|---|---|---|
| 1 | $n_{11}$ | ... | $n_{1k}$ | ... | $n_{1Q}$ | $n_{1\bullet}$ |
| $\vdots$ | $\vdots$ | | $\vdots$ | | $\vdots$ | $\vdots$ |
| $q$ | $n_{q1}$ | ... | $n_{qk}$ | ... | $n_{qQ}$ | $n_{q\bullet}$ |
| $\vdots$ | $\vdots$ | | $\vdots$ | | $\vdots$ | $\vdots$ |
| $Q$ | $n_{Q1}$ | ... | $n_{Qk}$ | ... | $n_{QQ}$ | $n_{Q\bullet}$ |
| | $n_{\bullet 1}$ | | $n_{\bullet k}$ | | $n_{\bullet Q}$ | $n$ |

Tabla 4.3: Matriz de Confusión

En la matriz de confusión, $n_{qk}$ hace referencia al número de patrones de la clase $q$ predichos como $k$, estando tanto $q$ como $k$ encuadrados dentro de las clases posibles definidas en el problema $\mathcal{Y} = \{\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_Q\}$. Por otra parte, $n_{q\bullet}$ indica el total de patrones pertenecientes a la clase $q$, siendo $n_{\bullet k}$ la suma de patrones predichos como clase $k$.

Las métricas que se presentan a continuación, implementan aquellas definidas dentro del estudio [18], realizado por el grupo AYRNA acerca de la materia que estamos tratando en esta sección.

### 4.5.1. CCR (*Correctly Classified Ratio*)

Esta métrica, también denominada *accuracy*, indica el porcentaje de patrones cuya clase ha sido correctamente predicha:

$$CCR = \frac{1}{n} \sum_{q=1}^{Q} n_{qq}.$$

### 4.5.2. *Minimum Sensitivity*

Devuelve el porcentaje de patrones bien clasificados con respecto a la clase peor clasificada:

$$MS = min\{S_q = \frac{n_{qq}}{n_{q\bullet}};\ q = 1, \ldots, Q\}.$$

### 4.5.3. Media Geométrica

Esta métrica calcula la media geométrica del ratio de verdaderos positivos (o sensibilidad). La sensibilidad devuelve el porcentaje de instancias de una clase que ha sido correctamente clasificado con respecto al total de patrones que pertenecen a dicha clase. De tal forma, que habrá un valor de sensibilidad por clase, el cual representaremos por $S_q$:

$$GM = \sqrt[Q]{\prod_{q=1}^{Q} S_q}.$$

### 4.5.4. MAE (*Mean Absolute Error*)

El error absoluto medio calcula la media de las diferencias relativas entre las posiciones de la clase predicha y la real:

$$MAE = \frac{1}{n}\sum_{q,k=1}^{Q} |\mathcal{O}(y_i) - \mathcal{O}(y_i^*)| n_{qk} \ = \ \frac{1}{n}\sum_{i=1}^{n} e(\mathbf{x_i}),$$

donde $e(\mathbf{x_i}) = |\mathcal{O}(y_i) - \mathcal{O}(y_i^*)|$, siendo $y_i$ la clase real de un patrón $i$ y $y_i^*$ la etiqueta predicha para el mismo.

### 4.5.5. *Average MAE*

Definido como la media de los *MAE* calculados de forma independiente por clase, siendo el *MAE* para la *j-ésima* clase:

$$MAE_j = \frac{1}{n_{q\bullet}} \sum_{k=1}^{Q} |\mathcal{O}(y_i) - \mathcal{O}(y_i^*)| n_{qk},$$

siendo el valor del AMAE:

$$AMAE = \frac{1}{Q} \sum_{q=1}^{Q} MAE_q.$$

### 4.5.6. *Maximum MAE*

Esta métrica es el valor del MAE para la clase con mayor distancia media entre la etiqueta real y la predicha:

$$MMAE = max\{MAE_q;\ \ q = 1, \ldots, Q\}.$$

### 4.5.7. MZE (*Mean Zero-One Error*)

Mejor conocida como ratio de error, es la medida complementaria del *CCR*.

$$MZE = 1 - CCR.$$

### 4.5.8. Coeficiente de Correlación de *Spearman*

Es una medida no paramétrica de la correlación entre dos variables:

$$r_s = \frac{\sum_{i=1}^{n}(\mathcal{O}(y_i) - \overline{\mathcal{O}(y)})\ (\mathcal{O}(y_i)^* - \overline{\mathcal{O}(y^*)})}{\sqrt{\sum_{i=1}^{n}(\mathcal{O}(y_i) - \overline{\mathcal{O}(y)})^2}\ \sqrt{\sum_{i=1}^{n}(\mathcal{O}(y_i^*) - \overline{\mathcal{O}(y^*)})^2}},$$

donde $\overline{\mathcal{O}(y)}$ y $\overline{\mathcal{O}(y^*)}$ son las medias de $\mathcal{O}(y)$ y $\mathcal{O}(y^*)$ respectivamente, representando $i = 1, \ldots, n$ a cada uno de los patrones. Los valores de la métrica se encuentran comprendidos en el rango [-1, 1].

### 4.5.9. Coeficiente de *Kendall*

El coeficiente de Kendall, o $\tau$ de Kendall, es un estadístico que mide la correlación por rangos entre dos ordenaciones de una distribución normal:

$$\tau_b = \frac{\sum_{i,j=1}^{n} c_{ij}^* c_{ij}}{\sqrt{(\sum_{i,j=1}^{n} c_{ij}^{*2})(\sum_{i,j=1}^{n} c_{ij}^2)}},$$

donde $c_{ij}^*$ es $+1$ si $\mathcal{O}(y_i^*) > \mathcal{O}(y_j^*)$, 0 si $\mathcal{O}(y_i^*) = \mathcal{O}(y_j^*)$, y $-1$ si $\mathcal{O}(y_i^*) < \mathcal{O}(y_j^*)$ para todos los patrones $(i, j = 1, \ldots, n)$. $c_{ij}$ se comporta de forma similar. La $\tau_b$ varía en el rango [-1, 1].

### 4.5.10. Coeficiente *Kappa* con pesos

Esta métrica es una versión modificada del coeficiente Kappa, que permite asignar distintos pesos a los diferentes niveles de agregación entre dos variables:

$$WKappa = \frac{P_{o(w)} - P_{e(w)}}{1 - P_{e(w)}},$$

siendo

$$P_{o(w)} = \frac{1}{n}\sum_{q=1}^{Q}\sum_{k=1}^{Q} w_{qk} n_{qk},$$

y

$$P_{e(w)} = \frac{1}{n} \sum_{q=1}^{Q} \sum_{k=1}^{Q} w_{qk} n_{q\bullet} n_{\bullet k},$$

donde el peso $w_{qk} = |\mathcal{O}(y_i) - \mathcal{O}(y_i^*)|$ cuantifica el grado de disparidad entre la etiqueta real, $q$ y la etiqueta predicha $k$, variando el valor del estadístico dentro del rango [-1,1].

# Capítulo 5

# Restricciones

A continuación se expondrán todas aquellas restricciones existentes en el ámbito del diseño, que condicionan la elección entre las distintas alternativas presentes en fases posteriores del Trabajo. Podemos observar la existencia de dos tipos de factores diferenciados entre sí:

- ***Factores dato:*** Son aquellas restricciones que no pueden ser modificadas, inherentes al entorno en el que se desarrolla el Trabajo. Por lo general son explicitadas por el cliente, en nuestro caso, el grupo de investigación AYRNA [3].

- ***Factores estratégicos:*** Representan las opciones elegidas por el desarrollador, esto es, cuando no existe una imposición por parte de un agente externo. Un ejemplo sería el uso del sistema de composición de textos LaTeXen la elaboración de la documentación, debido a la facilidad con la que trata con los aspectos de presentación del documento, incluida la formulación matemática.

## 5.1. Factores Dato

Los factores dato comprenden un grupo de restricciones que han de cumplirse a lo largo del desarrollo del Trabajo, sin tener que estar necesariamente justificadas.

- ***Restricciones Humanas:*** Debido a que nos encontramos desarrollando un Trabajo de Fin de Grado, este factor viene determiando por los directores del mismo (pertenecientes al grupo de investigación AYRNA). En nuestro caso, el Trabajo será realizado por un único alumno.

- ***Restricciones Económicas:*** Al tratarse de un Trabajo de Fin de Grado y no de un proyecto de desarrollo de una aplicación profesional, el desarrollador debe apoyarse lo máximo posible en herramientas de *software* libre. Si no hubiese más remedio, se utilizarían programas cuyo uso fuera del ámbito profesional estuviese exento de pago.

- ***Restricciones Temporales:*** Condicionadas por el cumplimiento de todos los objetivos que deben satisfacerse para que la aplicación desarrollada se considere como válida. No obstante, para no alargar en el tiempo de forma indefenida la realización de este Trabajo, se espera que este se encuentre terminado para la convocatoria de junio de 2020, siempre y cuando no ocurran imprevistos.

- ***Restricciones de Hardware:*** El alumno encargado de desarrollar el Trabajo hará uso de sus propios equipos informáticos para su elaboración.

- ***Restricciones de Software:*** El *framework* a desarrollar ha de estar implementado en el lenguaje de programación Python, además de encontrarse integrado con las diferentes bibliotecas que se han ido mencionando en este documento.

## 5.2. Factores Estratégicos

Los factores estratégicos son restricciones impuestas por el propio proyectista y sus directores de proyecto sobre el Trabajo.

- Se utilizará el lenguaje de programación de alto nivel Python. Esto es debido a una serie de ventajas que proporciona el uso del mismo:
  - Compatibilidad con importantes bibliotecas del área de la ciencia computacional, en concreto con la del aprendizaje de máquinas.
  - Al ser un lenguaje interpretado, su portabilidad entre sistemas informáticos es trivial, incluso si estos utilizan sistemas operativos diferentes.
  - Posee una documentación extensa para ayudar al programador.
- Los ficheros de configuración se especificarán en formato JSON, siendo este un formato de fácil comprensión, que hace uso de una cantidad mínima de reglas y tiene una correspondencia directa con un tipo de datos dentro del lenguaje Python, como son los diccionarios.
- Se integrará el programa con la biblioteca `scikit-learn` por dos motivos principalmente: primero, cuenta con una vasta cantidad de algoritmos de clasificación nominal desarrollados en su seno, lo que permitirá al *software* servir de *framework* para la experimentación con este tipo de clasificadores y/o utilizarlos como clasificadores internos dentro del algoritmo desarrollado y, segundo, dispone de una serie de objetos que facilita la implementación de la fase de validación cruzada. Además de estos dos puntos, también proporciona una gran cantidad de clases con las que se puede expandir la utilidad del sistema.
- Se utilizará el sistema de composición de textos LaTeXpara la generación de los documentos de los que se componga el presente Trabajo, debido a la facilidad que provee este para la maquetación de textos,

pudiendo incluir sin excesivas complicaciones toda la bibliografía utilizada, formulas matemáticas, índices, etc.

# Capítulo 6

# Recursos

Se procederá a continuación a detallar los recursos de los que se ha dispuesto para la realización del Trabajo. Estos recursos se dividirán en los siguientes apartados:

## 6.1. Recursos Humanos

- **Iván Bonaque Muñoz:** Alumno de Grado de Ingeniería Informática en la Escuela Politécnica Superior de la Universidad de Córdoba. Será el encargado del diseño, implementación y documentación del Trabajo de Fin de Grado.
- **Pedro Antonio Gutiérrez Peña:** Profesor titular de Universidad del Departamento de Informática y Analisis Numérico en la Escuela Politécnica Superior de la Universidad de Córdoba, investigador del Grupo de Investigación AYRNA y encargado de dar soporte técnico y teórico para las tareas de análisis, diseño y programación.
- **Javier Sánchez Monedero:** Investigador asociado a la Universidad de Cardiff, investigador del Grupo de Investigación AYRNA y encargado de dar soporte técnico y teórico para las tareas de análisis, diseño

y programación.

## 6.2. Recursos Hardware

Para la realización del Trabajo se hará uso de un ordenador portátil propiedad del proyectista, el cual cuenta con las siguientes características técnicas:

- Procesador Intel Core i7-Q740, Quad-Core a 1.73GHz.
- Memoria Principal de 4GB DDR3 a 1333MHz.
- Tarjeta Gráfica Nvidia GeForce GT 425M con 1GB de VRAM.
- Disco duro rígido de 450GB de capacidad.

## 6.3. Recursos Software

### 6.3.1. Sistemas Operativos

- GNU/Linux Debian 9.5 "Stretch"(64 bits)
- Windows 10 (64 bits)

### 6.3.2. Lenguajes de Programación

- *Python*, v2.7.13 y v3.5.3

### 6.3.3. Herramientas de Documentación

- *Dia v0.97*: Herramienta de creación de diagramas.

- *Gimp v2.8.18*: Herramienta de edición de imágenes.
- *PDFTeX v3.14159265-2.6-1.40.17*: Intérprete de LaTeX.

# Parte II

# Requisitos del Sistema

# Capítulo 7

# Especificación de Requisitos

En esta parte de la documentación se ofrecerá una visión más detallada de la aplicación que se va a desarrollar.

## 7.1. Introducción

Como se ha comentado en anteriores capítulos, la finalidad principal de este Trabajo es la de crear un *framework* para facilitar la experimentación con algoritmos de regresión ordinal, así como la implementación de un algoritmo basado en la descomposición ordinal.

La aplicación será desarrollada a petición del Grupo de Investigación AYRNA de la Universidad de Córdoba, que tienen en la regresión ordinal un importante campo de investigación. Sin embargo, todo el código de la aplicación, incluidos los comentarios, se encontrarán escritos en inglés para adecuarse a las bibliotecas que se utilizarán en su desarrollo.

## 7.2. Participantes del Trabajo

Los participantes del presente Trabajo de Fin de Grado son los indicados en las tablas 7.1, 7.2 y 7.3, que se muestran a continuación:

| **Participante** | Iván Bonaque Muñoz |
|---|---|
| **Organización** | EPS UCO |
| **Rol** | Desarrollador |
| **Desarrollador** | Si |
| **Cliente** | No |
| **Usuario** | No |
| **Comentarios** | Alumno de Grado en Ingeniería Informática de la Escuela Politécnica Superior de la Universidad de Córdoba, proyectista encargado del desarrollo de la aplicación. |

Tabla 7.1: Participante Iván Bonaque Muñoz

| **Participante** | Pedro Antonio Gutiérrez Peña |
|---|---|
| **Organización** | AYRNA |
| **Rol** | Tutor |
| **Desarrollador** | No |
| **Cliente** | Si |
| **Usuario** | Si |
| **Comentarios** | Profesor titular de Universidad del Departamento de Informática y Análisis Numérico de la Escuela Politécnica Superior de la Universidad de Córdoba, investigador del grupo AYRNA y encargado de dar soporte técnico y teórico para las tareas de análisis, diseño y programación. |

Tabla 7.2: Participante Pedro Antonio Gutiérrez Peña

| **Participante** | Javier Sánchez Monedero |
|---|---|
| **Organización** | AYRNA |
| **Rol** | Tutor |
| **Desarrollador** | No |
| **Cliente** | Si |
| **Usuario** | Si |
| **Comentarios** | Investigador asociado a la Universidad de Cardiff, investigador del Grupo AYRNA y encargado de dar soporte técnico y teórico para las tareas de análisis, diseño y programación. |

Tabla 7.3: Participante Javier Sánchez Monedero

## 7.3. Catálogo de Objetivos del Sistema

Los objetivos fundamentales de la aplicación que debemos desarrollar se expondrán en las tablas 7.4, 7.5, 7.6, 7.7, 7.8, 7.9, 7.10 y 7.11

| **OBJ-001** | Desarrollo de un *framework* |
|---|---|
| **Versión** | 1.0 |
| **Autores** | Iván Bonaque Muñoz |
| **Fuentes** | Pedro Antonio Gutiérrez Peña<br>Javier Sánchez Monedero |
| **Descripción** | Desarrollo de una aplicación para facilitar la experimentación con conjuntos de datos y algoritmos de clasificación ordinal. |
| **Subobjetivos** | OBJ-003 Opciones de configuración<br>OBJ-004 Datos de entrada<br>OBJ-005 Almacenamiento de resultados<br>OBJ-006 Adaptación de métricas |
| **Comentarios** | Ninguno. |

Tabla 7.4: Objetivo relativo al desarrollo de la aplicación de experimentación

| **OBJ-002** | Implementación algoritmo de regresión ordinal |
|---|---|
| **Versión** | 1.0 |
| **Autores** | Iván Bonaque Muñoz |
| **Fuentes** | Pedro Antonio Gutiérrez Peña<br>Javier Sánchez Monedero |
| **Descripción** | Desarrollo de un algoritmo de regresión ordinal basado en el método de descomposición ordinal. |
| **Subobjetivos** | OBJ-007 Implementación métodos descomposición<br>OBJ-008 implementación funciones de decisión |
| **Comentarios** | Como se comentó en la sección 4.4, el método a desarrollar será un *ensemble* (conjunto) constituido por varios clasificadores binarios, que se combinarán para predecir las etiquetas de un problema multiclase de clasificación ordinal. |

Tabla 7.5: Objetivo relativo a la implementación del método de descomposición ordinal

| **OBJ-003** | Opciones de configuración |
|---|---|
| **Versión** | 1.0 |
| **Autores** | Iván Bonaque Muñoz |
| **Fuentes** | Pedro Antonio Gutiérrez Peña<br>Javier Sánchez Monedero |
| **Descripción** | Los conjuntos de datos, métricas de evaluación y algoritmos a utilizar serán especificados mediante archivos de configuración. |
| **Subobjetivos** | Ninguno. |
| **Comentarios** | En el caso de los algoritmos, se podrán indicar una serie de parámetros con distintos valores, de entre los cuales escoger para construir el modelo que mejor se adapte al conjunto de datos durante la fase de validación cruzada. El formato de los ficheros de configuración se puede apreciar en el apéndice A.6. |

Tabla 7.6: Objetivo relativo a las opciones de configuración

| **OBJ-004** | Datos de entrada |
|---|---|
| **Versión** | 1.0 |
| **Autores** | Iván Bonaque Muñoz |
| **Fuentes** | Pedro Antonio Gutiérrez Peña<br>Javier Sánchez Monedero |
| **Descripción** | Para el correcto funcionamiento del *framework*, las BBDD deben encontrarse en formatos específicos. |
| **Subobjetivos** | Ninguno. |
| **Comentarios** | El formatos en el que deben encontrarse las entradas se exponen en el apéndice A.7. |

Tabla 7.7: Objetivo relativo a la entrada de datos

| **OBJ-005** | Almacenamiento de resultados |
|---|---|
| **Versión** | 1.0 |
| **Autores** | Iván Bonaque Muñoz |
| **Fuentes** | Pedro Antonio Gutiérrez Peña<br>Javier Sánchez Monedero |
| **Descripción** | Los resultados obtenidos tras la ejecución de un experimento se guardarán en un formato concreto de fácil interpretación. |
| **Subobjetivos** | Ninguno. |
| **Comentarios** | Los resultados se almacenarán en una subcarpeta dentro de la carpeta especificada en el fichero de configuración. La estructura y formato de los resultados se exponen en el apéndice A.8. |

Tabla 7.8: Objetivo relativo a la salida de datos

| **OBJ-006** | Adaptación de métricas |
|---|---|
| **Versión** | 1.0 |
| **Autores** | Iván Bonaque Muñoz |
| **Fuentes** | Pedro Antonio Gutiérrez Peña<br>Javier Sánchez Monedero |
| **Descripción** | Adaptación de las métricas de evaluación del rendimiento de clasificadores para hacerlas compatibles con el *framework*. |
| **Subobjetivos** | Ninguno. |
| **Comentarios** | Ninguno. |

Tabla 7.9: Objetivo relativo a las métricas de rendimiento

| **OBJ-007** | Implementación de métodos de descomposición |
|---|---|
| **Versión** | 1.0 |
| **Autores** | Iván Bonaque Muñoz |
| **Fuentes** | Pedro Antonio Gutiérrez Peña<br>Javier Sánchez Monedero |
| **Descripción** | Estos métodos dividirán la BBDD utilizada para entrenar a los clasificadores base en función de la matriz de codificación seleccionada, que variará en función del método de descomposición. |
| **Subobjetivos** | Ninguno. |
| **Comentarios** | A pesar de que el tipo de clasificador e hiperparámetros utilizados para todos los subproblemas son similares, cada uno genera un modelo diferente, ya que los patrones que usan para entrenar, divididos en clase positiva y negativa, son diferentes. |

Tabla 7.10: Objetivo relativo a la implementación de los distintos tipos de descomposición ordinal

| **OBJ-008** | Implementación de las funciones de decisión |
|---|---|
| **Versión** | 1.0 |
| **Autores** | Iván Bonaque Muñoz |
| **Fuentes** | Pedro Antonio Gutiérrez Peña<br>Javier Sánchez Monedero |
| **Descripción** | Codificación de cada una de las funciones para decidir a que clase pertenece un patrón durante la fase de generalización. |
| **Subobjetivos** | Ninguno. |
| **Comentarios** | Se implementarán 4 funciones diferentes, anteriormente explicadas en la sección 4.4. |

Tabla 7.11: Objetivo relativo a la codificación de las distintas funciones de decisión

## 7.4. Catálogo de Requisitos del Sistema

### 7.4.1. Requisitos de Información

La aplicación que se va a desarrollar va a ser utilizada para experimentar con conjuntos de datos y distintos modelos matemáticos, no poseyendo las características de un sistema de información. Esto implica que los requisitos de información serán escasos, concretamente tres, definidos en las tablas 7.12, 7.13 y 7.14

| **IRQ-001** | Ficheros de configuración |
|---|---|
| **Versión** | 1.0 |
| **Autores** | Iván Bonaque Muñoz |
| **Fuentes** | Pedro Antonio Gutiérrez Peña<br>Javier Sánchez Monedero |
| **Dependencias** | OBJ-003 Opciones de configuración |
| **Descripción** | Los distintos parámetros de configuración del experimento se deben poder explicitar mediante ficheros de formato JSON, que pueden ser reutilizados. |
| **Comentarios** | Ninguno. |

