ESCUELA POLITÉCNICA
SUPERIOR DE CÓRDOBA
Universidad de Córdoba

EPSC

TRABAJO DE FIN DE GRADO

Grado en Ingeniería Informática

Desarrollo de los algoritmos de clasificación ordinal KDLOR, SVC1V1 y SVC1VA para ORCA-Python

Autor
Cristian Torres Pérez de Gracia

Directores
Pedro Antonio Gutiérrez Peña
David Guijo Rubio

Septiembre, 2024

UNIVERSIDAD D CÓRDOBA

Pedro Antonio Gutiérrez Peña, Catedrático del Departamento de Informática y Análisis Numérico en la Escuela Polictécnica Superior de la Universidad de Córdoba e investigador del Grupo de Investigación AYRNA.

Informa:

Que el presente Trabajo de Fin de Grado titulado *Desarrollo de los algoritmos de clasificación ordinal KDLOR, SVC1V1 y SVC1VA para ORCA-Python*, constituye la memoria presentada por **Cristian Torres Pérez de Gracia** 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 necesarios exigidas en este tipo de trabajos.

Y para que conste, se expide y firma el presente certificado en Córdoba, 10 de septiembre de 2024.

El Director:

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

David Guijo Rubio investigador postdoctoral en la Universidad de Córdoba del Dpto. Informática y Análisis Numérico y miembro investigador del grupo AYRNA y encargado de dar soporte técnico y teórico para las tareas de análisis, diseño y programación.

Informa:

Que el presente Trabajo de Fin de Grado titulado *Desarrollo de los algoritmos de clasificación ordinal KDLOR, SVC1V1 y SVC1VA para ORCA-Python*, constituye la memoria presentada por **Cristian Torres Pérez de Gracia** 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 necesarios exigidas en este tipo de trabajos.

Y para que conste, se expide y firma el presente certificado en Córdoba, 10 de septiembre de 2024.

El Director:

Fdo: Prof. Dr. David Guijo Rubio

Índice general

Índice de figuras

Índice de cuadros

Parte I

Introducción

Capítulo 1

Introducción

La convergencia entre la informática y la comunicación ha creado una sociedad que se nutre de la información. A la información que se encuentra sin tratar se la conoce como datos. Si los datos se caracterizan como hechos registrados, entonces la información es el conjunto de patrones que subyacen de los datos. En la actualidad, hay una gran cantidad de información almacenada en las bases de datos, siendo esta de suma importancia por lo que cada vez se usan más procedimientos de análisis de datos. La minería de datos se encarga de extraer esa información que resulta de utilidad a partir del conjunto de datos generado por procedimientos estándares. En este punto es importante introducir el concepto de aprendizaje automático, que es el área que usa estos datos extraídos de numerosos procedimientos previos para obtener conclusiones más o menos generales a partir de casos reales que han ocurrido previamente.

1.1. Definición de aprendizaje automático

De forma simple el aprendizaje automático, o *machine learning*, proporciona mecanismos que permiten al ordenador aprender por si mismo a resolver un determinado problema. Ahora si lo que buscamos es una definición orientada más a la ingeniería Tom Mitchell[1] decía que un programa aprende de la experiencia E con respecto a una tarea T y alguna medida de rendimiento P, si su rendimiento en T, medido por P, mejora con la experiencia E.

Como puede ser algo difícil de entender procederemos a explicarlo con un ejemplo. Supongamos que tenemos un filtro de *spam* que es un programa de

machine learning que aprende a detectar el *spam* a partir de unas muestras de correo de *spam* (etiquetadas por los usuarios) y otras muestras de emails normales(no *spam*, también conocidos como *ham*). Estos ejemplos que el sistema usa para aprender son llamados conjuntos de entrenamiento. En este caso, la tarea T sería detectar el spam en los nuevos emails, la experiencia E son los datos de entrenamiento y la medida de rendimiento P necesita ser definida; por ejemplo, puedes usar el ratio de los emails correctamente clasificados. Esta medida de rendimiento se conoce como *accuracy* y es utilizada a menudo en tareas de clasificación.

Una vez terminado de definir lo que es el *machine learning* citaremos unos cuantos problemas para los cuales es ideal:

- Problemas para los que las soluciones existentes requieren mucho ajuste manual o listas de reglas: un algoritmo de *machine learning* no solo ayuda a simplificar el código sino que además mejora el rendimiento del mismo.
- Problemas complejos para los cuales no existen un buena solución utilizando métodos tradicionales.
- Entornos fluctuantes: un sistema de *machine learning* puede adaptarse a nueva información.
- Obtener información sobre problemas complejos y grandes cantidades de datos.

1.2. Los diferentes tipos de sistemas de *Machine Learning*

En el apartado anterior, veíamos qué era el *machine learning* y algunos de los problemas para los que un algoritmo de *machine learning* aportaba una mejor solución que los métodos tradicionales. En este apartado profundizaremos más en los diferentes tipos *machine learning* que existen:

- Si han sido o no entrenados bajo supervisión humana (supervisados, no supervisados, semi supervisados y aprendizaje por refuerzo).
- Si pueden aprender o no sobre la marcha(*online* versus *batch learning*)

- Si funcionan comparando nuevos puntos de datos con puntos ya conocidos, o si detectan patrones en los datos de entrenamiento y construyen un modelo predictivo, como lo hacen los científicos (aprendizaje basado en instancias versus aprendizaje basado en modelos).