Tabla 7.12: Requisito de información relativo a los ficheros de configuración

| **IRQ-002** | Entrada de datos |
|---|---|
| **Versión** | 1.0 |
| **Autores** | Iván Bonaque Muñoz |
| **Fuentes** | Pedro Antonio Gutiérrez Peña<br>Javier Sánchez Monedero |
| **Dependencias** | OBJ-004 Datos de entrada |
| **Descripción** | El *framework* debe leer los distintos conjuntos de datos desde ficheros CSV, siempre que cumplan una cierta estructura y restricciones. |
| **Comentarios** | El formato concreto que han de tener los ficheros de entrada se expone en el apéndice A.7. |

Tabla 7.13: Requisito de información relativo a la entrada de datos

| **IRQ-003** | Almacenamiento de resultados |
|---|---|
| **Versión** | 1.0 |
| **Autores** | Iván Bonaque Muñoz |
| **Fuentes** | Pedro Antonio Gutiérrez Peña<br>Javier Sánchez Monedero |
| **Dependencias** | OBJ-005 Almacenamiento de resultados |
| **Descripción** | La aplicación debe ser capaz de almacenar los resultados de un experimento en ficheros de texto. |
| **Comentarios** | La estructura y el formato de los datos a almacenar se explicará en el apéndice A.8. |

Tabla 7.14: Requisito de información relativo al almacenamiento de resultados

### 7.4.2. Requisitos Funcionales

Los distintos componentes que definen el comportamiento del *software* a desarrollar se definen en las tablas 7.15, 7.16, 7.17, 7.18, 7.19 y 7.20.

| **FRQ-001** | Carga de bases de datos |
|---|---|
| **Versión** | 1.0 |
| **Autores** | Iván Bonaque Muñoz |
| **Fuentes** | Pedro Antonio Gutiérrez Peña<br>Javier Sánchez Monedero |
| **Objetivos Asociados** | OBJ-004 Datos de entrada |
| **Requisitos Asociados** | Ninguno. |
| **Descripción** | El programa leerá las BBDD desde ficheros en formatos conocidos. Estas pueden, o no, estar divididas en múltiples particiones. |
| **Comentarios** | Los ficheros a leer deben seguir unas convenciones de nomenclatura y estructura en cuanto a los directorios que los contienen. |

Tabla 7.15: Requisito funcional relativo a la lectura de conjuntos de datos

| **FRQ-002** | Configuración de los experimentos |
|---|---|
| **Versión** | 1.0 |
| **Autores** | Iván Bonaque Muñoz |
| **Fuentes** | Pedro Antonio Gutiérrez Peña<br>Javier Sánchez Monedero |
| **Objetivos Asociados** | OBJ-003 Opciones de configuración |
| **Requisitos Asociados** | FRQ-003 Ejecución de los algoritmos de clasificación<br>FRQ-004 Realizar validación cruzada |
| **Descripción** | La aplicación necesitará de un fichero de configuración para poder funcionar. En él se introducirán todos los datos que definan al experimento. Dentro de cada archivo de configuración se pueden distinguir dos partes bien diferenciadas:<br>▪ **Configuración General**: Aquí se indican los conjuntos de datos con los que trabajar y donde encontrarlos, las métricas de rendimiento a utilizar, la métrica con la que guiar la fase de validación cruzada, el número de *folds* a crear durante la validación cruzada, el número de hilos en los que dividirá la ejecución del programa y la carpeta donde almacenar los resultados.<br>▪ **Configuración de los Algoritmos**: Se debe especificar una configuración por cada clasificador que se quiera utilizar, donde se especifica la ruta hasta el algoritmo de clasificación y los diferentes hiperparámetros utilizados para optimizar el modelo. |
| **Comentarios** | Los hiperparámetros se tendrán en cuenta durante la fase de validación cruzada. |

Tabla 7.16: Requisito funcional relativo a la configuración de los experimentos

| **FRQ-003** | Ejecución de los algoritmos de clasificación |
|---|---|
| **Versión** | 1.0 |
| **Autores** | Iván Bonaque Muñoz |
| **Fuentes** | Pedro Antonio Gutiérrez Peña<br>Javier Sánchez Monedero |
| **Objetivos Asociados** | OBJ-002 Implementación algoritmo de regresión ordinal<br>OBJ-003 Opciones de configuración<br>OBJ-005 Almacenamiento de resultados |
| **Requisitos Asociados** | Ninguno. |
| **Descripción** | Si la configuración proporcionada por el usuario es correcta, el programa realizará el experimento, consistente en la generación, por cada algoritmo de clasificación, de un modelo para cada conjunto de datos. Durante la ejecución se mostrará al usuario el progreso relativo. |
| **Comentarios** | En caso de estar fragmentada alguna BBDD en distintas particiones, se entrenará un modelo independiente para cada una de estas. |

Tabla 7.17: Requisito funcional relativo a la ejecución de los clasificadores

| **FRQ-004** | Realizar validación cruzada |
|---|---|
| **Versión** | 1.0 |
| **Autores** | Iván Bonaque Muñoz |
| **Fuentes** | Pedro Antonio Gutiérrez Peña<br>Javier Sánchez Monedero |
| **Objetivos Asociados** | OBJ-003 Opciones de configuración |
| **Requisitos Asociados** | FRQ-003 Ejecución de los algoritmos de clasificación |
| **Descripción** | El programa llevará a cabo una validación cruzada con el número de *folds* que se indique en el fichero de configuración del experimento. Al finalizar esta fase, se escogerá el modelo que mejores valores de la métrica de rendimiento haya obtenido para el conjunto de datos usado para validar. |
| **Comentarios** | Se realizarán tantos *k-folds* distintos como combinaciones de valores de hiperparámetros diferentes se puedan realizar. El modelo definitivo contará únicamente con un valor por parámetro especificado. |

Tabla 7.18: Requisito funcional relativo a la validación de hiperparámetros

| **FRQ-005** | Almacenamiento de resultados |
|---|---|
| **Versión** | 1.0 |
| **Autores** | Iván Bonaque Muñoz |
| **Fuentes** | Pedro Antonio Gutiérrez Peña<br>Javier Sánchez Monedero |
| **Objetivos Asociados** | OBJ-005 Almacenamiento de resultados |
| **Requisitos Asociados** | Ninguno. |
| **Descripción** | Tras finalizar correctamente la ejecución de un experimento, los resultados relativos al mismo se guardarán en ficheros de texto. Concretamente, se almacenará, por cada combinación de conjunto de datos y algoritmo, un fichero donde se indiquen los valores obtenidos por el mejor modelo para las distintas métricas, un fichero que almacene el objeto del mejor modelo obtenido y un fichero con las predicciones obtenidas por el modelo para los conjuntos de entrenamiento y generalización. |
| **Comentarios** | En caso de estar dividido un conjunto de datos en diversas particiones, se generará un modelo por partición, con lo que ello conlleva con respecto a los resultados generados. |

Tabla 7.19: Requisito funcional relativo al almacenamiento de resultados

| **FRQ-006** | Descomposición ordinal |
|---|---|
| **Versión** | 1.0 |
| **Autores** | Iván Bonaque Muñoz |
| **Fuentes** | Pedro Antonio Gutiérrez Peña<br>Javier Sánchez Monedero |
| **Objetivos Asociados** | OBJ-007 Implementación de métodos de descomposición<br>OBJ-008 Implementación de las funciones de decisión |
| **Requisitos Asociados** | FRQ-003 Ejecución de los algoritmos de clasificación |
| **Descripción** | El sistema dispondrá de una clase local donde se encuentre definido el algoritmo de regresión ordinal desarrollado. Este clasificador implementa la teoría expuesta en la sección 4.4. Cuenta con distintas variables con las que modificar su comportamiento: |

| | |
|---|---|
| | •**Tipo de Descomposición**: Cada uno de sus cuatro valores posibles generará una matriz de códigos diferente, atribuyendo a cada clase binaria distintas instancias según el caso.<br>•**Método de Decisión**: Al igual que la variable anterior, dispone de cuatro posibles valores, previamente analizados en la sección 4.4. Cada uno de ellos decidirá a que clase del problema ordinal pertenece cada patrón de forma distinta.<br>•**Clasificador Base**: Clasificador interno del que hará uso el algoritmo para generar los distintos modelos asociados a cada uno de los problemas binarios. Puede ser un clasificador existente dentro de la biblioteca `scikit-learn`, o un algoritmo desarrollado en el seno del *framework*.<br>•**Parámetros del Clasificador**: Como cualquier algoritmo de clasificación, el clasificador base dispone de una serie de parámetros con los que modificar su comportamiento. Los parámetros y posibles valores dependerán del algoritmo seleccionado. |
| **Comentarios** | Ninguno. |

Tabla 7.20: Requisito funcional relativo al algoritmo de descomposición ordinal

### 7.4.3. Requisitos No Funcionales

Los requisitos no funcionales que son exigidos para el sistema a desarrollar se detallan en las tablas 7.21, 7.22, 7.23, 7.24, 7.25, 7.26 y 7.27.

| **NFR-001** | Entorno de programación Python |
|---|---|
| **Versión** | 1.0 |
| **Autores** | Iván Bonaque Muñoz |
| **Fuentes** | Pedro Antonio Gutiérrez Peña<br>Javier Sánchez Monedero |
| **Dependencias** | OBJ-001 Desarrollo de un *framework*<br>OBJ-002 Implementación algoritmo de regresión ordinal |
| **Descripción** | Para el desarrollo del Trabajo se utilizará Python como lenguaje de programación. |
| **Comentarios** | Ninguno. |

Tabla 7.21: Requisito no funcional relativo al entorno de programación utilizado

| **NFR-002** | Usabilidad |
|---|---|
| **Versión** | 1.0 |
| **Autores** | Iván Bonaque Muñoz |
| **Fuentes** | Pedro Antonio Gutiérrez Peña<br>Javier Sánchez Monedero |
| **Dependencias** | OBJ-003 Opciones de configuración<br>OBJ-004 Datos de entrada<br>OBJ-005 Almacenamiento de resultados |
| **Descripción** | La herramienta software debe ser relativamente fácil e intuitiva para el usuario final. De forma que el investigador no necesite de demasiado tiempo para aprender a utilizarla. |
| **Comentarios** | La usabilidad se tendrá en cuenta a la hora de seleccionar el tipo de fichero de configuración a utilizar y el modo en que se estructurarán los distintos campos de configuración. |

Tabla 7.22: Requisito no funcional relativo a la usabilidad del sistema

| **NFR-003** | Formatos de texto soportados |
|---|---|
| **Versión** | 1.0 |
| **Autores** | Iván Bonaque Muñoz |
| **Fuentes** | Pedro Antonio Gutiérrez Peña<br>Javier Sánchez Monedero |
| **Dependencias** | OBJ-004 Datos de entrada |
| **Descripción** | La aplicación leerá las distintas bases de datos, siempre y cuando estas se encuentren en formato CSV. |
| **Comentarios** | Ninguno. |

Tabla 7.23: Requisito no funcional relativo a la entrada de datos

| **NFR-004** | Formato de ficheros de configuración |
|---|---|
| **Versión** | 1.0 |
| **Autores** | Iván Bonaque Muñoz |
| **Fuentes** | Pedro Antonio Gutiérrez Peña<br>Javier Sánchez Monedero |
| **Dependencias** | OBJ-003 Opciones de configuración |
| **Descripción** | Los ficheros de configuración han de estar especificados en formato JSON. |
| **Comentarios** | Ninguno. |

Tabla 7.24: Requisito no funcional relativo al formato de los archivos de configuración

| **NFR-005** | Fiabilidad y optimización |
|---|---|
| **Versión** | 1.0 |
| **Autores** | Iván Bonaque Muñoz |
| **Fuentes** | Pedro Antonio Gutiérrez Peña<br>Javier Sánchez Monedero |
| **Dependencias** | OBJ-001 Desarrollo de un *framework*<br>OBJ-002 Implementación algoritmo de regresión ordinal |
| **Descripción** | Tanto el *framework* como el algoritmo de clasificación ordinal desarrollados deben tener tolerancia a fallos, de modo que detecten entradas inesperadas por parte del usuario, y le notifiquen del error, lo más concisamente posible, a través de la terminal. También deben evitar añadir un gasto de tiempo innecesario al usuario, estando todos los métodos lo más optimizados que sea posible. |
| **Comentarios** | Ninguno. |

Tabla 7.25: Requisito no funcional relativo al rendimiento y la fiabilidad

| **NFR-006** | Desarrollo del sistema en inglés |
|---|---|
| **Versión** | 1.0 |
| **Autores** | Iván Bonaque Muñoz |
| **Fuentes** | Pedro Antonio Gutiérrez Peña<br>Javier Sánchez Monedero |
| **Dependencias** | OBJ-001 Desarrollo de un *framework*<br>OBJ-002 Implementación algoritmo de regresión ordinal<br>OBJ-003 Opciones de configuración<br>OBJ-005 Almacenamiento de resultados |
| **Descripción** | El sistema deberá estar completamente desarrollado en inglés; desde el nombre de las clases y todas las variables, hasta los comentarios más extensos del mismo. |
| **Comentarios** | Los resultados también se almacenarán y presentarán utilizando este idioma. |

Tabla 7.26: Requisito no funcional relativo al idioma utilizado.

| **NFR-007** | Formatos de almacenamiento de los resultados |
|---|---|
| **Versión** | 1.0 |
| **Autores** | Iván Bonaque Muñoz |
| **Fuentes** | Pedro Antonio Gutiérrez Peña<br>Javier Sánchez Monedero |
| **Dependencias** | OBJ-005 Almacenamiento de resultados |
| **Descripción** | El programa almacenará los resultados en ficheros de texto. La estructura de carpetas y el formato en el que se almacenarán será explicado en el apéndice A.8. |
| **Comentarios** | Ninguno. |

Tabla 7.27: Requisito no funcional relativo al formato de los archivos de salida

## 7.5. Matriz de Rastreabilidad

En esta sección mostraremos la matriz de rastreabilidad para los objetivos y requisitos antes desarrollados. Esta matriz, ilustrada en la tabla 7.28 nos será de gran ayuda a la hora de relacionar los objetivos con los distintos tipos de requisitos de una manera visual.

| TRM-0001 | OBJ-0001 | OBJ-0002 | OBJ-0003 | OBJ-0004 | OBJ-0005 | OBJ-0006 | OBJ-0007 | OBJ-0008 |
|---|---|---|---|---|---|---|---|---|
| IRQ-0001 | | | X | | | | | |
| IRQ-0002 | | | | X | | | | |
| IRQ-0003 | | | | | X | | | |
| FRQ-0001 | | | | X | | | | |
| FRQ-0002 | | | X | X | | | | |
| FRQ-0003 | | X | X | | X | | | |
| FRQ-0004 | | | X | | | | | |
| FRQ-0005 | | | | | X | | | |
| FRQ-0006 | | | | | | | X | X |
| NFR-0001 | X | X | | | | | | |
| NFR-0002 | | | X | X | X | | | |
| NFR-0003 | | | | X | | | | |
| NFR-0004 | | | X | | | | | |
| NFR-0005 | X | X | | | | | | |
| NFR-0006 | X | X | X | | X | | | |
| NFR-0007 | | | | | X | | | |

Tabla 7.28: Matriz de rastreabilidad

# Parte III

# Análisis y Diseño del Sistema

# Capítulo 8

# Casos de Uso

Los diagramas de casos de uso nos ayudan a representar gráficamente la interacción existente entre uno o varios actores (principalmente el usuario) y el sistema. De acuerdo con Alistair Cockburn [19], se puede definir un caso de uso mediante la especificación de los siguientes componentes:

- ***Actores o Roles***: Elementos o personas que utilizan el sistema.
- ***Límite del sistema***: Delimita el rango de acción del actor.
- ***Casos de uso***: Acciones que pueden realizar los distintos actores dentro de los límites del sistema.
- ***Relaciones***: Interacciones entre actores y casos de uso.

De forma similar a como se trabaja con los diagramas de flujo, comenzaremos mostrando la interacción usuario-sistema desde el punto de vista más general, para ir adentrándonos posteriormente en la estructura del mismo. A causa del objetivo que persigue nuestro sistema, la interacción del usuario con el mismo se limita a proporcionar ficheros de configuración con los que ejecutar un experimento y a interpretar los resultados que este devuelve durante la ejecución.

## 8.1. Caso de Uso: General

Mediante los siguientes casos de uso mostraremos el funcionamiento del sistema de forma completa. En ellos, hay un único actor, el usuario, el cual utiliza el sistema para realizar un experimento. Como se explicó con anterioridad en el capítulo 7, los experimentos consisten en aplicar una o varias configuraciones de clasificadores diferentes sobre una o varias bases de datos, generando en cada combinación clasificador-BBDD un modelo optimizado mediante validación cruzada.

El diagrama de la figura 8.1 muestra la interacción del usuario con el sistema en su mayor grado de abstracción. Por su parte, el diagrama de la figura 8.2 aumenta el detalle para este caso de uso general. Los detalles de ambos casos de uso se incluyen en la tabla 8.1.

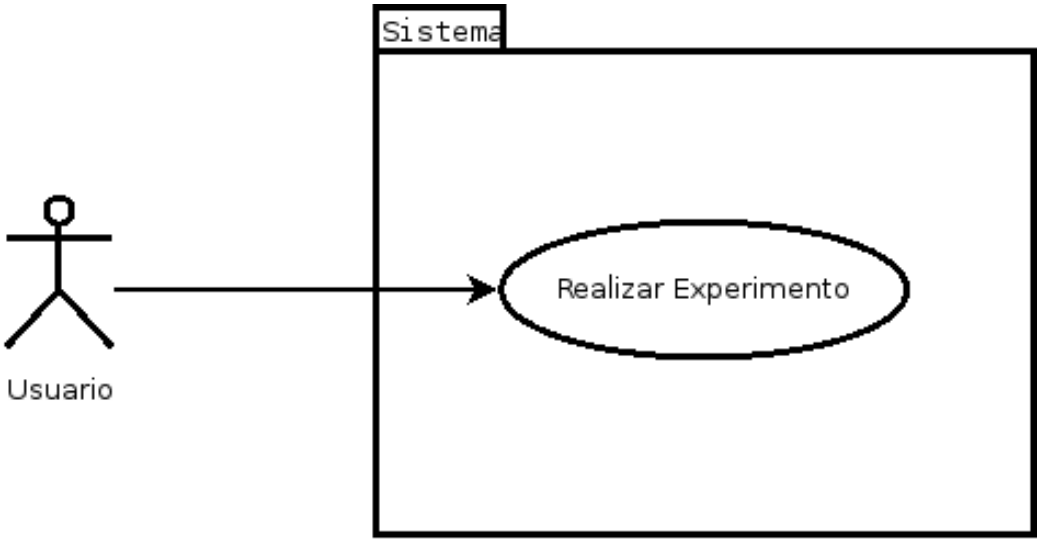


Figura 8.1: Caso de uso abstracción máxima del sistema

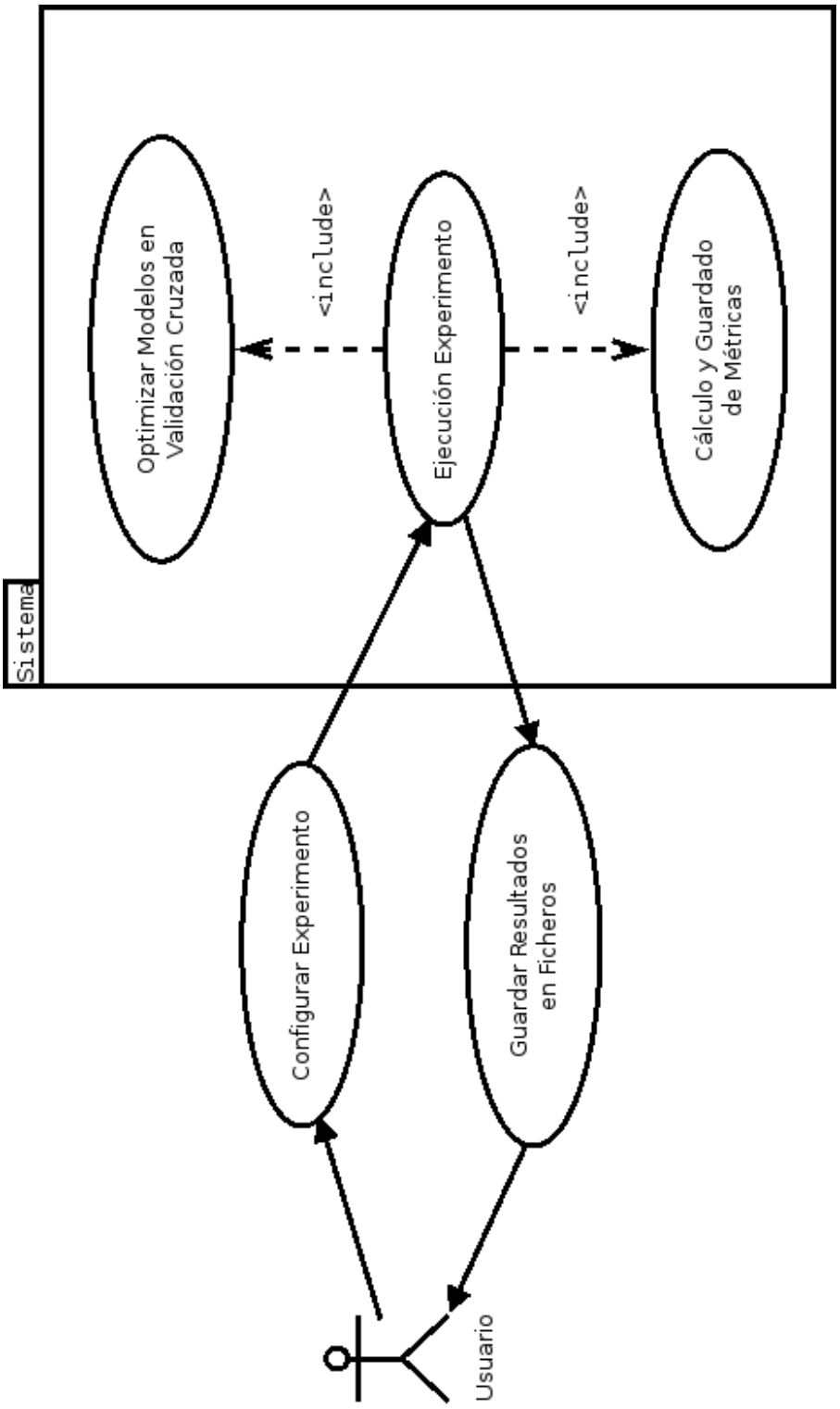


Figura 8.2: Caso de uso funcionamiento general del sistema

| **Caso de Uso** | *Framework* para clasificación ordinal |
|---|---|
| **Descripción** | El usuario realiza un experimento con un número variable de algoritmos de clasificación ordinal y conjuntos de datos |
| **Actores Principales** | Usuario |
| **Actores Secundarios** | Ninguno |
| **Condiciones Iniciales** | ▪ Las bases de datos deben estar correctamente formateadas<br>▪ La configuración proporcionada debe ser correcta |
| **Flujo Principal** | 1. El usuario proporciona la configuración al *framework*<br>2. Inicia la ejecución del programa con esa configuración<br>3. Se almacenan los resultados del experimento |
| **Condiciones Finales** | Se han almacenado correctamente los resultados del experimento |
| **Flujos Alternativos** | El programa finaliza prematuramente a causa de un error |

Tabla 8.1: Caso de uso de la abstracción total del sistema

## 8.2. Caso de Uso: Configuración de un Experimento

Para poder realizar un experimento, el usuario ha de crear un fichero de configuración donde se especifiquen distintos parámetros, necesarios para el correcto funcionamiento del programa. Como se ve en el diagrama de la figura 8.3, esta configuración consta de dos partes, que son explicadas con más detalle en el apéndice A.6.