Estos criterios de clasificación no son exclusivos entre si, sino que se pueden combinar. Por ejemplo, un filtro de *spam* que aprende sobre la marcha usando una red neuronal entrenada usando ejemplos de *spam* y *ham*; esto lo convierte en un sistema de aprendizaje *online*, basado en modelos y supervisado.

1.3. Tipos de aprendizaje supervisado y no supervisado

Los sistemas de *machine learning* pueden ser clasificados según la cantidad y tipo de supervisión que reciban durante su entrenamiento. Existen 4 categorías: aprendizaje supervisado, aprendizaje no supervisado, semisupervisado y refuerzo de aprendizaje.

1.3.1. Aprendizaje supervisado

En el aprendizaje supervisado, los datos de entrenamiento que alimentan al algoritmo incluyen las soluciones deseadas, llamadas etiquetas.

Una típica tarea de aprendizaje supervisado es la clasificación. El filtro de *spam* es un buen ejemplo de esto: es entrenado con diferentes emails de muestra junto con su clase (*spam* o *ham*) y debe aprender a clasificar nuevos emails.

Otro ejemplo típico sería la predicción de un valor numérico, tal como el precio de un coche, dado un conjunto de características (millas, edad, marca, etc) llamadas predictores. A este tipo de aprendizaje supervisado se le conoce como regresión. Para entrenar el sistema necesitas darle muestras de muchos coches, incluyendo sus predictores y sus etiquetas (por ejemplo su precio).

En *machine learning* un atributo es un tipo de dato (por ejemplo kilometraje), mientras que una característica tiene un significado dependiendo del contexto, pero generalmente significa un atributo más su valor (kilometraje = 15000). No obstante, mucha gente usa la palabra atributo y característica

de manera indistinta.

Notase que algunos algoritmos de regresión se pueden utilizar para la clasificación y viceversa. Un ejemplo de esto es la regresión logística que es usada comúnmente para clasificación, debido a que la salida produce un valor que se corresponde con la probabilidad de pertenecer a la clase dada (20 % de ser *spam*).

Aquí tenemos algunos de los más importantes algoritmos de aprendizaje supervisado:

- K vecinos más cercanos
- Regresión lineal
- Regresión logística
- SVMs (máquina de vectores soporte)
- Árboles de decisión y *Random forests*
- Redes neuronales

A continuación en la figura 1.1 podemos observar un ejemplo de red neuronal.

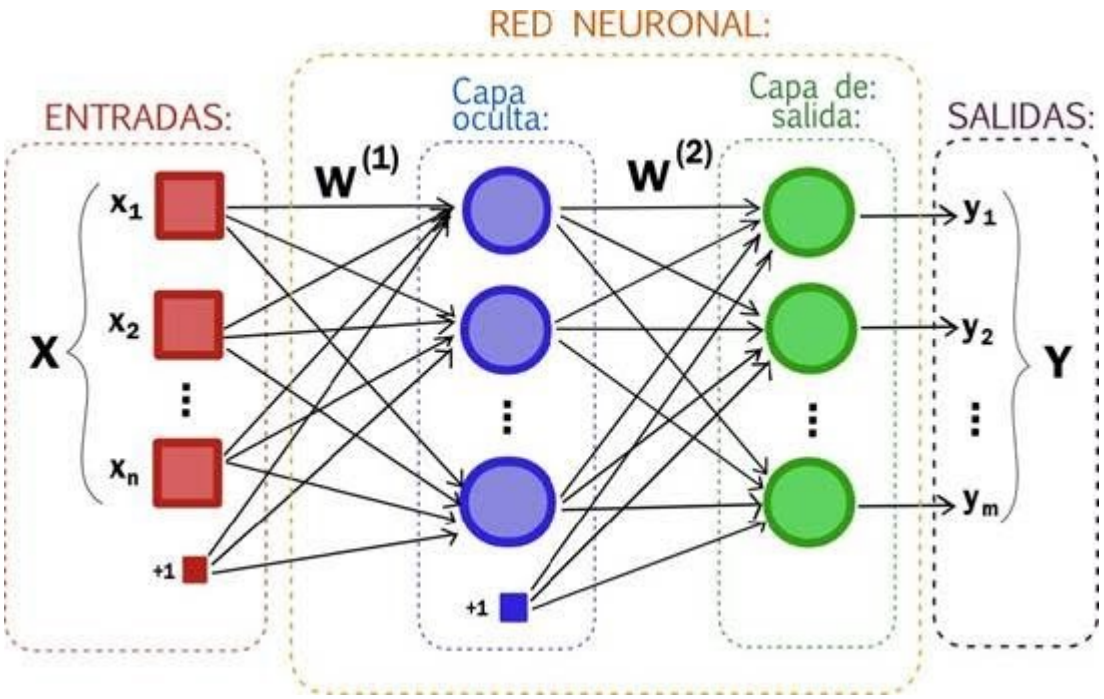

Figura 1.1: Ejemplo de una red neuronal

1.3.2. Aprendizaje no supervisado

A diferencia del aprendizaje supervisado en el aprendizaje no supervisado, las muestras de entrenamiento no están etiquetadas, es decir, se construyen descripciones, hipótesis o teorías a partir de un conjunto de hechos u observaciones, sin que exista información adicional acerca de cómo deberían agruparse los ejemplos del conjunto de entrenamiento.