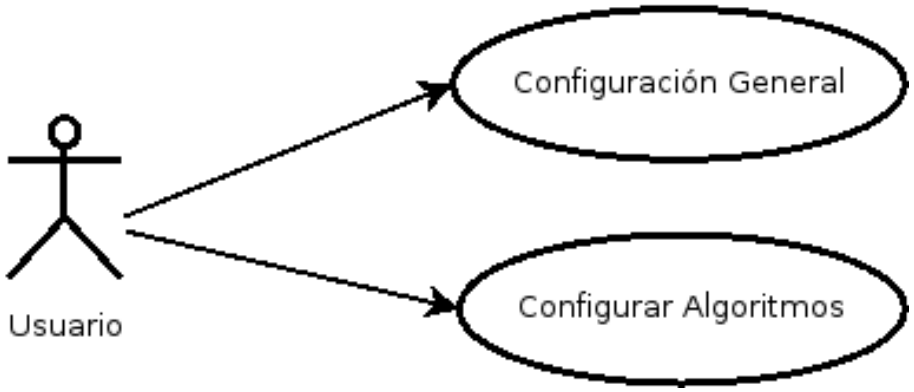


Figura 8.3: Caso de uso configuración experimento

| **Caso de Uso** | Configurar un experimento |
|---|---|
| **Descripción** | El usuario introduce los datos necesarios para el correcto funcionamiento del programa, incluyendo las bases de datos con las que entrenar los modelos y de las que obtener predicciones, las métricas a utilizar, los distintos clasificadores a utilizar, etc. |
| **Actores Principales** | Usuario |
| **Actores Secundarios** | Ninguno |
| **Condiciones Iniciales** | Ninguna |
| **Flujo Principal** | 1. El usuario indica la configuración general del experimento, compuesta por las BBDD, el número de folds con los que realizar la validación cruzada, las métricas a calcular y el número de trabajos a utilizar por el proceso.<br>2. El usuario indica los distintos algoritmos con los que quiere clasificar las bases de datos. Indicando además los hiperparámetros con los que quiere optimizar los modelos durante la fase de validación cruzada. |
| **Condiciones Finales** | Ninguna |
| **Flujos Alternativos** | Ninguno |

Tabla 8.2: Caso de uso configuración experimento

# Capítulo 9

# Tecnologías

En este capítulo se procederá a describir las principales tecnologías a utilizar para el desarrollo del sistema.

## 9.1. Lenguaje de Programación Python

Python es un lenguaje de programación de alto nivel, interpretado y de propósito general, que ha ido adquiriendo popularidad gracias a las múltiples bondades de las que hace gala, entre las que podemos enumerar: su versatilidad, su facilidad de aprendizaje, su amplia biblioteca estándar, etc.

Este incremento de fama también se ha visto reflejado en el mundo académico, gracias en parte a la licencia PSFL (*Python Software Foundation License*), la cual es una licencia de *software* libre parecida a BSD y compatible con GPL. Además, para este lenguaje se han desarrollado una gran cantidad de bibliotecas de libre utilización para el tratamiento de problemas dentro del ámbito de las Ciencias de la Computación, que serán de vital importancia en el desarrollo de nuestro sistema.

### 9.1.1. Breve Historia

El lenguaje de programación Python fue concebido a finales de los años 80 como un sucesor para ABC, aunque también fue influenciado por otros lenguajes tales como Icon, Modula-3 o Perl. Su implementación no comenzó hasta diciembre de 1989, mientras que la primera publicación del mismo no ocurrió hasta febrero de 1991.

Python ha recibido múltiples versiones a lo largo de sus años de vida, siendo la versión `3.x` la más actual. A pesar de la existencia de esta nueva versión principal, debido a la amplia utilización de la versión `2.x` de Python y a ciertas incompatibilidades entre versiones, se ha continuado prestando soporte a la misma, aunque el fin de dicho soporte se encuentra fechado para 2020.

Desde la concepción inicial de Python, Guido van Rossum ha sido el autor principal, así como la persona con la última palabra a la hora de marcar el rumbo que este debía tomar. Esto hizo que, en su momento, la comunidad de Python le otorgara el título de "Benevolente Dictador Vitalicio" (del inglés, *Benevolent Dictator For Life*, abreviado como BDFL). También a él debemos el nombre del lenguaje, referencia al famoso grupo de humoristas británicos, los *Monty Python*. En julio de 2018, Guido van Rossum finalmente dejó su puesto como BDFL.

### 9.1.2. Características Principales

Entre las características que se atribuyen a Python se encuentran las siguientes:

- **Lenguaje interpretado**: el código fuente no necesita ser compilado previamente a su ejecución, sino que un intérprete hará esta traducción en tiempo de ejecución. Este hecho hace que el código escrito en Python sea fácilmente portable a distintos sistemas operativos.

- **Tipado dinámico**: a diferencia de C y Java, en Python no es necesario declarar el tipo de la variable como paso previo a su utilización, sino que estos se definen mediante la asignación de un valor, de un tipo concreto, a la variable. Aunque esta característica suele ser criticada en otros, al ser Python un lenguaje fuertemente tipado, se mitigan la mayor parte de los problemas derivados de esta característica.

- **Multiparadigma**: aunque Python es, principalmente, un lenguaje orientado a objetos, no encasilla al desarrollador en un único paradigma de programación, sino que también soporta otros como la programación imperativa.

- **Fácil de aprender y utilizar**: tiene una sintáxis sencilla que lo hace accesible a personas que se quieran iniciar en él. Cabe mencionar también que, gracias a la necesidad de indentar el código y a su tipado dinámico, se facilita la comprensión del código.

### 9.1.3. Bibliotecas Utilizadas

A lo largo del desarrollo del sistema, se harán uso de múltiples bibliotecas relacionadas con el ámbito de las Ciencias de la Computación. Aquellas bibliotecas externas a la biblioteca estándar de las cuales se hará uso extensivo son las siguientes:

- ***NumPy***: Proporciona objetos para almacenar vectores o matrices multidimensionales, así como una vasta colección de funciones matemáticas para tratar estos objetos de forma sencilla.

- ***Pandas***: Esta biblioteca proporciona estructuras de datos y herramientas de análisis de datos, siendo de gran utilidad a la hora de leer y almacenar datos desde/en ficheros. Al estar integrada con `numpy`, hace trivial el paso de estructuras de una biblioteca a la otra.

- ***Scikit-Learn***: También referida por el nombre del paquete donde está codificada, `sklearn`, implementa una gran cantidad de herramientas para la minería y el análisis de datos. Entre ellas destacan las diferentes clases que contienen clasificadores, métodos de regresión y *clustering*. Además, cuenta con utilidades que realizan labores de preprocesamiento sobre los datos.

  Esta biblioteca se encuentra integrada con `numpy`, `scipy` [20] y `matplotlib` [21] y, al igual que estas, posee una licencia BSD de *software* libre.

- ***Sacred***: Esta herramienta proporciona métodos para configurar, organizar y reproducir experimentos, ayudando a proporcionar al *framework* todas las variables necesarias para ejecutarse mediante ficheros en formato JSON, además de permitir controlar la semilla de la que harán uso las funciones pseudo-aleatorias.

# Capítulo 10

# Arquitectura del Sistema

En este capítulo vamos a desarrollar una estructura modular del sistema y a representar las relaciones de control entre los distintos módulos o subsistemas.

## 10.1. Módulos

La aplicación que vamos a desarrollar está compuesta de tres módulos bien diferenciados:

- ***Algoritmos de clasificación ordinal:*** comprenderá al algoritmo de regresión ordinal que se implementará en este Trabajo. Como el clasificador se va a desarrollar para ser compatible con la biblioteca `scikit-learn`, este podrá ser utilizado para entrenar modelos y clasificar patrones de forma independiente al módulo central.
- ***Módulo de automatización de experimentos:*** módulo central del *framework*, encargado de llevar a cabo los experimentos explicitados mediante ficheros de configuración e interactuar con el resto de módulos. Debe de implementar el comportamiento especificado por los

Requisitos del Sistema.

- ***Módulo de guardado de resultados:*** subsistema encargado de almacenar los resultados de la ejecución de un experimento. Almacena la información obtenida a nivel de partición. Además, genera los resúmenes del experimento una vez que este ha finalizado.

En la figura 10.1 se muestra la composición del sistema en base a sus módulos. Mientras que en el diagrama 10.2 se aprecia la relación del sistema con el usuario, así como la existente entre los distintos subsistemas.

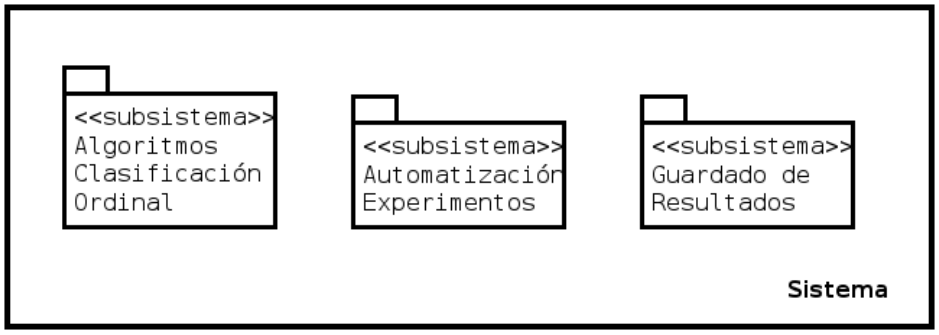


Figura 10.1: Arquitectura del sistema

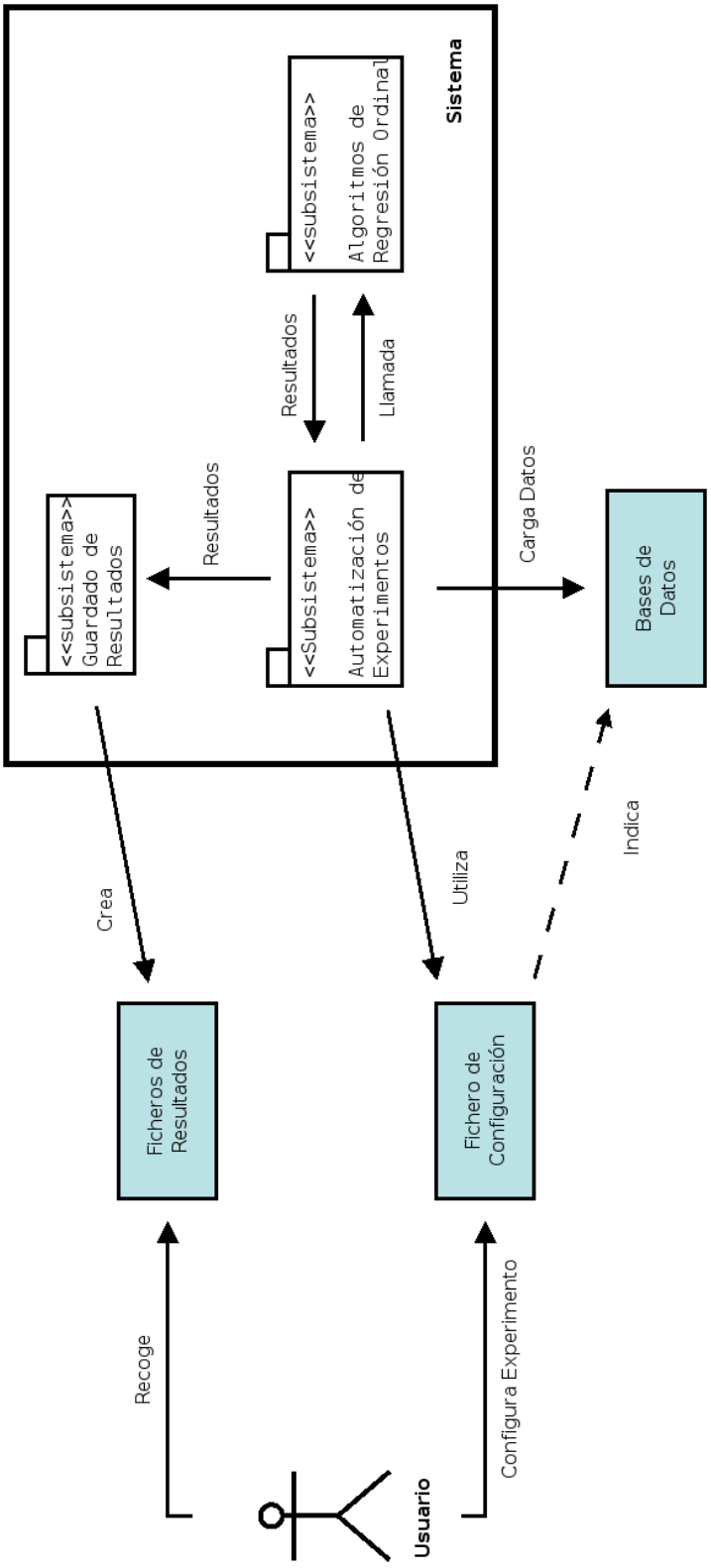


Figura 10.2: Diagrama de relaciones del sistema

# Capítulo 11

# Diagrama de Clases

Este apartado tiene como objetivo refinar el análisis previo del sistema. El paradigma de programación que se utilizará durante la construcción del programa es el de la Programación Orientada a Objetos (POO), ya que nos permite estructurar y separar mejor las distintas funcionalidades que debe incluir el *framework*. Por ello, se utilizarán diagramas de clases como modelo de especificación.

En este capítulo mostraremos todas las clases y funciones que han surgido durante la materialización del sistema desarrollado. Para su mejor comprensión, hemos dividido los diagramas en base a los distintos módulos comentados en la sección 10.1.

## 11.1. Diagramas de Clases

Los diagramas de clases son un tipo de diagrama dentro del *Lenguaje Unificado de Modelado*, abreviado como UML, que sirve para definir la estructura estática de un sistema, compuesta de clases de objetos y las relaciones que existen entre ellos. Para cada clase se definirán también sus atributos y métodos.

Existen distintos tipos de abstracción y de relaciones entre objetos, pero en nuestro sistema únicamente hemos utilizado dos tipos: la relación de asociación y la de herencia.

A la hora de describir cada clase existente en el sistema, utilizaremos la notación que se muestra en la Tabla 11.1.

| Clase: nombre | | |
|---|---|---|
| Descripción de la Clase | | |
| Datos | | |
| variable1 | tipo variable | descripción de la variable |
| ... | ... | |
| Métodos | | |
| método1 | tipo de método | descripción del método |
| ... | ... | |

Tabla 11.1: Ejemplo general para la especificación de clases

## 11.2. Paquete de Automatización de Experimentos

En esta sección se tratán los distintos módulos *software* que conforman el *framework* que se ha ido desarrollando en el presente Trabajo.

Las relaciones entre los distintos módulos se muestran en el diagrama de clases de la figura 11.1. En ella podemos ver, además de las distintas clases, dos componentes: *Clasificadores*, que sirve para englobar las clases que se explicarán en la sección 11.3, y *Resultados*.

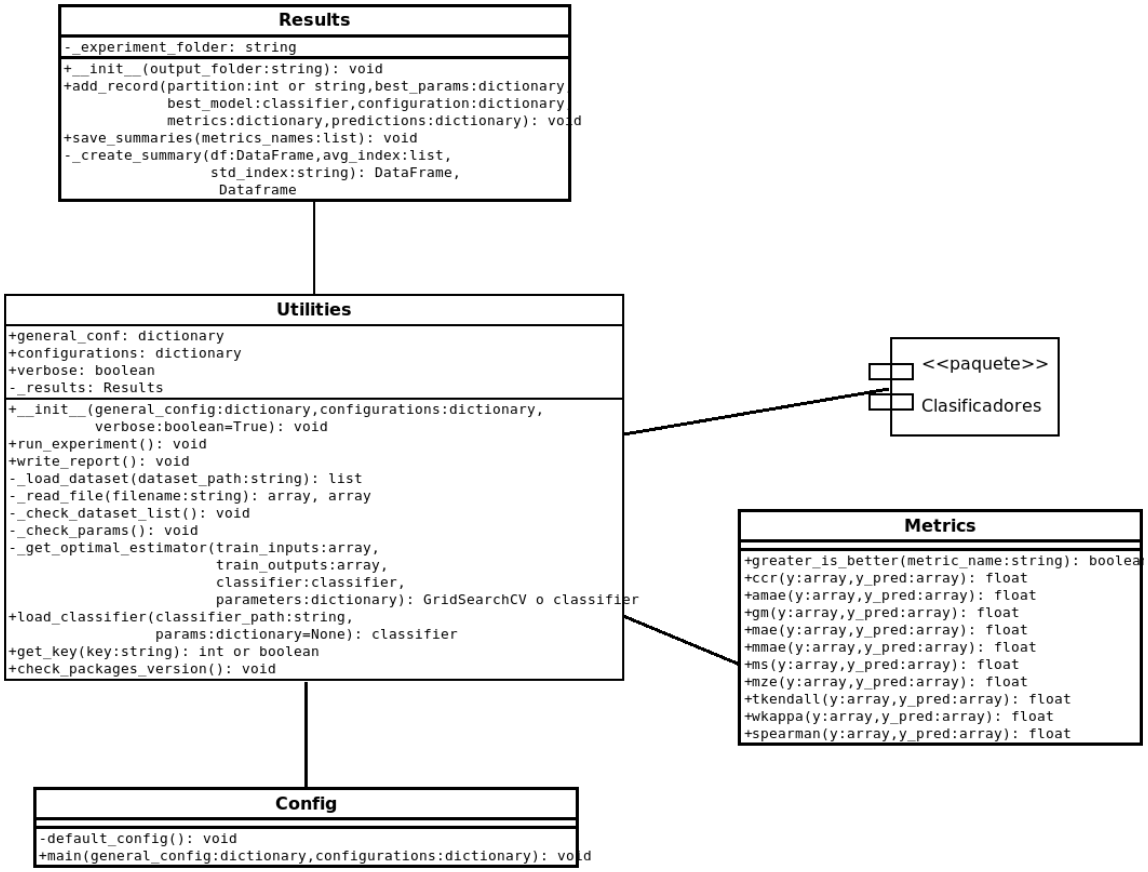


Figura 11.1: Diagrama del paquete Automatización de Experimentos

### 11.2.1. Módulo *Config*

Módulo que actúa de interfaz entre la funcionalidad del sistema y el usuario mediante ficheros de configuración en formato JSON. Para la interpretación de los ficheros de configuración y la automatización de otros elementos se hace uso de la biblioteca `sacred`.

Las funciones que integran el módulo se muestran en la tabla 11.2.

| **Módulo:** *Config* | | |
|---|---|---|
| Conjunto de métodos que sirven de interfaz para proporcionar al sistema un método efectivo para la automatización de experimentos. | | |
| **Métodos** | | |
| ***default_config*** | *void* | Otorga valores por defecto a varios parámetros necesarios para la ejecución del experimento. Estas variables se incluyen dentro del diccionario *general_config* |
| ***main*** | *void* | Recoge la información del fichero de configuración especificado por línea de comandos y llama al método principal de este paquete para llevar a cabo el experimento. |

Tabla 11.2: Especificación del módulo *Config*

### 11.2.2. Módulo *Metrics*

Conjunto de funciones que implementan distintas métricas, utilizadas para medir el rendimiento de un clasificador determinado.