Aquí hay algunos de los más importantes algoritmos de aprendizaje no supervisado:

- ***Clustering***
  - K Means
  - DBSCAN
  - HCA (*Hierarchical Cluster Analysis*)
- **Detección de anomalías y detección de novedades**

  - Máquinas de vectores soporte de una sola clase
  - *Isolation Forest*

- **Visualización y reducción de dimensionalidad**
  - PCA (*Principal Component Analysis*)
  - Kernel PCA
  - LLE(*Locally-Linear Embedding*)
  - t-SNE(*t-distributed Stochastic Neightbor Embedding*)

- **Reglas de asociación**
  - Apriori
  - Eclat

Se intentará explicar al lector la utilidades de los algoritmos de *clustering* con un ejemplo. Imaginemos que poseemos un blog y tenemos muchos datos sobre el mismo. Sería interesante aplicar un algoritmo de *clustering* para intentar detectar grupos de visitantes con ciertas similitudes. Aunque tienes esta información no dispones de forma de contarle al algoritmo a que grupo pertenece cada visitante, es decir, la información no se encuentra etiquetada. Sin embargo, el algoritmo de *clustering* encuentra las conexiones sin necesidad de ayuda. Supongamos, que una vez ejecutado el algoritmo encontramos que el 40 % de los visitantes son hombre que les encanta leer comic por la tarde, mientras que un 20 % son jovenes que aman la ciencia ficción y lo visitan durante el fin de semana, y así con los demás grupos. Si además, ejecutáramos un algoritmo de *clustering* jerárquico sería posible subdividir estos grupos en grupos más pequeños. Con esta información sería posible planificar a que grupo de personas va dirigido cada post.

Los algoritmos de visualización son otro buen ejemplo de algoritmos de aprendizaje no supervisados, se le proporciona un gran cantidad de datos complejos y no etiquetados, y ellos producen salidas 2D o 3D de los datos que pueden representarse fácilmente. Estos algoritmos tratan de preservar tanta información como le es posible (por ejemplo intentando mantener separados los *clusters* en el espacio de entrada para que no sobrepongan en la visualización), de tal forma que pueda entender como ser organizan los datos y quizás identificar patrones insospechados. La figura 1.2 es un ejemplo de este tipo de algoritmos.

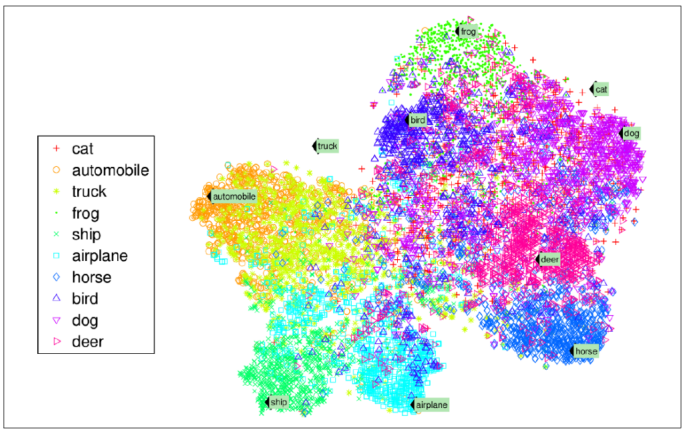

Figura 1.2: Ejemplo de un algoritmo de visualización

Otra ejemplo seria la reducción de dimensionalidad, en el cual la meta es simplificar los datos sin llegar a perder demasiada información. Una manera de hacerlo por ejemplo sería combinar varias características correlacionadas en una sola. Por ejemplo, el kilometraje de un coche puede estar muy relacionado con la edad del mismo, de esta forma el algoritmo de reducción de dimensionalidad combinará estas dos características en una sola que represente el desgaste del coche. Esto se denomina extracción de características.

Como nota adicional cabe señalar, que a menudo es una buena idea reducir la dimensionalidad de los datos de entrenamiento mediante una algoritmo de reducción de dimensionalidad antes de usarlo con otro algoritmo de *machine learning* (como por ejemplo el aprendizaje supervisado), ya que no solo hará más rápida su ejecución y ocupará menos espacio en disco y memoria, sino que además en algunos casos puede llegar a mejorar el rendimiento del algoritmo.

Otro algoritmo de aprendizaje no supervisado sería la detección de anomalías, por ejemplo, detectar transacciones de crédito inusuales para prevenir fraudes o remover ruido de muestras de entrenamiento antes de que lo use otro algoritmo de aprendizaje. Al sistema se le muestra mayormente instancias normales durante el entrenamiento, de tal forma que sea capaz de reconocerlas y cuando vea una nueva instancia sepa diferenciar si es normal o, por el contrario, se trata de una anomalía. Una tarea bastante similar sería la detección de novedades. La diferencia es que la detección de novedades solo espera ver muestras normales durante el entrenamiento, mientras

que por el contrario, los algoritmos de detección de anomalías suelen ser normalmente más tolerantes, llegando a tener un buen rendimiento incluso con un pequeño porcentaje de ruido en la muestra de entrenamiento.

Por ultimo, se verán las reglas de asociación, cuya meta consiste en examinar una gran cantidad de datos y descubrir interesantes relaciones entre sus atributos. Por ejemplo, supongamos que posee un supermercado. Si ejecuta una regla de asociación en su registro de ventas, puede descubrir que la gente que compra salsa barbacoa y patatas fritas también suele comprar filetes o que la gente que compra pañales a menudo compra también cerveza. Por lo tanto, es posible que quiera colocar estos productos cerca del otro.

1.3.3. Aprendizaje semisupervisado

En los apartados anterior se veían los algoritmos de aprendizaje supervisado y no supervisado, en algunas ocasiones se dan las circunstancias de que tenemos datos de entrenamiento parcialmente etiquetados, normalmente una gran cantidad de datos no etiquetados y unos pocos etiquetados, esto se conoce como aprendizaje semisupervisado.

Los métodos de aprendizaje semisupervisado son especialmente relevantes en situaciones en las que obtener una cantidad suficiente de datos etiquetados es costoso, pero grandes cantidades de datos no etiquetados son relativamente fáciles de adquirir. En dichas situaciones, ni los métodos de aprendizaje supervisado ni los no supervisados conseguirán soluciones óptimas.

Véase como ejemplo, las redes de creencia profunda o DBN (*Deep Belieb Network*) están compuesta por múltiples capas que se tratan de forma no supervisada. Estas capas están basadas en las máquinas de Boltzmann restringidas (RBM) apiladas una encima de otras, es decir, cada capa oculta de una subred sirve como la capa visible para la siguiente. Como se menciono antes RBMs son entrenadas de manera secuencial de forma no supervisada y luego el sistema como conjunto es entrenado utilizando técnicas de aprendizaje supervisado.

1.3.4. Aprendizaje por refuerzo

En el aprendizaje por refuerzo, el sistema llama un agente que en este contexto, puede observar el entorno, seleccionar y realizar una acción, en

función de las mismas será recompensado o puede ser penalizado. Por lo tanto el agente debe aprender cual es la mejor estrategia para resolver el problema por si mismo.

1.4. Regresión ordinal

En el apartado se veía que los algoritmos de aprendizaje supervisado podían ser de clasificación y regresión. En clasificación los algoritmos predicen etiquetas o clases que se conocen previamente, mientras que en regresión son utilizados en donde la salida se trata de un valor numérico. En este apartado se verá un área que se encuentra a medio camino entre la clasificación y la regresión.

La regresión ordinal o clasificación ordinal es el área del aprendizaje automático, en el que las etiquetas muestran un orden natural entre sí. Un claro ejemplo de estos mismos es cuando se evalúa la satisfacción de un servicio entre 1 y 5 en escala de Likert, siendo 1 muy deficiente y 5 excelente. Cuando se trabaja con este tipo de problemas hay que tener en cuenta que los errores de clasificación no pueden tener el mismo peso, es decir, no se debe penalizar de la misma forma un servicio clasificado como muy deficiente siendo este excelente, a si cometiéramos el error de clasificarlo como deficiente o decente. Además, el orden de la información puede ser usado para construir modelos más precisos.

Los problemas de regresión ordinal son muy comunes en muchas áreas de la investigación, y se han tratado como problemas nominales estándar lo cual conlleva a soluciones poco óptimas. Algunos de los campos donde se encuentra presenta la regresión ordinal son las investigaciones médicas, estimación de edades, reconocimiento facial, clasificación de imágenes y texto, etc. Todo estos trabajos son ejemplos de aplicaciones específicamente diseñado para modelos de regresión ordinal, donde la consideración del orden mejora el rendimiento con respecto a su contraparte nominal.

Una vez presentado la regresión ordinal se expondrá el tema de este Trabajo de Fin de Grado que se va a centrar en la clasificación ordinal, más concretamente, en la implementación de 3 algoritmos (`KDLOR`, `SVC1V1` y `SVC1VA`) que estaban previamente desarrollados en `ORCA` [2] y para los cuales debemos obtener resultados similares para unos problemas utilizando el *framework* `ORCA-Python` [3, 4]. `ORCA` es un *framework* de `MATLAB` [5]

que implementa e integra una amplía gama de métodos de regresión ordinal y de métricas de rendimiento. La implementación se va a realizar en `ORCA-Python`, que es un *framework* escrito en `Python` integrado con los módulos de `scikit-Learn` [6] y `sacred` [7].

1.5. Principales desafíos del *Machine Learning*

Antes de pasar al siguiente capítulo donde se definirá el problema a tratar en este proyecto, es importante que el lector sea consciente de los principales desafíos a los que se enfrentan los algoritmos de aprendizaje automático como pueden ser que no existan la suficiente cantidad de datos de entrenamiento, el sobreaprendizaje, que los datos no sean representativo, entre otros.