Las funciones que integran el módulo se muestran en la tabla 11.3.

| **Módulo:** *Metrics* | | |
|---|---|---|
| Conjunto de métodos que implementan distintas métricas de rendimiento de clasificadores, o funciones relacionadas con la correcta integración de estas con el sistema. | | |
| **Métodos** | | |
| ***greater_is_better*** | *boolean* | Determina si los valores que implican mejor rendimiento son los cercanos a 0 o los cercanos a 1. |
| ***ccr*** | *float* | Implementa la métrica *Correctly Classified Ratio*. |
| ***amae*** | *float* | Implementa la métrica *Average MAE*. |
| ***gm*** | *float* | Implementa la métrica *Geometric Mean*. |
| ***mae*** | *float* | Implementa la métrica *Mean Absolute Error*. |
| ***mmae*** | *float* | Implementa la métrica *Maximum MAE*. |
| ***ms*** | *float* | Implementa la métrica *Minimum Sensitivity*. |
| ***mze*** | *float* | Implementa la métrica *Mean Zeroone Error*. |
| ***tkendall*** | *float* | Implementa la métrica basada en el estadístico $t$ de Kendall. |
| ***wkappa*** | *float* | Calcula el estadístico *Weighted Kappa*. |
| ***spearman*** | *float* | Calcula el coeficiente de correlación de Spearman. |

Tabla 11.3: Especificación del módulo *Metrics*

### 11.2.3. Clase *Results*

Esta clase manejará todo lo relativo al guardado de los resultados obtenidos durante la ejecución de un experimento. Volcará en disco la información generada a nivel de partición para cada par configuración-base de datos, evitando así pérdidas de información y tiempo en caso de que el programa termine su ejecución de forma inesperada.

Las distintas variables y métodos implementados en esta clase se encuentran detallados en la tabla 11.4.

| **Clase:** *Results* | | |
|---|---|---|
| Clase que gestiona la creación de los ficheros de actividad de un experimento. La información estará recogida en un resumen de la ejecución del experimento a nivel global, así como los distintos modelos, predicciones y valores de métricas obtenidos por el par configuración-BBDD a nivel de partición. | | |
| **Datos** | | |
| ***_experiment_folder*** | *string* | Carpeta base donde se almacenarán los resultados de la ejecución del experimento. De ella colgarán las subcarpetas específicas a cada algoritmo-BBDD. |
| **Métodos** | | |
| ***__init__*** | *void* | Constructor de la clase *Results*. |
| ***add_record*** | *void* | Guarda en disco los datos de ejecución de una partición para una combinación BBDD-configuración. |

| | | |
|---|---|---|
| ***save_summaries*** | *void* | Crea y guarda el resumen de un experimento, donde cada par configuración-BBDD está representado por una fila que representará la media y desviación típica para las distintas métricas a lo largo de todas las particiones. Existirá un resumen para los datos de entrenamiento y otro para los de generalización. |
| ***_create_summary*** | *DataFrame*, *DataFrame* | Condensa la información relativa a una configuración en una única línea. Creará una línea resumen para los datos de entrenamiento y otra para los datos de generalización. |

Tabla 11.4: Especificación de la clase *Results*

### 11.2.4. Clase *Utilities*

Clase central de este *framework*. Lleva a cabo los experimentos, ejecutando para cada conjunto de datos todas las distintas configuraciones de algoritmos de clasificación, realizando un proceso de validación cruzada en el que se seleccionan los mejores valores de cada hiperparámetro de entre los indicados (se crea un modelo óptimo por partición de BBDD). Almacena los resultados a través de la clase *Results*.

Las distintas variables y métodos implementados en esta clase se encuentran detallados en la tabla 11.5.

| **Clase:** *Utilities* |
|---|
| Clase que lleva a cabo un experimento. Integra los demás módulos del sistema. |
| **Datos** |

| | | |
|---|---|---|
| ***general_conf*** | *dictionary* | Diccionario que contiene variables necesarias para la ejecución del experimento, tales como la carpeta donde encontrar los ficheros de BBDD, los nombres de los conjuntos de datos, el número de *folds* a usar durante el proceso de validación cruzada, etc. |
| ***configurations*** | *dictionary* | Estructura de datos que contiene, a su vez, tantos diccionarios como configuraciones de algoritmos de clasificación diferentes se quieran aplicar. |
| ***verbose*** | *boolean* | Indica el grado de información que se desea mostrar en la terminal. Utilizado para depuración y ejecución de tests. |
| ***_results*** | *Results* | Instancia de la clase *Results* utilizada para guardar los resultados conforme se obtengan. |
| **Métodos** | | |
| ***__init__*** | *void* | Constructor de la clase *Utilities*. |

| | | |
|---|---|---|
| ***run_experiment*** | *void* | Lleva a cabo toda la funcionalidad relativa a la ejecución del experimento, apoyándose en el resto de métodos. |
| ***write_report*** | *void* | Llama al método oportuno de *Results* para volcar a disco los resultados. |
| ***_load_dataset*** | *list* | Devuelve una lista conteniendo todas las particiones ordenadas para un determinado conjunto de datos, siendo cada elemento de la lista un diccionario que divide la partición en datos de entrenamiento y de generalización. |
| ***_read_file*** | *array*, *array* | Carga los datos de un fichero CSV de formato apropiado, dividiendo los datos del conjunto en dos arrays, uno con las características de los patrones, el otro con las etiquetas de clase a las que pertenece cada patrón. |
| ***_check_dataset_list*** | *void* | Este método trata de detectar errores en la lista de conjuntos de datos provista en el fichero de configuración. |

| | | |
|---|---|---|
| ***_check_params*** | *void* | Realiza algunas transformaciones sobre los parámetros de entrada con respecto al correcto uso de las listas. También modifica el diccionario de parámetros en caso de que un método de ensemble se haya usado, para hacerlo compatible con la clase `GridSearchCV` de `scikit-learn`. |
| ***_get_optimal_estimator*** | *GridSearchCV* o, *classifier* | Realiza el proceso de validación cruzada si es pertinente, automatizando la búsqueda del mejor conjunto de hiperparámetros de entre todas las combinaciones posibles. Devuelve el mejor modelo encontrado en base al conjunto de datos de validación. |

| | | |
|---|---|---|
| ***load_classifier*** | *classifier* | Carga de forma dinámica clases que implementan algortimos de clasificación (no necesariamente de regresión ordinal), ya sea desde una dirección relativa al paquete de `scikit-learn`, o de entre los algoritmos implementados en el seno del *framework*. |
| ***get_key*** | *int o string* | Comprueba si la clave de un diccionario se puede convertir a entero. Si es posible, lo devuelve, sino devuelve la clave tal cual. |

Tabla 11.5: Especificación de la clase *Utilities*

## 11.3. Paquete Clasificadores

En este paquete se incluye el algoritmo de regresión ordinal desarrollado, *OrdinalDecomposition*, el cual forzosamente debe ser una clase derivada de dos tipos de objetos definidos en `scikit-learn`: *BaseEstimator* y *ClasifierMixin*. En estas clases base se definen (e implementan) varios métodos comunes a todos los clasificadores.

El diagrama de clases que representa las relaciones entre las clases de este paquete se muestra en la figura 11.2.

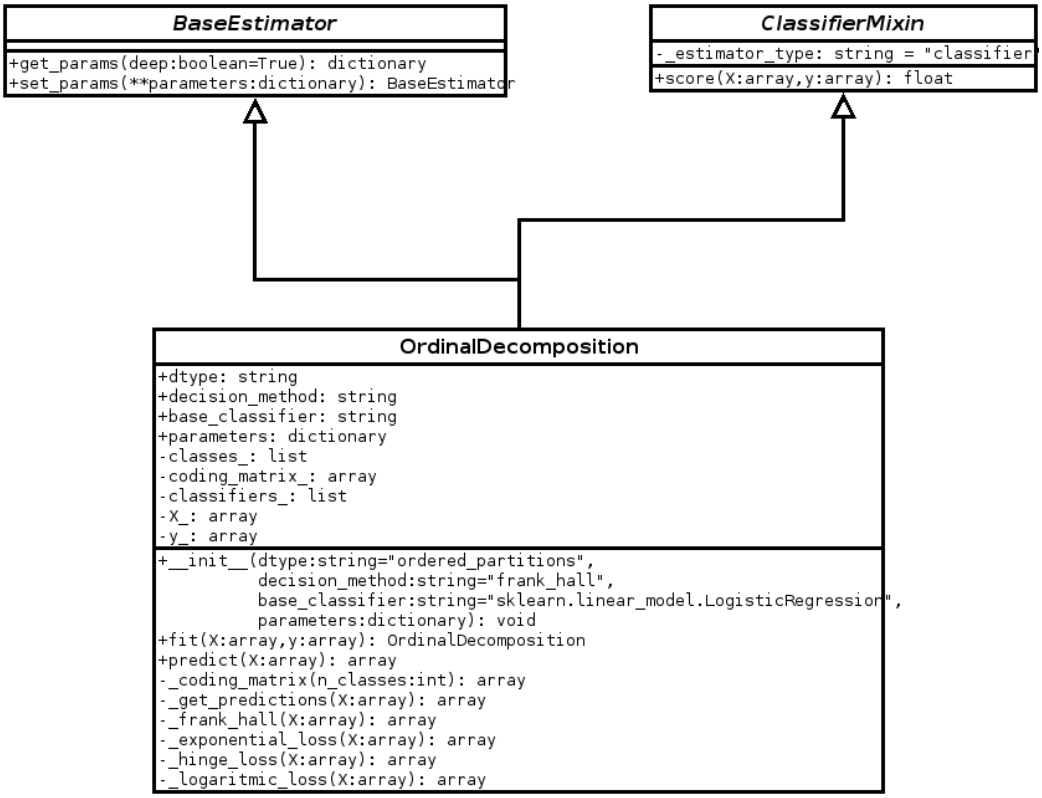


Figura 11.2: Diagrama de clases del paquete Clasificadores

### 11.3.1. Clases *BaseEstimator* y *ClasifierMixin*

Estas clases base de las que hereda nuestro algoritmo se encuentran implementadas dentro del paquete `sklearn` de Python. Aunque en el diagrama en el que se representan las relaciones entre clases aparecen ambas, esta es una representación mínima de las funciones públicas. Para más detalle acerca de sus métodos privados, se recomienda visitar la documentación de cada clase, recogida en [22] y [23] respectivamente.

### 11.3.2. Clase *OrdinalDecomposition*

Clase que implementa nuestro algoritmo de regresión ordinal. Para la correcta integración con el paquete central, debe seguir la notación propia de los clasificadores desarrollados dentro de la biblioteca `scikit-learn`, definiendo, al menos, los métodos públicos indispensables para su correcto funcionamiento. Los métodos requeridos son: el encargado del entrenamiento del modelo, *fit*, el encargado de obtener una predicción en base a los datos de generalización y a los modelos previamente generados, *predict*, y el constructor de la clase, que recibe los distintos parámetros con los que se puede modificar el comportamiento del mismo.

En la tabla 11.6 se puede observar la clase que implementa al algoritmo de descomposición ordinal.

| **Clase:** *OrdinalDecomposition* | | |
|---|---|---|
| Clase que codifica el método de clasificación ordinal por descomposición en varios subproblemas nominales relacionados mediante una matriz de códigos. | | |
| **Datos** | | |
| ***dtype*** | *string* | Tipo de descomposición ordinal que el algoritmo llevará a cabo. |
| ***decision_method*** | *string* | Modo en que se combinan las decisiones de los $n$ clasificadores nominales para dar respuesta al problema ordinal de $n+1$ clases. |
| ***base_classifier*** | *string* | Dirección desde la que se carga el algoritmo interno. |
| ***parameters*** | *dictionary* | Hiperparámetros para validar en el clasificador interno. |
| ***classes_*** | *list* | Lista de diferentes valores únicos que representan a las distintas clases del problema ordinal. |

| | | |
|---|---|---|
| ***coding_matrix_*** | *array* | Matriz de códigos con la que descomponer el problema original. |
| ***classifiers_*** | *list* | Lista de clasificadores nominales del tipo base_classifier. |
| ***X_*** | *array* | Matriz que contiene los valores de las distintas variables (columnas) para cada patrón (filas) de la BBDD. |
| ***y_*** | *array* | Vector que contiene la etiqueta de clase para cada patrón del problema. |
| **Métodos** | | |
| ***__init__*** | *void* | Constructor de la clase. |
| ***fit*** | *self* | Entrena el modelo en base a los datos de entrada y los valores de los parámetros proporcionados. |
| ***predict*** | *array* | Devuelve las etiquetas de clase predichas para un conjunto de datos. |
| ***_coding_matrix*** | *array* | Construye la matriz de códigos. |
| ***_get_predictions*** | *array* | Devuelve la probabilidad de pertenencia de cada patrón a la clase positiva. |
| ***_frank_hall*** | *array* | Transforma la predicción individual de $n$ clasificadores binarios en un único valor, que será la etiqueta de una de las $n+1$ clases del problema original. Para ello usa el método de Frank y Hall. |
| ***_exponential_loss*** | *array* | Devuelve el valor de pérdida exponencial para cada patrón en base a las probabilidades arrojadas por los clasificadores binarios. |
| ***_hinge_loss*** | *array* | Devuelve el valor de pérdida exponencial para cada patrón en base a las probabilidades arrojadas por los clasificadores binarios. |

| | | |
|---|---|---|
| ***_logarithmic_loss*** | *array* | Devuelve el valor de pérdida logarítmica para cada patrón en base a las probabilidades arrojadas por los clasificadores binarios. |

Tabla 11.6: Especificación de la clase *OrdinalDecomposition*

# Parte IV

# Pruebas

# Capítulo 12

# Pruebas

La fase de pruebas es una parte fundamental en el desarrollo de cualquier sistema *software*, ya que estas aseguran que se cumplan, al menos, los requisitos de funcionalidad que las mismas les imponen. Idealmente, para que la probabilidad de encontrar algun fallo o inconsistencia sea mayor, este apartado debería ser realizado por personas que no hayan estado involucradas en el desarrollado de la funcionalidad a probar, ya que este último tratará de probar, inconscientemente, que el *software* funciona, en lugar de buscar las partes donde no se comporta como se espera.

A lo largo del desarrollo del sistema se han realizado una serie de controles para asegurar la fiabilidad del mismo, sirviendo también para la detección de errores en fases tempranas de implementación.

Debido al paradigma de programación utilizado, nuestro sistema está compuesto de un conjunto de componentes que interactúan entre sí. Por ello es necesario probar la operatividad de los componentes de forma individual, antes de proceder a testar el funcionamiento sistémico del *framework*, para aseverarnos de que se comportan de la forma esperada, ya que si falla un componente, difícilmente funcionará el sistema que lo utiliza.

En este capítulo se hará una breve introducción teórica sobre el funda-

mento y finalidad de las pruebas, exponiendo a continuación algunas de las pruebas realizadas.

## 12.1. Estrategia de Pruebas

Una estrategia de pruebas efectiva es aquella que, mediante el diseño y desarrollo de casos de prueba, permite encontrar defectos existentes en el sistema, y a la vez conseguir que los resultados obtenidos sean lo suficientemente buenos con respecto a un margen establecido. Esto es, en caso de que los resultados o tiempos de ejecución no fueran aceptables, habría que optimizar y mejorar el sistema.

Como entorno de pruebas se han desarrollado una serie de pruebas denominadas: *pruebas de caja blanca* y *pruebas de caja negra*. Los resultados de estas pruebas han sacado a relucir algunos errores, los cuales han sido resueltos volviendo a fases anteriores y, de forma cíclica, aplicando los test sobre el nuevo código ya revisado.

Por tanto, el procedimiento utilizado para la eliminación de errores es la *vuelta atrás*, esto es, localizar el síntoma que hizo que el subsistema no superase el caso de prueba satisfactoriamente yendo hacia atrás hasta llegar a la causa.

Los casos de prueba pueden ser clasificados en cuatro tipos diferentes, los cuales se expondrán a continuación.

## 12.2. Pruebas Unitarias

Este tipo de pruebas tiene como misión la comprobación del funcionamiento de la menor unidad de código, el módulo, que comprende a las funciones no triviales, con extensiones no muy grandes, que se pueden probar de forma independiente. De esta manera nos aseguramos que cada unidad

funcione correctamente por separado antes de tratar la integración dentro del sistema.

Este apartado se compone de dos tipos de pruebas: las *pruebas de caja blanca* o pruebas estructurales 12.2.1, y las *pruebas de caja negra* o pruebas funcionales 12.2.2.

### 12.2.1. Pruebas de Caja Blanca

Las pruebas de caja blanca están asociadas a la fase de diseño, centrándose en validar el funcionamiento de los distintos módulos del sistema por separado, asegurándose de que cumplen con lo esperado. También se utilizan para testar los módulos en casos límite, o comprobar el comportamiento ante casos de errores contemplados durante la codificación.

Las pruebas estructurales se encuentran íntimamente ligadas a una implementación concreta, por lo que si se producen modificaciones sobre el código fuente de un módulo, probablemente sea necesario modificar este tipo de pruebas para adaptarse a la nueva versión.

La realización de estas pruebas durante el desarrollo del sistema se justifica gracias al soporte que proveen en el descubrimiento de nuevos errores no contemplados previamente, que ayudarán al módulo a evolucionar a un estado más óptimo y estable.

Estas pruebas se han realizado de la siguiente forma:

1. Se comprueba si las estructuras de datos que maneja cada uno de los módulos se trata de la forma en que se espera.

2. Se comprueba que el flujo de datos a través de las distintas bifurcaciones se realiza adecuadamente.

3. Se comprueba la ejecución de cada una de las sentencias que forman parte de un módulo analizando la ejecución completa de los mismos.

### 12.2.2. Pruebas de Caja Negra

Al contrario que en las pruebas de caja blanca, los procedimientos que se llevan a cabo dentro de los distintos módulos son transparentes a las pruebas de caja negra, las cuales únicamente se cercionarán de que el flujo de información de salida de un módulo es el esperado acorde a un flujo de información de entrada determinado (en otras palabras, se centran en la interfaz del módulo).

Estas pruebas funcionales, que se fundamentan en la especificación de requisitos del sistema, se han desarrollado de forma paralela y posterior a la codificación de los distintos módulos que conforman el sistema.

Los casos de prueba asociados a este tipo de test son relativamente cortos, ya que se limitan a cargar o definir un flujo de datos de entrada, llamar al método y comprobar la salida obtenida del mismo. Para ello se hace uso de distintas bases de datos y configuraciones en los casos que sea necesario.

## 12.3. Pruebas de Integración

Una vez desarrolladas y verificadas las pruebas unitarias sobre los distintos módulos del *software*, se procede a la generación de las pruebas integrales o pruebas de integración. Este conjunto de test tiene por misión comprobar que los distintos elementos unitarios del sistema se encuentran bien cohesionados, esto es, que los distintos módulos de los que se compone un proceso funcionan acorde a lo proyectado.

Las pruebas de integración se han realizado de forma incremental: se añadían módulos al caso de prueba conforme estos se validaban, comprobando entonces como se comportaba este con el conjunto de módulos que previamente habían sido probados juntos. El último paso incremental corresponde al de evaluar la funcionalidad del sistema al completo.

Por tanto, el proceso de construcción de este tipo de pruebas se ha realizado de la siguiente forma:

1. Se comprueba en cada incremento que el comportamiento del nuevo conjunto de módulos es el esperado.

2. Se comprueba que el flujo de información entre módulos es correcto.

En esta fase, se comprueba que el subsistema encargado de almacenar la información de un experimento se encuentra correctamente validado, corroborando posteriormente el funcionamiento del sistema de gestión de experimentos al completo.

## 12.4. Pruebas del Sistema

Tras asegurarnos de que las pruebas de las secciones anteriores están correctamente implementadas y que nuestro sistema funciona de forma correcta, se comienza la fase final de pruebas, donde se asevera que todos los componentes del sistema están bien acoplados y llevan a cabo la funcionalidad esperada.

Esta fase tiene como objetivo verificar que el sistema *software* cumple con sus requisitos. Dentro de ella pueden desarrollarse varios tipos de pruebas en función de los objetivos, que pueden ser funcionales, de usabilidad, de rendimiento, ...

Los resultados asociados a este tipo de pruebas se analizarán en el capítulo 13. De esta forma, se cumple con el objetivo de probar el sistema con distintas bases de datos y configuraciones, además de observar si los algoritmos implementados cumplen con los objetivos expuestos en capítulos previos.

## 12.5. Pruebas de Aceptación

Estas prueban son realizadas por los usuarios finales del sistema. Son básicamente pruebas funcionales, sobre el sistema ya desplegado que buscan una cobertura de la especificación de requisitos. Estas pruebas se llevan a cabo una vez pasadas las de integración, ya que en fases anteriores el sistema no es completamente funcional o puede contar con errores que el desarrollador es capaz de reconocer. Sin embargo, por el hecho de que las personas encargadas de desarrollar el sistema no son capaces de comportarse como un usuario lego ante el programa, suelen aparecer errores no previstos cuando el usuario lo utiliza.

Este tipo de pruebas se han realizado en dos ubicaciones diferentes: en el entorno de desarrollo del sistema y en el entorno de trabajo del usuario final.

Las primeras se realizan invitando a un usuario a probar el sistema en el entorno de desarrollo, el cual está más controlado, puesto que dispondrá de un experto para guiarlo y para ayudarle a analizar los resultados obtenidos por la ejecución del *software*.

Por el contrario, el otro tipo de pruebas se realizan en el lugar de trabajo del usuario, sin supervisión del desarrollador. El usuario tratará de encontrar fallos al sistema, ya sean estos reales o potenciales, de los cuales informará al equipo de desarrollo. También puede notificar cambios para mejorar la comodidad o eficiencia del sistema, adaptándolo así el máximo posible a sus necesidades.

La cuestión principal a evaluar suele ser la fiabilidad y la facilidad de uso del sistema. En nuestro caso, los usuarios finales son los directores del Trabajo, que instruyen sobre distintas cuestiones para mejorar la comodidad de uso del programa, encontrar fallos que el proyectista no tuviese en cuenta y probar la fiabilidad del mismo.

## 12.6. Casos de Prueba

En esta sección se expondrán los casos de prueba que se han generado durante el desarrollo del sistema, de los cuales, la mayor parte, implementan pruebas unitarias. Los casos de prueba se encuentran agrupados en cuatro bloques distintos, según la funcionalidad que prueba cada uno de ellos. De este modo, existirá un conjunto de pruebas por cada una de las clases del sistema, `1.x` para `Utilities`, `2.x` `Results` y `3.x` `OrdinalDecomposition`, y otro conjunto de test para comprobar las métricas de rendimiento, `4.x`.