1.5.1. Insuficiente cantidad de datos de entrenamiento

Para un niño aprender lo que es una manzana es relativamente fácil, todo lo que hace falta es decirle que es una manzana y repetir el proceso unas pocas veces. Ahora el niño sería capaz de reconocer una manzana. Sin embargo, en el aprendizaje automático hace falta bastante más datos para que esto funcione propiamente. Incluso para el problema más simple normalmente necesitarías miles de muestras, mientras que para problemas mas complejos como el reconocimiento de lenguaje o de imágenes la cantidad asciende a millones de muestras en el mejor de los casos.

1.5.2. Datos de entrenamiento no representativos

En orden para generalizar bien, es necesario que los datos de entrenamiento sean representativos de los nuevos casos que se quieran generalizar. La utilización de un conjunto de datos de entrenamiento que no es representativo, con lleva a que muy probablemente el modelo entrenado nunca va a hacer predicciones precisas. Por esto es crucial que el conjunto de entrenamiento sea representativo de los casos que se quieran generalizar. Esto es más fácil decirlo que hacerlo; si la muestra es muy pequeña tendrás muestras con ruido, pero si las muestras son demasiado grandes podrían ser no representativos resultando en muestras defectuosas. Esto es conocido con el nombre *sampling bias*, que es cuando unos miembros de una población tienen mas probabilidades de salir elegidos como muestra frente a otros.

1.5.3. Datos de baja calidad

Obviamente si tu conjunto de datos de entrenamientos, esta lleno de errores, *outliers* (es un valor atípico que se aleja considerablemente de la media del conjunto) y ruido, será muy difícil que el sistema haga un reconocimiento de patrones adecuados y por lo tanto es muy probable que el modelo entrenado no tenga buen rendimiento. Debido a esto a menudo es recomendable pasar tiempo limpiado el conjunto de datos de entrenamiento. Por ejemplo:

- Si algunas instancias son claramente valores anómalos, puede ayudar simplemente eliminarlos o intentar arreglar los errores manualmente.
- Si a algunas instancias les faltan características (por ejemplo una porción de tu lista de clientes no especificó su edad), debes decidir si vas a ignorar estos atributos, si vas a ignorar estas instancias, también es posible que te interese completar los valores faltantes (por ejemplo usando la media de la edad), o incluso entrenar un modelo con esa característica y otro sin la misma, etc.

1.5.4. Características irrelevantes

El sistema puede es solo capaz de aprender si el conjunto de entrenamiento tiene las suficientes características y no demasiadas irrelevantes. Una parte importante para el éxito de un proyecto de *machine learning* es extraer un buen conjunto de características de entrenamiento, a este proceso se le llama *feature engineering* y consiste en:

- Selección de las características: seleccionar las características mas útiles para entrenar entre todas las existentes.
- Extracción de características: consiste en combinar características ya existentes para crear nuevas que sean más útiles (como por ejemplo, los algoritmo de reducción de dimensionalidad que vimos anteriormente).
- Crear nuevas características reuniendo nuevos datos.

1.5.5. Sobreaprendizaje

Imagina que tienes varias manzanas, al ir a comerte una te das cuenta que esta podrida, ante lo cual puedes estar tentado a pensar que todas las demás manzanas también están en mal estado. Sobregeneralizar es algo

que no solo hacemos las personas, sino que los algoritmos de aprendizaje automático también pueden caer en esta trampa si no se es cuidadoso, a esto se le conoce como sobreaprendizaje. El sobreaprendizaje implica que el modelo funciona correctamente con los datos de entrenamiento pero no puede generalizar bien.

Los modelos complejos como las redes neuronales profundas pueden detectar patrones imperceptibles en los datos, pero si el conjunto de entrenamiento contiene mucho ruido o es demasiado pequeño, es más que probable que el modelo detecte patrones en el ruido también. El sobreaprendizaje ocurre cuando el modelo es demasiado complejo en relación a la cantidad o el ruido del conjunto de entrenamiento. Algunas posibles soluciones serían:

- Simplificar el modelo seleccionando menos parámetros (por ejemplo elegir un modelo lineal frente a un modelo polinómico de grado alto), reducir el numero de atributos en el conjunto de datos o limitando el modelo.
- Reunir más muestras de entrenamiento.
- Reducir el ruido en el conjunto de entrenamiento (como por ejemplo arreglar errores o eliminar ruido de la muestra).

Limitar o restringir un modelo para hacerlo más simple y reducir el riesgo de sobreaprendizaje se denomina como regularización. El grado de la regularización que se aplica durante el aprendizaje puede ser controlado mediante el uso de hiperparámetros. Los hiperparámetros son parámetros del algoritmo de aprendizaje no del modelo, es decir, como tal no afecta al algoritmo de aprendizaje sino que limita el entrenamiento del mismo. Si el valor del hiperparámetro es muy alto, conseguiremos un mal modelo; el modelo ciertamente no sobreaprenderá los datos de entrenamiento, pero tampoco encontrará una solución que sea óptima. Encontrar un valor óptimo para un hiperparámetro es una parte importante de construir un buen sistema de aprendizaje automático.

1.5.6. Infraaprendizaje

El infraaprendizaje es el proceso opuesto del sobreaprendizaje, ocurre principalmente cuando la estructura del modelo es demasiado simple para aprender la estructura de los datos. Por ejemplo, un modelo lineal con un conjunto de datos de entrenamiento muy disperso es un modelo propenso al

infraaprendizaje; el problema en cuestión es más complejo que el modelo, y por lo tanto las predicciones serán poco precisas. Algunas posibles soluciones serían:

- Escoger un modelo más complejo, con más parámetros.
- Alimentar al algoritmo con mejores características.
- Reducir la regularización del modelo.

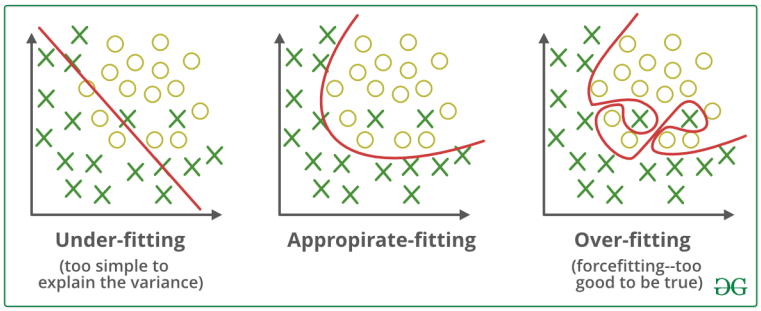

Figura 1.3: Ejemplos de sobreaprendizaje e infraaprendizaje

En la figura 1.3 podemos ver algunos ejemplos de sobreaprendizjae e infraaprendizaje.

Capítulo 2

Definición del problema

Antes de empezar a resolver un problema se debe comprender que se quiere resolver y para qué. Limitar un problema no es tarea sencilla, en la propia concepción del proyecto, se establece el entendimiento básico del problema, las personas que quieren una solución, la naturaleza de la solución deseada, así como la eficacia de la comunicación y colaboración preliminares entre los otros participantes y el equipo de software. Dada la naturaleza de este proyecto este apartado se puede dividir en:

- La **identificación del problema real** se define el problema desde el punto de vista del usuario final
- La **identificación del problema técnico** se define el problema desde el punto de vista del desarrollador del proyecto.

2.1. Definición del problema real

El problema real, planteado por los directores del proyecto, pertenecientes al grupo de investigación AYRNA, es el siguiente:

La implementación de tres algoritmos en`Python` de clasificación ordinal, previamente desarrollados para el *framework* ORCA escrito en MATLAB. Los algoritmos en cuestión son: `KDLOR`(*Kernel Discriminant Learning for Ordinal Regression*)[8], `SVC1V1`(*Support Vector Classifier using one-vs-one*) y `SVC1VA`(*Support Vector Classifier using one-vs-all*)[9]. Los tres clasificadores han de ser compatibles con `scikit-learn`.

2.2. Definición del problema técnico

A continuación, se procederá a la identificación del problema desde un punto de vista más técnico. Para ello, haremos uso de la metodología PDS (*Product Design Specificatión*), que nos permite hacer un análisis de los principales requisitos, restricciones y especificaciones que debe cumplir nuestro proyecto.

2.2.1. Funcionamiento

El objetivo de este proyecto es la adaptación de los 3 algoritmos previamente implementados en el *framework* ORCA, al *framework* ORCA-`Python`. Debido a que se trata de una ampliación de un Trabajo de Fin de Grado preexistente, hay ciertos elementos como la interfaz o los datos de salida que no será necesario implementar, sin embargo, esto también obliga a seguir unas ciertas directrices a la hora de usar el *framework* previamente implementado.A continuación se explicaran dichas directrices:

- **Base de datos**: el usuario proporcionará la base de datos que puede o no estar particionada, que se dividirá en train y test. El formato de los ficheros será CSV. Todas las bases de datos se deben encontrar en el directorio raíz indicado en el fichero de configuración.
- **Fichero de configuración**: el usuario introducirá una serie de instrucciones para la ejecución del experimento, tales como las bases de datos a utilizar,las métricas de evaluación, los clasificadores a utilizar, así como los valores o conjunto de valores que tendrán los parámetros de dichos clasificadores, etc.
- **Ejecución**: se ejecutaran los experimentos utilizando para ello, los ficheros de configuración previamente mencionados. Dichos experimentos proporcionarán unas métricas de evaluación.
- **Almacenamiento de resultados**: los resultados de los experimentos se almacenarán en la carpeta seleccionada por el usuario.

2.2.2. Entorno

El entorno de programación que se va a utilizar para el desarrollo de este proyecto será Ubuntu 20.04.6 LTS, aunque para comprobar su correcto comportamiento multiplataforma también se utilizará Windows 10, siendo las versiones anteriores de dicho *framework* compatible con ambos sistemas

operativos.

Los usuarios finales a los que va dirigido, deberán tener una serie de conocimientos para la correcta ejecución de los experimentos, como la creación de las reglas para los ficheros de configuración y el uso de la linea de comandos o *bash*.