### 12.6.1. Casos de Prueba 1.1 - 1.4

Estos casos de prueba corroboran que la lectura de las particiones se realiza correctamente. Existe un caso de prueba por cada una de las siguientes situaciones: el conjunto de datos no está particionado, el conjunto de datos cuenta con particiones, el conjunto de datos no contiene datos de test y el conjunto de datos no dispone de datos de entrenamiento.

Estos casos de prueba se encuentran representados en la tabla 12.1.

### 12.6.2. Caso de Prueba 1.5

El caso de prueba definido en la tabla 12.2 verifica la funcionalidad del método encargado de cargar un algoritmo de clasificación dado su *path*.

### 12.6.3. Caso de Prueba 1.6

Este caso de prueba, representado en la tabla 12.3, comprueba que la lista de parámetros del experimento se formatea de manera adecuada para poder ser utilizada por el objeto encargado de llevar a cabo la validación cruzada.

| Caso | Descripción | Resultado Esperado |
|---|---|---|
| 1.1 | Carga de una base de datos sin particiones, con un fichero de entrenamiento y otro de test. | La carga se produce de forma correcta. |
| 1.2 | Carga de una base de datos con varias particiones, cada una con un archivo de entrenamiento y otro de test. | La carga se produce de forma correcta. |
| 1.3 | Carga de una base de datos con varias particiones, de las cuales ninguna cuenta con ficheros de test. | La carga se produce de forma correcta. |
| 1.4 | Carga de una base de datos con particiones que únicamente cuentan con ficheros de test. | El programa devuelve una excepción al intentar cargar la base de datos. |

Tabla 12.1: Casos de prueba CP 1.1 - CP 1.4

| Caso | Descripción | Resultado Esperado |
|---|---|---|
| a | Carga de un clasificador local. | Se devuelve la clase de objeto esperada. |
| b | Carga de un clasificador de `sklearn`. | Se devuelve la clase de objeto esperada. |
| c | Intento de carga de un clasificador inexistente. | El programa devuelve una excepción al intentar cargar el clasificador. |

Tabla 12.2: Caso de prueba CP 1.5

| Caso | Descripción | Resultado Esperado |
|---|---|---|
| a | Uso de una configuración con un clasificador con varios parámetros, cada uno de ellos con uno o más valores. | La lista una vez formateada corresponde con la esperada. |
| b | Uso de una configuración que utiliza como clasificador un ensemble, en concreto `OrderedPartitions`. Se definen una serie de parámetros anidados. | La lista de parámetros formateados es la esperada. |
| c | Uso de una configuración cuyos parámetros solo cuentan con un valor por parámetros. | Ninguno de los parámetros cuenta con una lista de valores, solo un valor individual. |

Tabla 12.3: Caso de prueba CP 1.6

### 12.6.4. Caso de Prueba 1.7

Este caso, mostrado en la tabla 12.4, trata de ejecutar un experimento de la misma forma en la que lo haría un usuario final, comprobando después que todos los ficheros de salida han sido generados y tienen un formato correcto. Implementa las pruebas del sistema.

| Caso | Descripción | Resultado Esperado |
|---|---|---|
| | Ejecuta un experimento con una configuración y bases de datos reales. | La ejecución termina de forma satisfactoria, habiéndose generado todos los ficheros de salida pertinentes. |

Tabla 12.4: Caso de prueba CP 1.7

### 12.6.5. Caso de Prueba 2.1

El caso de prueba de esta subsección se encuentra representado en la tabla 12.5. En él se constata que la funcionalidad implementada en el méto-

do `add_record` de la clase `Results` realiza las tareas encomendadas por los requisitos del sistema.

| Caso | Descripción | Resultado Esperado |
|---|---|---|
| a | Guardado de la información relativa a dos configuraciones distintas. Una de ellas con dos particiones, la otra sin particiones. | La información se guarda en el sistema de ficheros sin errores de escritura. |
| b | Comprobamos los valores de las métricas almacenados. | Los valores leídos de disco son idénticos a los valores escritos. |
| c | Comprobamos los objetos que contienen las instancias de los modelos generados. | Los objetos cargados en disco son una instancia del clasificador apropiado. |
| d | Comprobamos las predicciones realizadas por los modelos. | Las predicciones de entrenamiento y de test son similares a las almacenadas. |

Tabla 12.5: Caso de prueba CP 2.1

### 12.6.6. Caso de Prueba 2.2

En el caso de prueba ilustrado en la tabla 12.6, se comprueba si los resúmenes que se generan tras finalizar la ejecución de un experimento se atienen a lo esperado. Concretamente, tratan de cerciorarse del correcto funcionamiento del método `create_summary`.

### 12.6.7. Caso de Prueba 3.1

En este test, expuesto en la tabla 12.7, se espera que el clasificador `OrderedPartitions` clasifique correctamente varias instancias de un problema sencillo, consistente en una serie de puntos en un plano bidimiensional. Este caso nos servirá como prueba de integración de los distintos métodos de la clase.

| Caso | Descripción | Resultado Esperado |
|---|---|---|
| a | Creamos y guardamos los datos pertenecientes a dos particiones de una misma configuración. | La información se guarda sin errores. |
| b | Comprobamos que los valores del resumen son los esperados. | Los valores tras el uso del método `create_summary` son similares a los calculados. |

Tabla 12.6: Caso de prueba CP 2.2

| Caso | Descripción | Resultado Esperado |
|---|---|---|
| | Entrenamos un modelo con los mismos datos que trataremos de predecir. | Se predicen los puntos dentro de las clases reales. |

Tabla 12.7: Caso de prueba CP 3.1

### 12.6.8. Caso de Prueba 3.2

Este caso comprueba que las matrices de codificación generadas para un número determinado de clases (5 en nuestro caso) se atienen a lo esperado. El caso de prueba se expone en la tabla 12.8.

### 12.6.9. Casos de Prueba 3.3 - 3.6

En los casos de prueba que se exponen en la tabla 12.9, se comprueba que los distintos métodos de decisión del clasificador desarrollado funcionen en base a lo expuesto en la sección 4.4 del capítulo de Antecedentes. Se parte de un supuesto problema de 5 clases, para el cual disponemos de los valores predichos por cada clasificador binario para una serie de instancias.

| Caso | Descripción | Resultado Esperado |
|---|---|---|
| a | Se genera una matriz de descomposición del tipo `ordered_partitions` para un problema de 5 clases. | Los valores de la matriz son los esperados. |
| b | Se genera una matriz de descomposición del tipo `one_vs_next` para un problema de 5 clases. | Los valores de la matriz son los esperados. |
| c | Se genera una matriz de descomposición del tipo `one_vs_followers` para un problema de 5 clases. | Los valores de la matriz son los esperados. |
| d | Se genera una matriz de descomposición del tipo `one_vs_previous` para un problema de 5 clases. | Los valores de la matriz son los esperados. |

Tabla 12.8: Caso de prueba CP 3.2

### 12.6.10. Casos de Prueba `4.x`

Estos casos de prueba verifican que, en base a las etiquetas de clase reales y predichas para una serie de patrones, los valores devueltos por cada una de las métricas de evaluación del rendimiento son correctos. Existe un caso de prueba por cada una de las métricas, diez en total.

| Caso | Descripción | Resultado Esperado |
|---|---|---|
| 3.3 | Utilizamos el método `frank_hall` para calcular la probabilidad de pertencia de cada instancia a una de las cinco clases posibles. | Las predicciones son similares a las esperadas. |
| 3.4 | Utilizamos el método `exponential_loss` para calcular la pérdida exponencial de cada patrón con respecto a cada una de las clases. | La matriz de pérdidas es similar a la esperada. |
| 3.5 | Utilizamos el método `logarithmic_loss` para calcular la pérdida logarítmica de cada patrón con respecto a cada una de las clases. | La matriz de pérdidas es similar a la esperada. |
| 3.6 | Utilizamos el método `hinge_loss` para calcular la pérdida de cada patrón con respecto a cada una de las clases. | La matriz de pérdidas es similar a la esperada. |

Tabla 12.9: Casos de prueba CP 3.3 - CP 3.6

# Capítulo 13

# Resultados Experimentales

Una vez desarrollado el sistema y tras la primera fase de pruebas, es conveniente realizar un conjunto de pruebas experimentales a nivel de sistema que nos permitan:

1. Comprobar el correcto funcionamiento del algoritmo de descomposición ordinal.

2. Realizar pruebas de estrés reales sobre el sistema para completar el proceso de pruebas.

3. Realizar una búsqueda de los parámetros de configuración óptimos para el algoritmo desarrollado, con la intención de obtener los mejores resultados posibles sobre una serie de conjuntos de datos.

4. Contraponer los resultados obtenidos por nuestro clasificador frente a otros algoritmos de regresión ordinal contrastados, con el objeto de comprobar que dichos resultados son satisfactorios.

Con la intención de que este capítulo sea lo más claro posible, dividiremos el mismo en dos secciones: una primera donde se definirá la estrategia experimental que se planea seguir, consistente en un conjunto de configuraciones

de algortimos de regresión ordinal y bases de datos sobre las que construir los modelos, y una segunda parte en la que se comentarán los datos obtenidos durante dicha fase de experimentación.

## 13.1. Diseño Experimental

### 13.1.1. Conjuntos de Datos Seleccionados

Las bases de datos que hemos seleccionado pueden ser catalogadas como de dos tipos diferentes:

- **Bases de datos de regresión discretizadas**: Estos conjuntos de datos no representan problemas de regresión ordinal reales, sino que consisten en problemas de regresión donde los distintos valores de salida posibles han sido discretizados en 5 clases de igual frecuencia. Cada una de estas bases de datos se encuentran fraccionadas en 20 particiones obtenidas mediante la realización de dicho número de *hold-outs* sobre una misma base de datos.
- **Bases de datos de clasificación ordinal**: A diferencia de las anteriormente mencionadas, estas BBDD representan problemas de clasificación ordinal reales, pudiendo observarse una falta de equilibrio entre el número de patrones de cada clase. Constan de 30 particiones cada una, obtenidas de forma similar a las del tipo anterior.

Las bases de datos de regresión han sido obtenidas del repositorio proporcionado por Chu y Ghahramani [24], mientras que los conjuntos que contienen problemas de tipo ordinal han sido extraidos de los repositorios de dominio público de la Universidad de California [25] y *mldata.io* [26].

Para la realización de los experimentos se ha decidido trabajar con cinco bases de datos de cada uno de los tipos expuestos, considerándose este

subconjunto como suficientemente diverso en cuanto a características se refiere. Cada partición de estas bases de datos se encuentra dividida en un subconjunto de entrenamiento y otro de test.

Se pueden apreciar las características de las distintas bases de datos seleccionadas en la tabla 13.1.

| **Bases de datos de regresión discretizadas** | | | | |
|---|---|---|---|---|
| Nombre BBDD | #Inst. | #Atr. | #Clas. | Distribución Clas. |
| *computer5* (**C5**) | 8192 | 12 | 5 | $\approx$ 1639 por clase |
| *machine5* (**M5**) | 209 | 7 | 5 | $\approx$ 42 por clase |
| *bank5* (**B5**) | 8192 | 8 | 5 | $\approx$ 1639 por clase |
| *housing5* (**H5**) | 506 | 14 | 5 | $\approx$ 101 por clase |
| *pyrim5* (**P5**) | 74 | 27 | 5 | $\approx$ 15 por clase |
| **Bases de datos de clasificación ordinal** | | | | |
| Nombre BBDD | #Inst. | #Atr. | #Clas. | Distribución Clas. |
| *bondrate* (**BO**) | 57 | 37 | 5 | (6,33,12,5,1) |
| *tae* (**TA**) | 151 | 54 | 3 | (49,50,52) |
| *squash-stored* (**SS**) | 52 | 51 | 3 | (23,21,8) |
| *newthyroid* (**NT**) | 215 | 5 | 3 | (30,150,35) |
| *toy* (**TO**) | 300 | 2 | 5 | (35,87,79,68,31) |

Tabla 13.1: Características de las BBDD utilizadas

### 13.1.2. Parámetros de los Experimentos

Una vez decididas las bases de datos con las que se va a tratar, es necesario especificar el resto de las variables necesarias para ejecutar un algoritmo. Estas variables se engloban en los parámetros generales del experimento y en las configuraciones de parámetros para los clasificadores, con los que realizaremos la fase de validación cruzada en cada experimento. Para ver en detalle la configuración de un experimento, se recomienda leer el apéndice A.6.

Los parámetros generales permanecerán invariantes a lo largo de toda la

fase de experimentación. Durante la fase de validación cruzada llevaremos a cabo un *5-fold*. Las métricas a optimizar serán *MAE* y *MZE*.

Cabe comentar que, en función de la métrica utilizada para guiar la fase de validación cruzada se obtendrán resultados diferentes, ya que se está optimizando el valor de la métrica que guía el proceso. Por tanto, se habrá de ejecutar cada experimento dos veces, una por cada métrica guía.

Como estamos testando el correcto funcionamiento del algoritmo de descomposición ordinal desarrollado, únicamente utilizaremos este como clasificador principal. Sin embargo, se probarán distintas configuraciones de clasificadores base (o internos), disponiendo cada uno de ellos de distintos hiperparámetros a optimizar. En concreto, los clasificadores base a probar, todos ellos incluidos en la biblioteca `scikit-learn`, y los hiperparámetros de los que haremos uso en cada caso son los siguientes:

1. **SVM** [27]: Codificación del algoritmo de máquinas vector soporte mediante *libsvm*. Implementa la clasificación multiclase mediante el esquema *OneVsOne*. Los distintos hiperparámetros (así como el conjunto de valores a probar) que tendremos en cuenta durante la validación cruzada son:
   - $C$: Coste del error. Se buscará el valor óptimo de entre el conjunto de valores $[10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 10^2, 10^3]$.
   - *gamma*: Coeficiente utilizado por algunos kernel del algoritmo. Los valores a explorar son similares a los del parámetro $C$.
2. **Árbol de Decisión** [28]: Clasificador basado en un modelo de árboles de decisión. Los parámetros a optimizar son:
   - *criterion*: Función que mide la calidad de la división de un nodo. Existen dos funciones para establecer el criterio de división: [*gini*, *entropy*].
   - *splitter*: Estrategia utilizada para realizar la división en cada nodo. Puede tomar los valores *best* y *random*.

- *min_samples_leaf*: Mínimo número de patrones necesarios para que un nodo sea hoja. Se probarán los valores $[1, 2, 3, 5, 7]$

3. **Regresión Logística** [29]: Clasificador basado en algortimos de regresión logística. Optimizaremos los valores de los siguiente hiperparámetros:

   - *penalty*: Norma utilizada para la penalización de los errores a la hora de construir el modelo. Probaremos con los valores *l1*, *l2*.
   - *tol*: Tolerancia del algoritmo para detener la generación del modelo. Optimizaremos el modelo escogiendo de entre los siguientes valores:$[10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}]$.
   - $C$: Inversa del valor de regularización. Se probará con los valores $[0.01, 0.1, 1, 10, 100]$.

Además de las variables especificadas, cada algoritmo cuenta con un mayor número parámetros, cuyos valores se especifican por defecto al momento de crear los objetos durante la ejecución. Para informarse en mayor detalle de cuales son estos parámetros no vistos en este documento, sus efectos y los valores que estos pueden tomar, visite la página de documentación del clasificador.

Para cada configuración distinta del meta-clasificador se estudiará, además, el efecto que tiene el tipo de descomposición realizada y el método de decisión seleccionado. Para ahorrar en número de ejecuciones, se lanzará un experimento diferente por tipo de descomposición, mientras que únicamente se obtendrá el valor del método de decisión con el que se ha generado el mejor modelo para una partición determinada durante la fase de validación cruzada.

Durante la fase de experimentación se llevarán a cabo 24 experimentos: 3 configuraciones de clasificadores base diferentes, 4 tipos de descomposición ordinal y 2 métricas con las que guiar la fase de validación cruzada. Todo ello utilizando la misma semilla a lo largo de todos los experimentos.

Para comprender mejor lo expuesto anteriormente, en la figura 13.1 se ilustra el fichero de configuración de uno de los experimentos realizados. El formato de estos ficheros se explica con mayor detenimiento en el apéndice A.6.

```
{


  "general_conf": {

    "basedir": "datasets/tfg/",
    "datasets": ["all"],
    "hyperparam_cv_nfolds": 5,
    "jobs": -1,
    "output_folder": "my_runs/",
    "metrics": ["mae", "mze"],
    "cv_metric": "mae"
  },


  "configurations": {

    "SVC": {

      "classifier": "OrdinalDecomposition",
      "parameters": {
        "dtype": ["oredered_partitions"],
        "decision_method": ["frank_hall", "exponential_loss",
                    "logarithmic_loss", "hinge_loss"],
        "base_classifier": "sklearn.svm.SVC",
        "parameters": {
          "C": [0.001, 0.01, 0.1, 1.0, 10, 100, 1000],
          "gamma": [0.001, 0.01, 0.1, 1.0, 10, 100, 1000],
          "probability": ["True"]
        }

      }
    }

  }


}
```

Figura 13.1: Ejemplo de un experimento

## 13.2. Resultados

La presente sección se dedicará a mostrar los resultados obtenidos de la ejecución del diseño experimental propuesto en la sección anterior.

Para comenzar, compararemos el rendimiento de cada una de las distintas configuraciones en función del tipo de descomposición escogido. Este rendimiento se medirá en base a dos métricas, *MAE* y *MZE*, cada una de las cuales habrá guiado un experimento donde todos los parámetros menos *cv_metric* son similares.

Las tablas que ilustran estos resultados se pueden observar en 13.2, 13.4 y 13.6, para los clasificadores `sklearn.svm.SVC`, `sklearn.tree.DecisionTreeClassifier` y `sklearn.linear_model.LogisticRegression`, respectivamente.

| **Bases de datos de regresión discretizadas** ($Media_{DesviacionTipica}$) | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | *ordered partitions* | | *one vs next* | | *one vs previous* | | *one vs followers* | |
| | MAE | MZE | MAE | MZE | MAE | MZE | MAE | MZE |
| **C5** | $\mathbf{0.48_{0.02}}$ | $\mathbf{0.42_{0.02}}$ | $0.74_{0.22}$ | $0.51_{0.06}$ | $0.61_{0.04}$ | $0.50_{0.03}$ | $\mathit{0.57_{0.06}}$ | $\mathit{0.45_{0.02}}$ |
| **M5** | $\mathbf{0.46_{0.07}}$ | $\mathbf{0.42_{0.05}}$ | $0.68_{0.12}$ | $0.52_{0.05}$ | $\mathit{0.52_{0.09}}$ | $\mathit{0.44_{0.05}}$ | $0.55_{0.10}$ | $0.46_{0.05}$ |
| **B5** | $\mathbf{0.44_{0.05}}$ | $\mathbf{0.42_{0.04}}$ | $1.23_{0.44}$ | $0.74_{0.10}$ | $\mathit{0.59_{0.12}}$ | $\mathit{0.54_{0.08}}$ | $0.65_{0.15}$ | $0.54_{0.08}$ |
| **H5** | $\mathbf{0.38_{0.03}}$ | $\mathbf{0.35_{0.02}}$ | $0.59_{0.06}$ | $0.42_{0.03}$ | $\mathit{0.44_{0.04}}$ | $\mathit{0.39_{0.03}}$ | $0.50_{0.05}$ | $0.41_{0.04}$ |
| **P5** | $\mathbf{0.61_{0.12}}$ | $\mathbf{0.54_{0.06}}$ | $1.09_{0.20}$ | $0.74_{0.09}$ | $\mathit{0.89_{0.22}}$ | $\mathit{0.61_{0.08}}$ | $1.00_{0.15}$ | $0.65_{0.04}$ |
| **Bases de datos de clasificación ordinal** ($Media_{DesviacionTipica}$) | | | | | | | | |
| **BO** | $\mathbf{0.62_{0.06}}$ | $\mathbf{0.42_{0.02}}$ | $\mathbf{0.62_{0.06}}$ | $\mathit{0.42_{0.03}}$ | $\mathbf{0.62_{0.06}}$ | $\mathit{0.42_{0.03}}$ | $\mathbf{0.62_{0.06}}$ | $\mathit{0.42_{0.03}}$ |
| **TA** | $\mathbf{0.46_{0.06}}$ | $\mathit{0.44_{0.06}}$ | $0.59_{0.12}$ | $0.46_{0.06}$ | $0.55_{0.13}$ | $\mathit{0.44_{0.06}}$ | $\mathit{0.54_{0.07}}$ | $\mathbf{0.43_{0.06}}$ |
| **SS** | $\mathbf{0.57_{0.03}}$ | $\mathbf{0.57_{0.03}}$ | $\mathbf{0.57_{0.03}}$ | $\mathbf{0.57_{0.03}}$ | $\mathbf{0.57_{0.03}}$ | $\mathbf{0.57_{0.03}}$ | $\mathbf{0.57_{0.03}}$ | $\mathbf{0.57_{0.03}}$ |
| **NT** | $\mathbf{0.04_{0.02}}$ | $\mathbf{0.04_{0.02}}$ | $\mathit{0.05_{0.02}}$ | $\mathit{0.05_{0.02}}$ | $\mathit{0.05_{0.02}}$ | $\mathit{0.05_{0.02}}$ | $\mathbf{0.04_{0.02}}$ | $\mathbf{0.04_{0.02}}$ |
| **TO** | $\mathbf{0.06_{0.02}}$ | $\mathbf{0.06_{0.02}}$ | $\mathit{0.07_{0.03}}$ | $\mathit{0.07_{0.03}}$ | $\mathit{0.07_{0.03}}$ | $\mathit{0.07_{0.03}}$ | $0.08_{0.03}$ | $0.08_{0.03}$ |

Tabla 13.2: Media y desviación típica de los resultados de Generalización con el algoritmo `sklearn.svm.SVC`

En las distintas tablas se muestran los resultados de generalización para las distintas configuraciones, mostrándose diferentes valores en función del clasificador interno y del tipo de decomposición utilizados sobre el problema.