2.2.3. Vida esperada

Estimar la vida útil del software en el caso que nos ocupa no es algo sencillo de calcular, dado que el campo de investigación se encuentra en constante evolución. Sumado a la propia naturaleza del problema, dependerá en gran medida de cuanto tiempo transcurra en encontrarse mejores soluciones a los problemas de clasificación para los que fueron diseñados dichos algoritmos. Además, a eso hay que sumarle que se trata de un *framework*, que todavía se encuentra en desarrollo, siendo solo unos pocos algoritmos los que se han adaptado a `Python` de los implementados en el *framework* `ORCA`. Por estos motivos, a priori el tiempo de vida del producto es incierto.

2.2.4. Ciclo de mantenimiento

El ciclo de mantenimiento del producto consistirá en todas aquellas mejoras a los algoritmos ya presentes en el *framework*, como la obtención de posibles mejores resultados debido al uso de nuevas métricas u optimizaciones de los algoritmos ya desarrollados. Además como se mencionó en el apartado todas las futuras implementaciones de nuevos algoritmos se incluirían en el ciclo de mantenimiento.

2.2.5. Competencia

WEKA (Waikato Environment for Knowledge Analysis) [10] proporciona implementaciones de algoritmos de aprendizaje que puedes aplicar fácilmente a tu base de datos. También incluye una variedad de herramientas que transforman las bases de datos, como los algoritmos de discretación y muestreo. Puedes preprocesar base de datos, alimentar esquemas de aprendizaje, y analizar el clasificador resultante y su rendimiento.

El *workbench* incluye métodos para los principales problemas de la minería de datos como son la regresión, la clasificación, *clustering*, asociación de reglas de minado y selección de atributos.

Otro de sus competidores sería `ORCA`, *framework* predecesor de este proyecto implementado en `MATLAB` del cual hablaremos en más profundidad en la Sección 4.4.

Por ultimo, tenemos a `mord`[11] que es una colección de algoritmos de regresión ordinal escrito en `Python`, siguiendo una `API` compatible con `scikit-learn`. El paquete incluye diferentes modelos adecuados para las tareas de regresión ordinal y los clasifica en modelos basados en umbrales, basados en regresión y basado en clasificación.

2.2.6. Aspecto externo

Este proyecto será presentado mediante la plataforma Moodle de la UCO, a petición de la Escuela Politécnica Superior en sus directrices sobre la entrega de proyectos de final de grado.

Se incluirá lo siguiente: en primer lugar la aplicación desarrollada, además también se incluirá el Manual técnico y de Usuario, redactado de forma concisa y clara, que tratará de explicar al usuario final de la aplicación su utilización de forma detallada.

El código de la aplicación se encontrará totalmente disponible en un repositorio de la plataforma `Github`, que en conjunto con la herramienta `Git`, para facilitar el desarrollo colaborativo y evitar posibles pérdidas de información.

2.2.7. Estandarización

Para el proceso de estandarización del proyecto, se seguirán los diferentes estándares enseñados a lo largo del Grado de Ingeniería Informática, en cuanto a la programación orientada a objetos se refiere.

En cuanto al lenguaje de programación, se seguirá la guía de estilo `PEP8` dado que utilizaremos `Python` para el proyecto.

- Los nombres de los módulos y paquetes deberían ser breves y en minúscula. En el caso de los módulos los guiones bajos pueden ser usados si mejora la lectura, no obstante, en el caso de los paquetes se desaconseja el uso de los mismos.
- El nombre de las clases debería normalmente utilizar la convención de *CapWords*.

- El nombre de las funciones y variables deberían escribirse en minúscula, pudiéndose usar el uso de guiones bajos para separar las palabras si mejora la lectura.
- El nombre de las variables de instancia y los métodos deberían escribirse en minúscula, permitiéndose el uso de guiones bajos para mejorar la lectura. No obstante, solamente para los métodos privados y las variables de instancia se permite que empiecen con guión bajo.
- Los nombres de las constantes deberían ir en mayúscula, utilizando los guiones bajos para separar las palabras.
- El límite máximo de caracteres de todas las lineas es 79, 72 para los comentarios

2.2.8. Calidad y fiabilidad

La calidad del trabajo será supervisada tanto por el autor del mismo, como por los directores que supervisarán y asesorarán para asegurar dicha calidad.

En cuanto a la fiabilidad, se tendrá presenta una serie de pruebas que aseguren el correcto funcionamiento del flujo de ejecución y en caso contrario se mostrará por la terminal un mensaje de error que explique de forma concisa y clara , el motivo del error.

2.2.9. Programa de tareas

El desarrollo de un software sigue, de forma general, las siguientes fases:

- **Fase de preparación.** Durante esta primera fase será necesario realizar un estudio teórico del problema, en el que se identifica los principios generales y básico para abordarlo. En este caso, se tendrá que identificar y estudiar el problema, que es el desarrollo de tres algoritmos de clasificación ordinal para el *framework* `ORCA-Python`.
  Para dos de los algoritmos (SVC1V1 y SVC1VA) será necesario tener conocimientos de `scikit-learn` ya que se implementarán usando para ello este paquete de `Python` en su totalidad, asegurando la compatibilidad con el *framework* previamente diseñado. Por último, en el caso del clasificador `KDLOR` se trata de una migración, por lo que será necesario conocer no solo el funcionamiento de la librería de ORCA-`Python` sino también de su precursora, `ORCA`, para integrar dicho método.