Para ilustrar en qué manera influye el método de decisión sobre los resultados, podemos observar el número de particiones en las que el mejor modelo construido, para una base de datos y configuración determinadas, ha utilizado un método de decisión u otro. Este recuento se realizará tomando únicamente una de las dos métricas con las que hemos dirigido los experimentos, siendo en este caso *MAE* la métrica escogida para ilustrar los resultados obtenidos.

Las tablas que representan estos datos son 13.3, 13.5 y 13.7, emparejándose cada una con la tabla de resultados de generalización del correspondiente algoritmo. En ellas, las etiquetas situadas sobre cada columna se corresponden con abreviaciones de cada uno de los métodos de decisión (*fh* para *Frank y Hall*, *el* para *exponential loss*, etc.).

| **Bases de datos de regresión discretizadas** ($Media_{DesviacionTipica}$) | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *ordered partitions* | | | | *one vs next* | | | | *one vs previous* | | | | *one vs followers* | | | |
| | *fh* | *el* | *ll* | *hl* | *fh* | *el* | *ll* | *hl* | *fh* | *el* | *ll* | *hl* | *fh* | *el* | *ll* | *hl* |
| **C5** | *4* | **15** | 1 | 0 | - | *7* | **13** | 0 | - | **12** | *7* | 1 | - | **13** | *6* | 1 |
| **M5** | *5* | **15** | 0 | 0 | - | *2* | **18** | 0 | - | *9* | **10** | 1 | - | *8* | **12** | 0 |
| **B5** | *5* | **15** | 0 | 0 | - | **12** | *6* | 2 | - | **15** | *4* | 1 | - | **13** | *7* | 0 |
| **H5** | *6* | **15** | 0 | 0 | - | **7** | **7** | *6* | - | *6* | **14** | 0 | - | *3* | **17** | 0 |
| **P5** | **10** | **10** | 0 | 0 | - | **12** | *4* | *4* | - | **17** | *3* | 0 | - | **16** | *4* | 0 |
| **Bases de datos de clasificación ordinal** ($Media_{DesviacionTipica}$) | | | | | | | | | | | | | | | | |
| **BO** | **30** | 0 | 0 | 0 | - | **30** | 0 | 0 | - | **30** | 0 | 0 | - | **30** | 0 | 0 |
| **TA** | *1* | **29** | 0 | 0 | - | **27** | 0 | *3* | - | **29** | *1* | 0 | - | **29** | 0 | *1* |
| **SS** | 0 | **30** | 0 | 0 | - | **30** | 0 | 0 | - | **30** | 0 | 0 | - | **30** | 0 | 0 |
| **NT** | **27** | *3* | 0 | 0 | - | **22** | *6* | 2 | - | **25** | *5* | 0 | - | **19** | 3 | *8* |
| **TO** | **20** | *10* | 0 | 0 | - | *12* | **17** | 1 | - | *11* | **19** | 0 | - | *11* | **19** | 0 |

Tabla 13.3: Cantidad de mejores modelos construidos en función del método de decisión (`SVC`)

| **Bases de datos de regresión discretizadas** ($Media_{DesviacionTipica}$) | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | *ordered partitions* | | *one vs next* | | *one vs previous* | | *one vs followers* | |
| | MAE | MZE | MAE | MZE | MAE | MZE | MAE | MZE |
| **C5** | $\mathbf{0.59_{0.03}}$ | $\mathbf{0.48_{0.02}}$ | $0.78_{0.13}$ | $0.55_{0.05}$ | $0.64_{0.05}$ | $0.50_{0.02}$ | $\mathit{0.62_{0.05}}$ | $\mathit{0.50_{0.02}}$ |
| **M5** | $\mathbf{0.51_{0.07}}$ | $\mathbf{0.42_{0.05}}$ | $0.63_{0.09}$ | $0.48_{0.05}$ | $\mathit{0.53_{0.08}}$ | $\mathit{0.44_{0.05}}$ | $\mathit{0.53_{0.08}}$ | $\mathit{0.44_{0.05}}$ |
| **B5** | $\mathbf{0.52_{0.10}}$ | $\mathbf{0.46_{0.06}}$ | $0.96_{0.21}$ | $0.64_{0.08}$ | $0.59_{0.11}$ | $0.50_{0.05}$ | $\mathit{0.56_{0.12}}$ | $\mathit{0.49_{0.06}}$ |
| **H5** | $\mathbf{0.48_{0.04}}$ | $\mathbf{0.40_{0.03}}$ | $0.70_{0.13}$ | $0.49_{0.07}$ | $\mathit{0.51_{0.05}}$ | $\mathit{0.42_{0.02}}$ | $0.54_{0.05}$ | $0.44_{0.02}$ |
| **P5** | $\mathbf{0.81_{0.16}}$ | $\mathit{0.57_{0.08}}$ | $1.10_{0.21}$ | $0.67_{0.12}$ | $0.82_{0.18}$ | $0.58_{0.08}$ | $\mathit{0.82_{0.14}}$ | $\mathbf{0.57_{0.06}}$ |
| **Bases de datos de clasificación ordinal** ($Media_{DesviacionTipica}$) | | | | | | | | |
| **BO** | $\mathbf{0.60_{0.15}}$ | $\mathit{0.49_{0.12}}$ | $0.66_{0.13}$ | $0.52_{0.10}$ | $0.66_{0.10}$ | $\mathbf{0.48_{0.08}}$ | $\mathit{0.64_{0.16}}$ | $0.54_{0.12}$ |
| **TA** | $\mathit{0.51_{0.11}}$ | $\mathit{0.41_{0.06}}$ | $0.70_{0.11}$ | $0.51_{0.07}$ | $0.52_{0.10}$ | $\mathbf{0.40_{0.06}}$ | $\mathbf{0.50_{0.10}}$ | $\mathit{0.41_{0.06}}$ |
| **SS** | $0.41_{0.14}$ | $\mathit{0.36_{0.12}}$ | $\mathit{0.40_{0.14}}$ | $0.37_{0.11}$ | $\mathbf{0.40_{0.13}}$ | $\mathbf{0.35_{0.11}}$ | $0.44_{0.13}$ | $0.42_{0.11}$ |
| **NT** | $\mathit{0.07_{0.04}}$ | $\mathit{0.07_{0.04}}$ | $\mathit{0.07_{0.04}}$ | $\mathit{0.07_{0.04}}$ | $\mathbf{0.06_{0.03}}$ | $\mathbf{0.06_{0.03}}$ | $0.08_{0.03}$ | $0.08_{0.03}$ |
| **TO** | $\mathbf{0.12_{0.04}}$ | $\mathit{0.12_{0.03}}$ | $0.13_{0.06}$ | $\mathbf{0.11_{0.04}}$ | $\mathbf{0.12_{0.04}}$ | $\mathit{0.12_{0.03}}$ | $\mathit{0.12_{0.05}}$ | $\mathbf{0.11_{0.04}}$ |

Tabla 13.4: Media y desviación típica de los resultados de Generalización con el algoritmo `sklearn.tree.DecisionTreeClassifier`

| **Bases de datos de regresión discretizadas** ($Media_{DesviacionTipica}$) | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *ordered partitions* | | | | *one vs next* | | | | *one vs previous* | | | | *one vs followers* | | | |
| | *fh* | *el* | *ll* | *hl* | *fh* | *el* | *ll* | *hl* | *fh* | *el* | *ll* | *hl* | *fh* | *el* | *ll* | *hl* |
| **C5** | **12** | *8* | 0 | 0 | - | **13** | 3 | *4* | - | **15** | 0 | *5* | - | **16** | 0 | *4* |
| **M5** | *4* | **16** | 0 | 0 | - | **11** | *7* | 2 | - | **10** | 0 | *1* | - | **16** | *2* | *2* |
| **B5** | *9* | **11** | 0 | 0 | - | **16** | *2* | *2* | - | **18** | *2* | 0 | - | **16** | 0 | *4* |
| **H5** | *4* | **14** | 2 | 0 | - | **10** | *7* | 3 | - | **19** | *1* | 0 | - | **18** | *1* | *1* |
| **P5** | **10** | **10** | 0 | 0 | - | **11** | 2 | *7* | - | **15** | 2 | *3* | - | **15** | 1 | *4* |
| **Bases de datos de clasificación ordinal** ($Media_{DesviacionTipica}$) | | | | | | | | | | | | | | | | |
| **BO** | *14* | **16** | 0 | 0 | - | **17** | 4 | *9* | - | **21** | 1 | *8* | - | **30** | 0 | 0 |
| **TA** | **27** | *3* | 0 | 0 | - | **18** | 2 | *10* | - | **29** | *1* | 0 | - | **18** | 0 | *12* |
| **SS** | **28** | *2* | 0 | 0 | - | **27** | 0 | *3* | - | **23** | 0 | *7* | - | **28** | 0 | *2* |
| **NT** | **29** | *1* | 0 | 0 | - | **29** | 0 | *1* | - | **25** | 0 | *5* | - | **30** | 0 | 0 |
| **TO** | **24** | *6* | 0 | 0 | - | **30** | 0 | 0 | - | **20** | 0 | *10* | - | **24** | 0 | *6* |

Tabla 13.5: Cantidad de mejores modelos construidos en función del método de decisión (`DecisionTree`)

| **Bases de datos de regresión discretizadas** ($Media_{DesviacionTipica}$) | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | *ordered partitions* | | *one vs next* | | *one vs previous* | | *one vs followers* | |
| | MAE | MZE | MAE | MZE | MAE | MZE | MAE | MZE |
| **C5** | $\mathbf{0.47_{0.01}}$ | $\mathbf{0.39_{0.01}}$ | $0.59_{0.07}$ | $0.47_{0.04}$ | $\mathit{0.50_{0.02}}$ | $\mathit{0.43_{0.02}}$ | $0.51_{0.04}$ | $0.43_{0.02}$ |
| **M5** | $\mathbf{0.45_{0.08}}$ | $\mathbf{0.40_{0.06}}$ | $0.54_{0.08}$ | $0.45_{0.08}$ | $\mathit{0.48_{0.06}}$ | $\mathit{0.42_{0.05}}$ | $0.55_{0.08}$ | $0.46_{0.06}$ |
| **B5** | $\mathbf{0.34_{0.03}}$ | $\mathbf{0.33_{0.02}}$ | $0.45_{0.10}$ | $0.40_{0.06}$ | $0.40_{0.05}$ | $0.39_{0.04}$ | $\mathit{0.38_{0.86}}$ | $\mathit{0.35_{0.03}}$ |
| **H5** | $\mathbf{0.38_{0.03}}$ | $\mathbf{0.34_{0.02}}$ | $0.57_{0.05}$ | $0.42_{0.03}$ | $\mathit{0.44_{0.03}}$ | $\mathit{0.37_{0.02}}$ | $0.46_{0.04}$ | $0.38_{0.03}$ |
| **P5** | $\mathbf{0.65_{0.12}}$ | $\mathbf{0.49_{0.06}}$ | $1.01_{0.18}$ | $0.63_{0.09}$ | $0.73_{0.13}$ | $0.54_{0.06}$ | $\mathit{0.70_{0.15}}$ | $\mathit{0.52_{0.08}}$ |
| **Bases de datos de clasificación ordinal** ($Media_{DesviacionTipica}$) | | | | | | | | |
| **BO** | $\mathit{0.64_{0.14}}$ | $\mathit{0.48_{0.08}}$ | $\mathbf{0.64_{0.11}}$ | $\mathit{0.48_{0.08}}$ | $\mathbf{0.64_{0.11}}$ | $\mathit{0.48_{0.08}}$ | $\mathit{0.64_{0.14}}$ | $\mathbf{0.47_{0.09}}$ |
| **TA** | $\mathbf{0.60_{0.09}}$ | $\mathbf{0.46_{0.07}}$ | $0.64_{0.08}$ | $0.48_{0.05}$ | $\mathit{0.62_{0.09}}$ | $\mathit{0.47_{0.05}}$ | $\mathit{0.62_{0.09}}$ | $\mathbf{0.46_{0.07}}$ |
| **SS** | $0.37_{0.14}$ | $\mathbf{0.35_{0.11}}$ | $0.39_{0.15}$ | $\mathit{0.37_{0.11}}$ | $\mathbf{0.37_{0.13}}$ | $\mathit{0.35_{0.12}}$ | $0.37_{0.16}$ | $0.35_{0.13}$ |
| **NT** | $\mathbf{0.02_{0.02}}$ | $\mathbf{0.02_{0.02}}$ | $\mathit{0.03_{0.02}}$ | $\mathbf{0.02_{0.02}}$ | $\mathit{0.03_{0.02}}$ | $\mathit{0.03_{0.02}}$ | $\mathbf{0.02_{0.02}}$ | $\mathbf{0.02_{0.02}}$ |
| **TO** | $\mathbf{0.90_{0.05}}$ | $0.70_{0.03}$ | $\mathit{1.12_{0.04}}$ | $\mathit{0.70_{0.01}}$ | $1.13_{0.01}$ | $\mathbf{0.70_{0.00}}$ | $1.15_{0.04}$ | $0.76_{0.02}$ |

Tabla 13.6: Media y desviación típica de los resultados de Generalización con el algoritmo `sklearn.linear_model.LogisticRegression`

| **Bases de datos de regresión discretizadas** ($Media_{DesviacionTipica}$) | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *ordered partitions* | | | | *one vs next* | | | | *one vs previous* | | | | *one vs followers* | | | |
| | *fh* | *el* | *ll* | *hl* | *fh* | *el* | *ll* | *hl* | *fh* | *el* | *ll* | *hl* | *fh* | *el* | *ll* | *hl* |
| **C5** | *7* | **13** | 0 | 0 | - | **12** | *8* | 0 | - | **13** | *7* | 0 | - | **16** | *3* | 1 |
| **M5** | *9* | **11** | 0 | 0 | - | *9* | **11** | 0 | - | **11** | *9* | 0 | - | *4* | **16** | 0 |
| **B5** | *5* | **15** | 0 | 0 | - | **17** | 1 | *2* | - | **16** | 1 | *3* | - | **14** | 2 | *4* |
| **H5** | *3* | **17** | 0 | 0 | - | *3* | **17** | 0 | - | *5* | **15** | 0 | - | *4* | **16** | 0 |
| **P5** | *4* | **16** | 0 | 0 | - | **13** | 3 | *4* | - | **12** | *8* | 0 | - | **13** | 1 | *6* |
| **Bases de datos de clasificación ordinal** ($Media_{DesviacionTipica}$) | | | | | | | | | | | | | | | | |
| **BO** | **17** | *13* | 0 | 0 | - | **24** | 0 | *6* | - | **23** | 1 | *6* | - | **19** | 2 | *9* |
| **TA** | *12* | **18** | 0 | 0 | - | *13* | 0 | **17** | - | **18** | *6* | *6* | - | **13** | *4* | **13** |
| **SS** | **20** | *10* | 0 | 0 | - | **18** | 3 | *9* | - | **23** | 0 | *7* | - | **22** | *4* | *4* |
| **NT** | **30** | 0 | 0 | 0 | - | **21** | *3* | *6* | - | **24** | 2 | *4* | - | **25** | 1 | *4* |
| **TO** | *4* | **26** | 0 | 0 | - | **27** | 0 | *3* | - | **30** | 0 | 0 | - | **14** | 6 | *10* |

Tabla 13.7: Cantidad de mejores modelos construidos en función del método de decisión (`LogisticRegression`)

A lo largo de los resultados mostrados en esta sección, es posible apreciar que el tipo de descomposición *Ordered Partitions* consigue resultados iguales o mejores que el resto de manera consistente. Aunque podemos observar que este hecho se manifiesta de forma especialmente clara en las bases de datos de regresión discretizadas.

Con el objetivo de comparar con mayor facilidad la eficiencia de las distintas configuraciones, vamos a mostrar dos tablas alternativas, 13.8 y 13.9, una por métrica considerada durante los experimentos, donde únicamente se mostrarán los mejores resultados encontrados para una combinación de clasificador y conjunto de datos. En estas dos tablas, los mejores resultados obtenidos para una base de datos concreta se mostrarán en negrita.

| **Bases de datos de regresión discretizadas** ($Media_{DesviacionTipica}$) | | | |
|---|---|---|---|
| | *SVC* | *DecisionTreeClassifier* | *LogisticRegression* |
| **C5** | $\mathit{0.48}_{\mathit{0.02}}$ | $0.59_{0.03}$ | $\mathbf{0.47_{0.01}}$ |
| **M5** | $\mathit{0.46}_{\mathit{0.07}}$ | $0.51_{0.07}$ | $\mathbf{0.45_{0.08}}$ |
| **B5** | $\mathit{0.44}_{\mathit{0.05}}$ | $0.52_{0.10}$ | $\mathbf{0.34_{0.03}}$ |
| **H5** | $\mathbf{0.38_{0.03}}$ | $\mathit{0.48}_{\mathit{0.04}}$ | $\mathbf{0.38_{0.03}}$ |
| **P5** | $\mathbf{0.61_{0.12}}$ | $0.81_{0.16}$ | $\mathit{0.65}_{\mathit{0.12}}$ |
| **Bases de datos de clasificación ordinal** ($Media_{DesviacionTipica}$) | | | |
| **BO** | $\mathit{0.62}_{\mathit{0.06}}$ | $\mathbf{0.60_{0.15}}$ | $0.64_{0.11}$ |
| **TA** | $\mathbf{0.46_{0.06}}$ | $\mathit{0.50}_{\mathit{0.10}}$ | $0.60_{0.09}$ |
| **SS** | $0.57_{0.03}$ | $\mathit{0.40}_{\mathit{0.13}}$ | $\mathbf{0.37_{0.13}}$ |
| **NT** | $\mathit{0.04}_{\mathit{0.02}}$ | $0.06_{0.03}$ | $\mathbf{0.02_{0.02}}$ |
| **TO** | $\mathbf{0.06_{0.02}}$ | $\mathit{0.12}_{\mathit{0.04}}$ | $0.90_{0.05}$ |

Tabla 13.8: Media y desviación típica *MAE* de los mejores resultados de Generalización obtenidos

| **Bases de datos de regresión discretizadas** ($Media_{DesviacionTipica}$) | | | |
|---|---|---|---|
| | *SVC* | *DecisionTreeClassifier* | *LogisticRegression* |
| **C5** | $\mathit{0.42}_{\mathit{0.02}}$ | $0.48_{0.02}$ | $\mathbf{0.39_{0.01}}$ |
| **M5** | $\mathit{0.42}_{\mathit{0.05}}$ | $\mathit{0.42}_{\mathit{0.05}}$ | $\mathbf{0.40_{0.06}}$ |
| **B5** | $\mathit{0.42}_{\mathit{0.04}}$ | $0.46_{0.06}$ | $\mathbf{0.33_{0.02}}$ |
| **H5** | $\mathit{0.35}_{\mathit{0.02}}$ | $0.40_{0.03}$ | $\mathbf{0.34_{0.02}}$ |
| **P5** | $\mathit{0.54}_{\mathit{0.06}}$ | $0.57_{0.06}$ | $\mathbf{0.49_{0.06}}$ |
| **Bases de datos de clasificación ordinal** ($Media_{DesviacionTipica}$) | | | |
| **BO** | $\mathbf{0.42_{0.02}}$ | $0.48_{0.08}$ | $\mathit{0.47}_{\mathit{0.09}}$ |
| **TA** | $\mathit{0.43}_{\mathit{0.06}}$ | $\mathbf{0.40_{0.06}}$ | $0.46_{0.07}$ |
| **SS** | $\mathit{0.57}_{\mathit{0.03}}$ | $\mathbf{0.35_{0.11}}$ | $\mathbf{0.35_{0.11}}$ |
| **NT** | $\mathit{0.04}_{\mathit{0.02}}$ | $0.06_{0.03}$ | $\mathbf{0.02_{0.02}}$ |
| **TO** | $\mathbf{0.06_{0.02}}$ | $\mathit{0.11}_{\mathit{0.04}}$ | $0.70_{0.00}$ |

Tabla 13.9: Media y desviación típica *MZE* de los mejores resultados de Generalización obtenidos

### 13.2.1. Comparación con otros algoritmos

Para corroborar que el clasificador desarrollado obtiene resultados conforme a lo esperado, vamos a comparar el rendimiento obtenido por el mismo, esto es, los mejores resultados conseguidos para cada conjunto de datos a lo largo de todos los experimentos, frente a los resultados devueltos por varios algoritmos de regresión ordinal de contrastada eficiencia, por supuesto que sobre los mismos conjuntos de datos. El estudio en el que se detalla la experimentación con los algoritmos que no han sido expuestos en el presente trabajo pertenece al artículo de investigación [10] citado con anterioridad.

De cara a comprobar la eficiencia de nuestro algoritmo, tomaremos un subconjunto de los clasificadores utilizados en el mencionado artículo, puesto que la extensión del mismo escapa al propósito de este Trabajo. La comparación entre clasificadores se ha realizado de forma independiente para una y otra métrica, dando como resultado las tablas 13.10 y 13.11. Se puede observar como, pese a suponer aproximaciones relativamente senciallas frente a otros algoritmos más complejos expuestos en [10], los resultados obtenidos son altamente competitivos, obteniendo en varios casos mejores resultados que todos los algoritmos contemplados en el estado del arte.

| **Bases de datos de regresión discretizadas** ($Media_{DesviacionTipica}$) | | | | | | | |
|---|---|---|---|---|---|---|---|
| | *OrdDec* | *SVMOP* | *ELMOP* | *POM* | *SVOREX* | *SVORIM* | *GPOR* |
| **C5** | $0.47_{0.01}$ | $0.50_{0.03}$ | $0.55_{0.04}$ | $\mathbf{0.43_{0.02}}$ | $0.45_{0.05}$ | $\mathit{0.45_{0.04}}$ | $\mathbf{0.43_{0.02}}$ |
| **M5** | $0.45_{0.08}$ | $0.46_{0.08}$ | $0.45_{0.09}$ | $\mathbf{0.43_{0.08}}$ | $0.47_{0.07}$ | $0.44_{0.10}$ | $\mathit{0.44_{0.07}}$ |
| **B5** | $0.34_{0.03}$ | $0.47_{0.04}$ | $0.69_{0.10}$ | $\mathit{0.28_{0.03}}$ | $0.37_{0.02}$ | $\mathit{0.28_{0.03}}$ | $\mathbf{0.27_{0.02}}$ |
| **H5** | $0.38_{0.03}$ | $0.40_{0.04}$ | $0.42_{0.04}$ | $0.40_{0.02}$ | $\mathit{0.36_{0.03}}$ | $\mathit{0.36_{0.03}}$ | $\mathbf{0.34_{0.04}}$ |
| **P5** | $\mathbf{0.61_{0.12}}$ | $0.69_{0.16}$ | $1.12_{0.26}$ | $0.70_{0.20}$ | $0.66_{0.12}$ | $\mathit{0.63_{0.11}}$ | $0.65_{0.13}$ |
| **Bases de datos de clasificación ordinal** ($Media_{DesviacionTipica}$) | | | | | | | |
| **BO** | $\mathit{0.60_{0.15}}$ | $\mathbf{0.59_{0.12}}$ | $0.65_{0.17}$ | $0.95_{0.32}$ | $0.62_{0.09}$ | $0.61_{0.09}$ | $0.62_{0.06}$ |
| **TA** | $\mathbf{0.46_{0.06}}$ | $0.50_{0.08}$ | $0.62_{0.12}$ | $0.63_{0.12}$ | $\mathit{0.47_{0.06}}$ | $0.47_{0.07}$ | $0.86_{0.16}$ |
| **SS** | $\mathbf{0.37_{0.13}}$ | $0.41_{0.12}$ | $0.48_{0.12}$ | $0.81_{0.23}$ | $\mathit{0.37_{0.15}}$ | $0.38_{0.13}$ | $0.63_{0.15}$ |
| **NT** | $\mathbf{0.02_{0.02}}$ | $0.04_{0.03}$ | $0.05_{0.02}$ | $\mathit{0.03_{0.02}}$ | $\mathit{0.03_{0.02}}$ | $\mathit{0.03_{0.02}}$ | $\mathbf{0.02_{0.02}}$ |
| **TO** | $0.06_{0.02}$ | $0.07_{0.03}$ | $0.08_{0.03}$ | $0.98_{0.04}$ | $\mathbf{0.02_{0.01}}$ | $\mathbf{0.02_{0.01}}$ | $\mathit{0.05_{0.02}}$ |

Tabla 13.10: Comparación de la media y desviación tipica de *OrderedPartitions* frente a algoritmos regresión ordinal (*MAE*)

| **Bases de datos de regresión discretizadas** ($Media_{DesviacionTipica}$) | | | | | | | |
|---|---|---|---|---|---|---|---|
| | *OrdDec* | *SVMOP* | *ELMOP* | *POM* | *SVOREX* | *SVORIM* | *GPOR* |
| **C5** | $0.39_{0.01}$ | $0.41_{0.08}$ | $0.46_{0.03}$ | $\mathit{0.38_{0.01}}$ | $0.39_{0.02}$ | $0.39_{0.03}$ | $\mathbf{0.37_{0.01}}$ |
| **M5** | $0.40_{0.06}$ | $0.41_{0.06}$ | $0.40_{0.07}$ | $\mathbf{0.39_{0.06}}$ | $0.43_{0.05}$ | $0.41_{0.06}$ | $\mathit{0.40_{0.05}}$ |
| **B5** | $0.33_{0.02}$ | $0.43_{0.03}$ | $0.54_{0.04}$ | $\mathit{0.27_{0.02}}$ | $\mathit{0.27_{0.02}}$ | $\mathit{0.27_{0.02}}$ | $\mathbf{0.26_{0.01}}$ |
| **H5** | $0.34_{0.02}$ | $0.35_{0.03}$ | $0.36_{0.02}$ | $0.35_{0.01}$ | $\mathit{0.32_{0.02}}$ | $\mathit{0.32_{0.02}}$ | $\mathbf{0.31_{0.02}}$ |
| **P5** | $\mathit{0.49_{0.06}}$ | $0.52_{0.08}$ | $0.65_{0.08}$ | $\mathbf{0.48_{0.11}}$ | $0.50_{0.09}$ | $0.49_{0.09}$ | $0.51_{0.09}$ |
| **Bases de datos de clasificación ordinal** ($Media_{DesviacionTipica}$) | | | | | | | |
| **BO** | $\mathbf{0.42_{0.02}}$ | $\mathit{0.44_{0.09}}$ | $0.56_{0.13}$ | $0.65_{0.16}$ | $0.45_{0.06}$ | $0.45_{0.07}$ | $0.47_{0.09}$ |
| **TA** | $\mathbf{0.40_{0.06}}$ | $0.45_{0.05}$ | $0.43_{0.07}$ | $0.49_{0.07}$ | $\mathit{0.41_{0.07}}$ | $\mathbf{0.40_{0.06}}$ | $0.43_{0.05}$ |
| **SS** | $\mathbf{0.35_{0.11}}$ | $0.40_{0.09}$ | $0.44_{0.16}$ | $0.61_{0.15}$ | $\mathit{0.37_{0.12}}$ | $\mathit{0.37_{0.12}}$ | $0.39_{0.12}$ |
| **NT** | $\mathbf{0.02_{0.02}}$ | $0.04_{0.02}$ | $0.57_{0.02}$ | $\mathbf{0.02_{0.02}}$ | $\mathit{0.03_{0.02}}$ | $\mathit{0.03_{0.02}}$ | $\mathbf{0.02_{0.02}}$ |
| **TO** | $\mathit{0.06_{0.02}}$ | $0.07_{0.02}$ | $0.07_{0.02}$ | $0.71_{0.02}$ | $\mathbf{0.02_{0.01}}$ | $\mathbf{0.02_{0.01}}$ | $0.10_{0.03}$ |

Tabla 13.11: Comparación de la media y desviación tipica de *OrderedPartitions* frente a algoritmos regresión ordinal (*MZE*)

# Parte V

# Conclusiones

# Capítulo 14

# Conclusiones del Autor

Llegados a este punto, tras la superación de las pertinentes pruebas de integración y aceptación y, por ende, el final del desarrollo de la aplicación que se buscaba construir en este Trabajo, procedemos a discutir el grado de efectividad con que se han implantado los distintos requisitos especificados durante las fases preliminares del mismo.

## 14.1. Objetivos de Formación Alcanzados

En esta sección expondremos los conocimientos adquiridos por el alumno gracias al desarrollo del presente Trabajo:

- Se ha obtenido un mayor grado de destreza en la utilización de lenguajes incluidos en el paradigma de la orientación a objetos, específicamente con Python, con el cual se ha desarrollado la aplicación. Además, se ha profundizado en el manejo de diversas estructuras de datos inherentes al lenguaje.
- Se ha mejorado en el manejo de bibliotecas científicas propias del lenguaje Python para la carga y tratamiento de grandes cantidades de

datos como son `pandas` y `numpy`, y para la realización de tareas de minería de datos y de aprendizaje de máquinas: `scikit-learn`.

- Se han adquirido conocimientos relativos a la regresión ordinal y más concretamente al método de clasificación ordinal basado en la descomposición de un problema ordinal en varios subproblemas nominales relacionados.

- Se han conseguido habilidades prácticas sobre el el desarrollo de una aplicación, obtenida gracias a la documentación del Trabajo, donde se incluye el manual de usuario.

- Además, se ha aprendido a manejar la herramienta de composición de textos indispensable para la construcción del presente documento LaTeX, así como herramientas auxiliares para la creación y modificación de diagramas e imágenes como son *Dia* y *Gimp*, respectivamente.

## 14.2. Objetivos Operacionales Alcanzados

Los objetivos operacionales que ha sido posible alcanzar durante el desarrollo del sistema son los siguientes:

- Se ha elaborado una aplicación que automatiza la experimentación con algoritmos de clasificación, particularmente con los de regresión ordinal, siempre que estos se incluyan en el paquete `scikit-learn` o se hayan implementado internamente.

- Se ha desarrollado una sintaxis de ficheros de configuración con la que personalizar completamente cada experimento mediante formato JSON.

- El guardado de los resultados se realiza con el nivel de granularidad máximo posible, de forma tal que no suponga un cuello de botella

para posteriores implantaciones de paralelización en la ejecución del *framework*.

- Implementación de un algoritmo de clasificación ordinal basado en el método de descomposición ordinal, visto en la primera parte de este documento.
- Se ha llevado a cabo un estudio sobre la validez del algoritmo desarrollado sobre una serie de conjuntos de datos de prueba, donde se han comparado sus resultados con los de otros clasificadores que sirven al mismo propósito.
- Se han implementado diversos casos de prueba para comprobar el correcto funcionamiento de cada módulo que compone al sistema, así como para corroborar el funcionamiento al completo de la aplicación.

Dados por conseguidos los objetivos anteriormente enumerados, suponemos cumplidos los requisitos ligados a la realización de este Trabajo.

# Capítulo 15

# Futuras Mejoras

Dada la naturaleza de los Trabajos de Fin de Grado, la envergadura del mismo debe encontrarse limitada a un tamaño razonable, a efectos de cumplir su función para con el estudiante. Por ello, hay aspectos del sistema que se han dejado fuera del Trabajo, que pueden ser abarcados en futuras mejoras o iteraciones en el desarrollo de la aplicación, aumentando estas mejoras la funcionalidad y rendimiento del programa. En este capítulo comentaremos varios aspectos que, a nuestro juicio, pueden mejorar la utilidad del *software*.

## 15.1. Algoritmos de Clasificación Ordinal

A pesar de disponer de los diversos métodos de clasificación disponibles en la biblioteca `scikit-learn`, el *framework* únicamente dispone de un algoritmo de clasificación ordinal, desarrollado en este Trabajo. Por ello, creemos necesaria la inclusión de nuevos clasificadores, en particular aquellos existentes en el seno del *framework* predecesor de este, ORCA, para que la aplicación desarrollada cumpla con su propósito de forma más efectiva.

## 15.2. Mejorar la Capacidad de Procesamiento Paralelo

En función de las configuraciones utilizadas dentro de un experimento y/o del tamaño de las bases de datos sobre las que se va a realizar el proceso de clasificación, el tiempo computacional necesario para llevar a cabo el experimento puede ser muy elevado. Para disminuir el tiempo de ejecución necesario, sería deseable dividir el flujo de ejecución del programa en múltiples unidades de ejecución independientes que puedan ser ejecutadas en paralelo en sistemas que soporten la paralelización por *hardware*, como un clúster de ordenadores.

Cabe mencionar que, actualmente, el proceso de validación cruzada realizado a nivel de partición soporta la creación de hilos para su ejecución concurrente, gracias al objeto `GridSearchCV`, incluido en la biblioteca `scikit-learn`, encargado de este cometido.

## 15.3. Preprocesamiento

Para la utilización del programa, es necesario que las bases de datos se encuentren en un formato concreto, de lo contrario la ejecución no se realizará de forma satisfactoria.

Sin embargo, el *framework* no dispone de herramientas que faciliten al usuario la tarea de formatear las BBDD para que se adecuen al estilo deseado. La inclusión de métodos para realizar estas funciones de manera trivial se antoja deseable.

`scikit-learn` dispone de una amplia cartera de métodos de minería de datos de los que puede disponer la aplicación y que ayudarían en la realización de estas tareas, así como de otras que se crean deseables, sin necesidad de añadir nuevas dependencias *software*. Adicionalmente, se propone mejorar la

documentación para guiar al usuario en la preparación de los datos, una vez añadido este módulo.

# Bibliografía


[1] Tom Mitchell. *Machine Learning*. McGraw Hill, 1997.

[2] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA data mining software: an update. *SIGKDD Explorations*, 11(1):10–18, 2009.

[3] Departamento de Informática y Análisis Numérico de la Universidad de Córdoba. Aprendizaje y redes neuronales artificiales. `https://www.uco.es/grupos/ayrna/index.php/es`. [Online; Visitado 24 octubre 2018].

[4] J. Sánchez-Monedero, P. A. Gutiérrez, and M. Pérez-Ortiz. ORCA: A Matlab/Octave Toolbox for Ordinal Regression. *Journal of Machine Learning Research*, (125):1–5, 2019.

[5] MATLAB. *version 9.0 (R2016a)*. The MathWorks Inc., Natick, Massachusetts, 2016.

[6] John W. Eaton. *GNU Octave Manual*. Network Theory Limited, 2002.

[7] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825 –2830, 2011.

[8] Stéfan van der Walt and Jarrod Millman, editors. *Data Structures for Statistical Computing in Python*, 2010.

[9] Oliphant Travis E. *A Guide to NumPy*. USA: Trelgol Publishing, 2006.

[10] P.A. Gutiérrez, M. Pérez-Ortiz, J. Sánchez-Monedero, F. Fernandez-Navarro, and C. Hervás-Martínez. Ordinal regression methods: survey and experimental study. *IEEE Transactions on Knowledge and Data Engineering*, 28(1), 2016.

[11] Y. Shafranovich. Common format and mime type for comma-separated values (csv) files. RFC 4180, RFC Editor, 10 2005.

[12] Github: the world's leading software development platform. `https://github.com/`. Online; Visitado 10 Enero 2019.

[13] Git: –everything-is-local. `https://git-scm.com/`. Online; Visitado 1 Octubre 2019.

[14] PEP 8 – Style Guide for Python Code. `https://www.python.org/dev/peps/pep-0008`. Online; Visitado 10 enero 2019.

[15] Katy Huff, David Lippa, Dillon Niederhut, and M Pacer, editors. *The Sacred Infrastructure for Computational Research*, 2017.

[16] E. L. Allwein, R. E. Schapire, , Y. Singer, and J. Mach. Reducing multiclass to binary: A unifying approach for margin classifiers. *Journal of Machine Learning Research*, 1:113–141, 2001.

[17] Sébastien Destercke y Gen Yang. Cautious ordinal classification by binary decomposition. machine learning and knowledge discovery in databases. *European Conference ECML/PKDD*, pages 323–327, 9 2014.

[18] M Cruz-Ramírez, Cesar Martínez, Javier Sánchez-Monedero, and Pedro Antonio Gutiérrez. Metrics to guide a multi-objective evolutionary algorithm for ordinal classification. *Neurocomputing*, 135:21–31, 07 2014.

[19] Alistair Cockburn. *Writing Effective Use Cases*. Addison-Wesley, 2001.

[20] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. [Online; Visitado 14 octubre 2018].

[21] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007.

[22] BaseEstimator: Base class for all estimators in Scikit-Learn. `https://scikit-learn.org/stable/modules/generated/sklearn.base.BaseEstimator.html`. Online; Visitado 7 mayo 2019.

[23] ClassifierMixin: Mixin class for all classifiers in Scikit-Learn. `https://scikit-learn.org/stable/modules/generated/sklearn.base.ClassifierMixin.html`. Online; Visitado 7 mayo 2019.

[24] Wei Chu and Zoubin Ghahramani. Gaussian processes for ordinal regression. *J. Mach. Learn. Res.*, 6:1019–1041, December 2005.

[25] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. Online; Visitado 23 Abril 2019.

[26] Machine learning data repository, 2018. Online; Visitado 23 Abril 2019.

[27] SVC: C-Support Vector Classification. `https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html`. Online; Visitado 1 Junio 2019.

[28] DecisionTreeClassifier: A decision tree classifier. `https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html`. Online; Visitado 1 Junio 2019.

[29] LogisticRegression: Logistic Regression (aka logit, MaxEnt) classifier. `https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html`. Online; Visitado 1 Junio 2019.

# Parte VI

# Apéndices

# Apéndice A

# Manual de Usuario

Este documento constituye el manual de usuario del programa desarrollado en Python que forma parte del Trabajo Fin de Grado: *Framework en Python para problemas de clasificación ordinal.*

## A.1. Descripción del Producto

El *software* generado durante el transcurso del Trabajo tiene como misión facilitar al usuario la experimentación con algoritmos de clasificación ordinal, aunque admite la utilización de cualquier clasificador siempre que este sea compatible con `scikit-learn`. El *framework* dispondrá de aquellos clasificadores que hayan sido desarrollados por sus potenciales usuarios, además de la mayoría de los algoritmos de clasificación nominal incluidos dentro de la biblioteca previamente nombrada.

Cabe recordar que, para que un algoritmo desarrollado por un usuario sea utilizable por el sistema, este ha de ser compatible con el paquete `sklearn` de Python, ya que el objeto encargado de realizar la validación cruzada, fase fundamental de la experimentación, únicamente es capaz de utilizar clasificadores compatibles con esta biblioteca.

Podemos resumir el funcionamiento de la aplicación de la siguiente manera: el módulo central recibe los detalles del experimento a realizar por medio de un fichero de configuración, que ha de contener, como mínimo, la configuración de los algoritmos y los conjuntos de datos con los que se desea trabajar. El programa entonces lleva a cabo el proceso de generación de un modelo óptimo para cada clasificador y base de datos distintos. Una vez concluida la ejecución de forma correcta, se encontrará almacenada toda la información de interés en un directorio de forma estructurada y fácil de comprender.

El sistema dispone de las siguientes características:

- Desarrollado con el lenguaje de programación interpretado Python, hogar de varias bibliotecas que facilitan las tareas de la Ciencia Computacional, entre ellas las ramas de la minería de datos y el aprendizaje automático.
- Opciones de configuración para controlar la totalidad de las variables de un experimento, pudiendo ser estos reproducibles.
- Almacenamiento de la información generada durante la ejecución de un experimento, incluyendo instancias de clasificadores, predicciones de etiquetas de clase para los patrones de una base de datos y valores para métricas de rendimiento de los clasificadores.
- Generación de un resumen del comportamiento de los distintos clasificadores, esto es, de su rendimiento en base a unas métricas especificadas, con respecto a cada conjunto de datos.

## A.2. Requisitos del Sistema

Gracias a la naturaleza multiplataforma de Python, la aplicación puede ser utilizada en cualquier plataforma que disponga de una versión compatible

del intérprete de este lenguaje, así como de las dependencias de paquetes que requiere para su funcionamiento.

Los requisitos *hardware* de la aplicación estarán íntimamente ligados al tamaño de las bases de datos, a la complejidad computacional de cada clasificador y a la cantidad de valores diferentes para los parámetros utilizados para optimizar el modelo.

### A.2.1. Requisitos Mínimos

- Procesador de 32 bits.
- 1GB de memoria RAM.
- 5MB de espacio disponible en disco, más el necesario para la instalación del intérprete de Python y sus dependencias, así como espacio suficiente para albergar la información generada durante la ejecución.

### A.2.2. Requisitos Recomendados

- Procesador de 64 bits con múltiples núcleos para hacer uso de la ejecución paralela de hilos.
- 4GB de memoria RAM.
- 5MB de espacio disponible en disco, más el necesario para la instalación del intérprete de Python y sus dependencias, así como espacio suficiente para albergar la información generada durante la ejecución.

### A.2.3. Requisitos Software

- Sistemas Operativos *GNU/Linux* o *Windows*, siendo prefreribles las últimas versiones.

- Intérprete de Python *v2.7.13* o Python *v3.5.3*.
- *Git*.
- Bibliotecas de las cuales depende nuestro *software*:
  - *SciPy v1.1.0*
  - *NumPy v1.15.2*
  - *Pandas v0.23.4*
  - *Scikit-Learn v0.20.0*
  - *Sacred v0.7.3*

## A.3. Instalación del *software*

El *software* desarrollado viene incluido en un CD-ROM adjunto a este manual. La aplicación en sí no necesita de instalación alguna, pudiéndose ejecutar diréctamente desde el disco. Sin embargo, sería deseable el copiado de los ficheros al disco duro para poder almacenar los resultados de los experimentos en una subcarpeta dentro del directorio que contiene al *framework*.

Para poder ejecutar el programa será necesario tener instalado un intérprete de Python y las dependencias mencionadas en el apartado A.2.3. En los siguientes apartados se indicará al detalle como realizar la instalación de cada uno de estos elementos, utilizándose *Debian GNU/Linux* como distribución ejemplo en el caso de los sistemas *Linux*.

### A.3.1. Python

#### *GNU/Linux*

Las distintas distribuciones suelen incorporar intérpretes de Python preinstalados, tanto de las versiones `2.x` como de las `3.x`. En caso de que esto no ocurra, será necesario instalar uno de la siguiente forma:

```
apt-get install python2.x
apt-get install python3.x
```

En los comandos anteriores, `x` indica una versión específica que se encuentre disponible para nuestra distribución. Si se busca instalar una versión no disponible por medio de los gestores de paquetes, es posible hacerlo mediante un repositorio de terceros o instalando directamente desde la fuente.

***Windows***

Para instalar el intérprete en esta familia de sistemas, bastará con descargar un ejecutable de la versión que deseemos instalar de la página oficial de Python, procediendo a ejecutarlo posteriormente. En la ventana en la que confirmaremos la instalación, debemos indicar al instalador que deseamos añadir el ejecutable del intérprete de Python al *path* del usuario.

Este instalador también incluye varios elementos adicionales como son la documentación y *pip* entre otros.

### A.3.2. *Pip* y Entornos Virtuales

*pip* es un gestor de paquetes de Python que facilita la instalación, mantenimiento y borrado de estos. En conjunto con con el sistema de entornos virtuales, esta herramienta nos ayudará a instalar las dependencias de paquetes del *framework* de forma aislada, sin que estos se instalen al nivel del Sistema Operativo. De esta forma podremos evitar conflictos de dependencias con otras aplicaciones.

***GNU/Linux***

La mayor parte de las distribuciones cuentan con un paquete con el cual instalar *pip*:

```
apt-get install python-pip
```

También es posible instalar *pip* de forma directa mediante el intérprete de Python:

```
python -m pip install --user --upgrade pip
```

Como hemos comentado anteriormente, utilizaremos *pip* dentro de entornos virtuales, por lo que también hemos de instalar las utilidades necesarias para crearlos. Esto se realiza mediante `virtualenv` en Python2 o con `venv` en Python3, siendo únicamente necesario instalar el primero, ya que python3 contiene a `venv` dentro de la biblioteca estándar del lenguaje. Instalaremos `virtualenv` mediante el comando:

```
python -m pip install --user virtualenv
```

Una vez instalado, crearemos y activaremos un entorno virtual en Python2 con

```
python2 -m virtualenv py2env
source py2env/bin/activate
```

Para realizar la tarea análoga en Python3 únicamente habra que sustituir `virtualenv` por `venv`. Una vez activado el entorno virtual, se procederá a la instalación de las dependencias de la aplicación. Mencionar también que, para poder utilizar el *framework* será necesario siempre acceder al entorno virtual donde se encuentren instalados los paquetes necesarios.

#### *Windows*

Salvo que especifiquemos lo contrario, el instalador del intérprete de Python también instalará *pip* al mismo tiempo. Sin embargo, si la versión des-

cargada es anterior a una determinada, será necesario instalarlo de forma separada, para ello ejecutaremos los siguientes comandos:

```
curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
python get-pip.py --user
```

Para hacer uso de entornos virtuales, también debemos de instalar la herramienta con la que crearemos y accederemos a los mismos, *virtualenv*. Para su instalación basta con ejecutar la siguiente línea:

```
pip install virtualenv
```

Para crear, activar y desactivar el entorno virtual realizaremos lo siguiente:

```
virtualenv py3env
py3env\Scripts\activate.bat
py3env\Scripts\deactivate.bat
```

### A.3.3. Git

Este sistema de control de versiones será necesario para poder ejecutar la aplicación, ya que `sacred` hace uso del mismo en sus fuentes. Además, aporta una gran utilidad a los desarrolladores de *software*.

#### *GNU/Linux*

Por lo general, *Git*, al igual que Python, suele venir por defecto en todas las distribuciones Linux. En caso de que este no se encuente instalado en el sistema, fuere cual fuese el motivo, es posible instalarlo mediante el siguiente comando:

```
apt-get install pip
```

***Windows***

A diferencia de los sistemas *GNU/Linux*, *Windows* no integra esta herramienta entre sis utilidades base. Para instalar el programa bastará con descargar el instalador desde la página oficial de *Git* [13] y proceder con su instalación, la cual, gracias a su asistente, resulta sencilla de realizar, dando a elegir múltiples opciones.

### A.3.4. Dependencias

La instalación de los paquetes se realiza cómodamente gracias a *pip*, necesitando de los mismos comandos en cualquiera de los dos Sistemas Operativos que venimos tratando.

```
pip install numpy
pip install scipy
pip install pandas
pip install sklearn
pip install sacred
```

Con todos los paquetes instalados, ya deberíamos ser capaces de ejecutar el *framework*, aunque es posible que si los paquetes instalados difieren de las versiones listadas en este documento, pueden existir errores por falta de compatibilidad entre versiones.

## A.4. Desinstalación del *Software*

Como la aplicación en sí misma no necesita de instalación alguna, para eliminarla del sistema únicamente es necesario eliminar los ficheros y direc-

torios asociados a ella. Por su parte, dado que las distintas bibliotecas de las cuales dependía el *framework* han sido instaladas dentro de un entorno virtual, para desinstalarlas bastará con eliminar la carpeta sobre la que se definió el mismo (desactivando primero el entorno).

Los únicos elementos que deben ser desinstalados son el intérprete de Python y *pip*, borrado que puede ser efectuado mediante las herramientas de que disponga el Sistema Operativo específico para esta tarea. Ten en cuenta que en muchas distribuciones de *GNU/Linux* esto no es posible, ya que el propio sistema tiene alguna funcionalidad escrita sobre el lenguaje.

## A.5. Ejecución del *Framework*

Llevar a cabo un experimento una vez se ha definido la configuración del mismo es un hecho trivial, siendo necesario un único comando con una sintaxis sencilla:

```
python config.py with experiment_file.json
```

Este comando simplemente utiliza el intérprete de Python para llamar a la interfaz de nuestro *framework*, indicándole que el fichero de configuración donde se encuentra definido nuestro experimento es `experiment_file.json`. Sin embargo, ejecutar el programa de esta forma cuenta con dos problemas: el primero es el exceso de información que muestra `sacred` por defecto, el segundo es la imposibilidad de reproducir los resultados de un experimento si no se da un valor a la semilla que controla los métodos pseudo-aleatorios. Para solucionar esto se modifica el comando de la siguiente forma:

```
python config.py with experiment_file.json seed=12345 -l ERROR
```

donde `seed=12345` define el valor de la semilla para los procesos pseudo-aleatorios, que debe encontrarse definido después de `with`, mientras que `-l`

`ERROR` define el nivel de verbosidad de `sacred` (existen múltiples niveles que pueden ser consultados en la documentación del paquete).

## A.6. Ficheros de Configuración

Como hemos indicado a lo largo del documento, para que el *framework* pueda funcionar es necesario que el usuario le indique una serie de parámetros, constituyéndose este conjunto de variables, algunas opcionales y otras obligatorias, en un experimento. Estas órdenes le serán transmitidas a la aplicación a través de ficheros de configuración en formato JSON, de sintaxis relativamente sencilla.

Los ficheros JSON consisten en un conjunto de pares clave-valor, donde la clave debe ser forzosamente una cadena y donde su valor puede ser: una cadena, un número, una lista de valores (definiendo todos los valores dentro de corchetes) o un objeto (delimitado por llaves, que a su vez contiene pares clave-valor). El fichero estará delimitado por una pareja de llaves.

Para que se entienda mejor el formato de los ficheros de configuración, es mejor ilustrarlo mediante un ejemplo, mostrado en la figura A.1.

Como se puede observar, existen dos partes bien diferenciadas:

1. **Configuración General**: Aquí se incluirán los elementos indispensables para el funcionamiento del experimento, así como algunas variables que afectan de forma general a la ejecución del mismo. Las distintas variables que se contemplan dentro de este apartado de la configuración son:

    - Carpeta que alberga los conjuntos de datos.
    - Lista con los nombres de los conjuntos de datos.
    - Número de *folds* para la validación cruzada.

```
{
  "general_conf": {

    "basedir": "path/to/datasets/folder/",
    "datasets": ["dataset1", "dataset2"],
    "hyperparam_cv_nfolds": 3,
    "jobs": 10,
    "output_folder": "my_runs/",
    "metrics": ["ccr", "mae", "mze"],
    "cv_metric": "mae"
  },

  "configurations": {

    "configuration_1": {
      "classifier": "path.to.classifier",
      "parameters": {
        "parameter1": ["valor1", "valor2"],
        "parameter2": 3.1415
      }
    }

  }
}
```

Figura A.1: Ejemplo de un fichero de configuración

- Número de hilos que creará el programa para su ejecución (solo durante la fase de validación cruzada).
- Carpeta donde almacenar los resultados.
- Lista de métricas a calcular para cada modelo.
- Métrica con la que guiar la validación cruzada.

2. **Configuración de Algortimos**: En este apartado se añadirá una configuración (objeto JSON) por clasificador. Cada clasificador constará de dos elementos:
   - Dirección del clasificador, que puede hacer referencia a un fichero en el sistema o ser relativa a una biblioteca.
   - Hiperparámetros, con los cuales optimizar el modelo durante la fase de validación cruzada (si al menos uno de ellos cuenta con más de un valor posible).

Es necesario aclarar que, para que el sistema pueda utilizar algoritmos desarrollados por usuarios, es necesario que el nombre del fichero que alberga al clasificador y el nombre de la clase que lo implementa sean idénticos, de lo contrario la carga del mismo fallará.

## A.7. Formato Bases de Datos

Como se ha visto en la sección A.6, es necesario indicar la carpeta y el nombre de los conjuntos de datos si queremos utilizarlos. De esta manera, y según el ejemplo A.1, el *framework* esperará los ficheros pertenecientes al conjunto de datos `dataset1` dentro de la carpeta `path/to/datasets/folder/dataset1/`.

Siguiendo con el ejemplo, los ficheros que contengan los datos dentro de esta carpeta no pueden tener nombres arbitrarios, sino que deben venir indicados de la siguiente manera: `train_dataset1.csv` en caso de ser un

fichero que contenga datos para usar durante la fase de entrenamiento, o `test_dataset1.csv` en el caso contrario. Si la BBDD ha sido fragmentada en distintas particiones, el tipo de fichero, `.csv`, se sustituirá por el número de partición que le corresponda.

Por su parte, el interior de los ficheros es similar sin importar su nomenclatura. Están estructurados como un fichero CSV (*Comma Separated Value*), aunque puede usarse cualquier delimitador de campo dentro de ellos. La sintaxis de los mismos se puede ver reflejada en la Figura A.2.

```
1.0 0.0 0.0 1.0 0.0 0.0 1.0 0.0 0.0 1.0 0.0 0.0 1
1.0 0.0 0.0 1.0 1.0 0.0 1.0 0.0 0.0 1.0 1.0 0.0 1
0.0 1.0 0.0 0.0 0.0 1.0 0.0 1.0 0.0 0.0 0.0 1.0 2
1.0 0.0 0.0 1.0 0.0 1.0 1.0 0.0 0.0 1.0 0.0 1.0 2
1.0 0.0 0.0 0.0 1.0 1.0 1.0 0.0 0.0 0.0 1.0 1.0 3
1.0 0.0 0.0 1.0 1.0 1.0 1.0 0.0 0.0 1.0 1.0 1.0 3
```

Figura A.2: Ejemplo de un fichero de datos

Dentro de estos ficheros, cada línea representará un ítem de datos y cada columna una variable del patrón, con excepción de la columna final, reservada para definir la clase a la que pertenece. Las variables pueden representarse como números enteros o reales, mientras que las etiquetas de clase deben especificarse como valores enteros (no admitiéndose valores nominales).

## A.8. Formato de Salida de Información

Durante y tras la ejecución sin errores de un experimento, el programa almacena toda la información de interés producida. Para ilustrar al lector acerca de la información que almacena el programa, seguiremos utilizando el ejemplo de configuración mostrado en A.1, donde podemos observar como los resultados de salida se almacenarán en un directorio ubicado en el *path* relativo `my_runs`. En este directorio se generará una subcarpeta con el nombre `exp-año-mes-día-hora-minuto-segundo` (donde todos los elementos menos `exp` son variables), concerniente al experimento ejecutado en ese instante

determinado. Lo que encontraremos dentro de dicha carpeta será lo siguiente:

- **Subcarpetas BBDD-Clasificador**: Tantas subcarpetas como combinaciones de conjuntos de datos y clasificadores diferentes hayamos indicado en el fichero que configura el experimento. En nuestro caso serán 2, que se nombrarán `dataset1-configuration1` y `dataset2--configuration1`. A su vez, dentro de estas subcarpetas encontraremos:

  - Una carpeta albergando los mejores modelos obtenido en la fase de validación, uno por partición. Estos modelos se almacenan en ficheros con un formato definido por el paquete `pickle` de Python, en *formato binario*.
  - Una carpeta almacenando las predicciones efectuadas por el mejor modelo para los datos de entrenamiento y, si existiesen, de generalización. El formato en el que se almacenan es muy sencillo: en cada línea solo aparece el valor entero, que representa la clase predicha para el patrón que ocupa la misma posición en la base de datos en la que se encuentra definido. Se puede ver un ejemplo de este tipo de ficheros en la figura A.3.
  - Un CSV conteniendo los valores obtenidos para las diferentes métricas especificadas en el fichero de configuración. Cada métrica se calculará independientemente para los conjuntos de entrenamiento y de test. Además también guardará los valores de los hiperparámetros con los que se obtuvieron los resultados y el tiempo computacional que tardó en superar cada fase. Existirá una fila diferente por cada partición. Se puede apreciar un ejemplo de este tipo de ficheros en la figura A.4.

- **Ficheros Resumen**: Se crearán dos ficheros, uno para las métricas de entrenamiento y otro para las de generalización. En estos ficheros CSV habrá tantas líneas como subcarpetas BBDD-clasificador y columnas

como el doble de métricas y tiempos computacionales. En ellos se indicará la media y la desviación típica a lo largo de las particiones de cada una de esas combinaciones, para cada métrica y tiempo computacional. Se puede ver ilustrado en la figura A.5.


Figura A.3: Ejemplo de fichero albergando predicciones

| | A | B | C | D | E | F | G | H | I | J | K | L |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | C | tol | penalty | dtype | base_classifier | decision_method | ccr_train | ccr_test | mae_train | mae_test | mze_train |
| 2 | 0 | 1 | 0,0001 | l2 | ordered_partitions | sklearn.linear_model.LogisticRegression | frank_hall | 0,987577639751553 | 1 | 0,012422360248447 | 0 | 0,012422360248447 |
| 3 | 1 | 10 | 0,0001 | l1 | ordered_partitions | sklearn.linear_model.LogisticRegression | frank_hall | 0,987577639751553 | 1 | 0,012422360248447 | 0 | 0,012422360248447 |
| 4 | 2 | 1 | 0,0001 | l2 | ordered_partitions | sklearn.linear_model.LogisticRegression | frank_hall | 0,993788819875776 | 0,962962962963 | 0,00621118012422 | 0,037037037037037 | 0,00621118012422 |
| 5 | 3 | 10 | 0,0001 | l2 | ordered_partitions | sklearn.linear_model.LogisticRegression | frank_hall | 0,993788819875776 | 0,981481481481482 | 0,00621118012422 | 0,018518518518519 | 0,00621118012422 |
| 6 | 4 | 1 | 0,0001 | l1 | ordered_partitions | sklearn.linear_model.LogisticRegression | frank_hall | 0,987577639751553 | 1 | 0,012422360248447 | 0 | 0,012422360248447 |
| 7 | 5 | 100 | 0,001 | l2 | ordered_partitions | sklearn.linear_model.LogisticRegression | frank_hall | 0,993788819875776 | 0,981481481481482 | 0,00621118012422 | 0,018518518518519 | 0,00621118012422 |
| 8 | 6 | 10 | 0,0001 | l1 | ordered_partitions | sklearn.linear_model.LogisticRegression | frank_hall | 0,987577639751553 | 1 | 0,012422360248447 | 0 | 0,012422360248447 |
| 9 | 7 | 1 | 0,01 | l1 | ordered_partitions | sklearn.linear_model.LogisticRegression | frank_hall | 0,993788819875776 | 0,962962962963 | 0,00621118012422 | 0,037037037037037 | 0,00621118012422 |
| 10 | 8 | 10 | 0,0001 | l2 | ordered_partitions | sklearn.linear_model.LogisticRegression | frank_hall | 0,993788819875776 | 0,981481481481482 | 0,00621118012422 | 0,018518518518519 | 0,00621118012422 |
| 11 | 9 | 10 | 0,01 | l1 | ordered_partitions | sklearn.linear_model.LogisticRegression | frank_hall | 0,975155279503105 | 0,981481481481482 | 0,024844720496894 | 0,018518518518519 | 0,024844720496895 |
| 12 | 10 | 1 | 0,0001 | l1 | ordered_partitions | sklearn.linear_model.LogisticRegression | frank_hall | 0,993788819875776 | 0,962962962963 | 0,00621118012422 | 0,037037037037037 | 0,00621118012422 |
| 13 | 11 | 10 | 0,001 | l1 | ordered_partitions | sklearn.linear_model.LogisticRegression | frank_hall | 1 | 0,907407407407407 | 0 | 0,092592592592593 | 0 |
| 14 | 12 | 1 | 0,0001 | l2 | ordered_partitions | sklearn.linear_model.LogisticRegression | frank_hall | 0,987577639751553 | 1 | 0,012422360248447 | 0 | 0,012422360248447 |
| 15 | 13 | 100 | 0,0001 | l2 | ordered_partitions | sklearn.linear_model.LogisticRegression | frank_hall | 1 | 0,962962962963 | 0 | 0,037037037037037 | 0 |
| 16 | 14 | 10 | 0,1 | l1 | ordered_partitions | sklearn.linear_model.LogisticRegression | frank_hall | 0,968944099378882 | 0,944444444444444 | 0,031055900621118 | 0,055555555555556 | 0,031055900621118 |
| 17 | 15 | 1 | 0,0001 | l2 | ordered_partitions | sklearn.linear_model.LogisticRegression | frank_hall | 0,987577639751553 | 0,981481481481482 | 0,012422360248447 | 0,018518518518519 | 0,012422360248447 |
| 18 | 16 | 10 | 0,0001 | l1 | ordered_partitions | sklearn.linear_model.LogisticRegression | frank_hall | 0,987577639751553 | 1 | 0,012422360248447 | 0 | 0,012422360248447 |
| 19 | 17 | 10 | 0,001 | l2 | ordered_partitions | sklearn.linear_model.LogisticRegression | frank_hall | 0,987577639751553 | 1 | 0,012422360248447 | 0 | 0,012422360248447 |
| 20 | 18 | 10 | 0,001 | l1 | ordered_partitions | sklearn.linear_model.LogisticRegression | frank_hall | 0,993788819875776 | 0,981481481481482 | 0,00621118012422 | 0,018518518518519 | 0,00621118012422 |
| 21 | 19 | 1 | 0,0001 | l1 | ordered_partitions | sklearn.linear_model.LogisticRegression | frank_hall | 0,987577639751553 | 1 | 0,012422360248447 | 0 | 0,012422360248447 |
| 22 | 20 | 100 | 0,001 | l1 | ordered_partitions | sklearn.linear_model.LogisticRegression | frank_hall | 0,993788819875776 | 0,962962962963 | 0,00621118012422 | 0,037037037037037 | 0,00621118012422 |
| 23 | 21 | 1 | 0,0001 | l2 | ordered_partitions | sklearn.linear_model.LogisticRegression | frank_hall | 0,987577639751553 | 1 | 0,012422360248447 | 0 | 0,012422360248447 |
| 24 | 22 | 1 | 0,01 | l2 | ordered_partitions | sklearn.linear_model.LogisticRegression | frank_hall | 0,975155279503105 | 0,962962962963 | 0,024844720496894 | 0,037037037037037 | 0,024844720496895 |
| 25 | 23 | 1 | 0,0001 | l2 | ordered_partitions | sklearn.linear_model.LogisticRegression | frank_hall | 0,993788819875776 | 0,962962962963 | 0,00621118012422 | 0,037037037037037 | 0,00621118012422 |
| 26 | 24 | 1 | 0,0001 | l1 | ordered_partitions | sklearn.linear_model.LogisticRegression | frank_hall | 0,987577639751553 | 0,981481481481482 | 0,012422360248447 | 0,018518518518519 | 0,012422360248447 |
| 27 | 25 | 10 | 0,001 | l2 | ordered_partitions | sklearn.linear_model.LogisticRegression | frank_hall | 0,987577639751553 | 0,981481481481482 | 0,012422360248447 | 0,018518518518519 | 0,012422360248447 |
| 28 | 26 | 1 | 0,0001 | l1 | ordered_partitions | sklearn.linear_model.LogisticRegression | frank_hall | 0,987577639751553 | 1 | 0,012422360248447 | 0 | 0,012422360248447 |
| 29 | 27 | 1 | 0,001 | l2 | ordered_partitions | sklearn.linear_model.LogisticRegression | frank_hall | 0,993788819875776 | 0,981481481481482 | 0,00621118012424 | 0,018518518518519 | 0,00621118012424 |

Figura A.4: Ejemplo de fichero de salida para una configuración y un conjunto de datos específicos

| | A | B | C | D | E | F | G | H | I | J |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | ccr_mean | ccr_std | mae_mean | mae_std | mze_mean | mze_std | mmae_mean | mmae_std | cv_time_mean |
| 2 | computer1-SVC | 0,587215768660405 | 0,017051093496898 | 0,484954275827978 | 0,027555059502671 | 0,412784231339595 | 0,017051093496898 | 0,685981347120985 | 0,119754935063063 | 0,004831359873013 |
| 3 | machine-SVC | 0,576271186440678 | 0,056348302220709 | 0,465254237288135 | 0,070523869519485 | 0,423728813559322 | 0,056348302220709 | 0,733333333333333 | 0,119697474409935 | 0,006412807484062 |
| 4 | bondrate-SVC | 0,577777777777778 | 0,031964220099026 | 0,624444444444444 | 0,061857166663923 | 0,422222222222222 | 0,031964220099026 | 2,9 | 0,305128576629365 | 0,00346692241779 |
| 5 | tae-SVC | 0,571052631578947 | 0,049425972282576 | 0,461403508771193 | 0,06509922231838 | 0,428947368421053 | 0,049425972282576 | 0,727991452991453 | 0,115466658784136 | 0,004166838563218 |
| 6 | bank1-SVC | 0,584647506755097 | 0,039944022394943 | 0,441605256693687 | 0,05452031252272 | 0,415352493244903 | 0,039944022394943 | 0,635065998947214 | 0,10618871782255 | 0,003969621719146 |
| 7 | squash-stored-SVC | 0,425641025641026 | 0,039032020261917 | 0,574358974358974 | 0,039032020261917 | 0,574358974358974 | 0,039032020261917 | 1 | 0 | 0,002585185255323 |
| 8 | newthyroid-SVC | 0,951851851851852 | 0,024606068104812 | 0,048765432098766 | 0,025495113028923 | 0,048148148148148 | 0,024606068104812 | 0,213377192982456 | 0,117295639690809 | 0,004246535641806 |
| 9 | toy-SVC | 0,934222222222222 | 0,024242284917747 | 0,065777777777778 | 0,024242284917747 | 0,065777777777778 | 0,024242284917747 | 0,166135246762182 | 0,070006684319194 | 0,00723586136792 |
| 10 | housing-SVC | 0,651699029126214 | 0,023459846285395 | 0,387621359223301 | 0,034730572976089 | 0,348300970873786 | 0,023459846285395 | 0,55191637630662 | 0,082371888577682 | 0,020203220576656 |
| 11 | pyrim-SVC | 0,4875 | 0,090785235938051 | 0,614583333333333 | 0,124542290663624 | 0,5125 | 0,090785235938051 | 1,16 | 0,247938871923154 | 0,004663568893257 |
| 12 | | | | | | | | | | |

Figura A.5: Ejemplo de fichero resumen