- **Fase de diseño.** Una vez analizados y estudiados los requisitos previos del sistema, se procederá a su diseño para su posterior codificación.
- **Fase de implementación.** A continuación, tendrá lugar la fase de implementación, en ella se codificarán los algoritmos `SVC1V1`, `SVC1VA` y `KDLOR`, con las pertinentes modificaciones para que sean integrados en el *framework* y garantizar su compatibilidad con el mismo.
- **Fase de pruebas.** Durante esta fase se verificará el correcto funcionamiento de nuestro programa. Ocurrirá de forma paralela a la fase de implementación, debido a que es más fácil localizar y corregir errores durante dicha fase, evitando así resultados no deseados.
- **Fase de aplicación.** En esta fase se compararán los resultados obtenidos por el clasificador desarrollado, frente a los resultados obtenidos por los algoritmos de regresión ordinal, dichos resultados son extraídas del trabajo del artículo de investigación aportado por el grupo `AYRNA`.
- **Fase de documentación.** En esta fase ocurre la realización de esta memoria, así como la del manual de usuario. Tendrá lugar durante la realización de todo el proyecto.

2.2.10. Pruebas

En la fase de pruebas se comprobará el correcto funcionamiento del software desarrollado. Para ello se realizarán pruebas a nivel de módulo para comprobar el correcto funcionamiento de los algoritmos implementados. Además se debe comprobar su correcta integración con el *framework* ORCA-`Python`. Una vez comprobado que los clasificadores, funcionan correctamente se procederá a analizar los resultados para verificar la calidad del producto comparando los resultados con su predecesor ORCA. Y por último se procederá a realizar una serie de pruebas para asegurar que funcionan en diferentes plataformas.

2.2.11. Seguridad

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

Además, como se ejecutará localmente y la aplicación no manejará datos que violen la privacidad de los usuarios ni de terceros, la privacidad no será un requerimiento.

Capítulo 3

Objetivos

El objetivo principal de este proyecto es desarrollar una biblioteca escrita en `Python` que implemente los algoritmos de clasificación ordinal propuestos. Para ello, se toma como base el *framework* de `ORCA-Python`, el cual consiste en una adaptación y ampliación del *framework* `ORCA` que esta escrito en `MATLAB/Octave`. `ORCA-Python` es un *framework* que incluye implementaciones previas realizadas por otros compañeros a partir de Trabajos Fin de Grado.

Estos algoritmos ya se encuentran implementados en el *framework* `ORCA`, desarrollado por el grupo `AYRNA`, por lo que uno de los objetivos será profundizar en estos algoritmos de clasificación ordinal para su posterior adaptación e implementación en `ORCA-Python`.

A continuación, se citan los puntos que se van a abordar en este trabajo:

- Implementación del algoritmo de `KDLOR`: reformulación del aprendizaje discriminante que permite el cálculo del óptimo mapeo unidimensional de los datos, mediante el análisis de dos objetivos: la maximización de la distancias entre clases, y la disminución de la distancia intraclase. Para reformular el algoritmo de regresión ordinal, una restricción de orden sobre las clases contiguas es impuesta sobre el promedio de los patrones proyectados de cada clase.

- Implementación del algoritmo de `SVC1V1`: máquina de vectores soporte nominal que realiza la formulación OneVsOne. Es considerado como un enfoque ingenuo para los problemas de regresión ordinal, ya que ignora el orden de la información

- Implementación del algoritmo de `SVC1VA`: máquina de vectores soporte nominal que realiza la formulación OneVsAll. Es considerado como un

enfoque ingenuo para los problemas de regresión ordinal,ya que ignora el orden de la información. Este método construye K clasificadores, uno por cada clase. Ese clasificador se entrena tomando como etiquetas positivas las muestras pertenecientes a esa clase y como negativas las del resto de clases.

- Comprobación del correcto funcionamiento de cada algoritmo a través de diversas pruebas.

3.1. Objetivos de formación

- Análisis del *framework* `ORCA`, así como del lenguaje `MATLAB`, para obtener una mejor comprensión del funcionamiento de dicho *framework* y el código que posteriormente se va a migrar a `ORCA-Python`.
- Estudio teórico de los métodos `KDLOR`, `SVC1V1` y `SVC1VA`.
- Análisis y comprensión del *framework* `ORCA-Python`
- Estudio del lenguaje de programación `Python`, así como de las bibliotecas pertenecientes a este y que se usarán en este proyecto, como son, `numpy`[12], `pandas`[13], `scikit-learn`[6], `scipy`[14], `sacred`[7] y `cvxopt`[15].

3.2. Objetivos operacionales

A continuación, se explicarán los objetivos operacionales los cuales son:

- La implementación y adaptación de los algoritmos `SVC1V1`, `SVC1VA` y `KDLOR` del *framework* `ORCA` a `ORCA-Python`.
- La comprobación del correcto funcionamiento de dichos algoritmos mediante una serie de archivos de pruebas.

Capítulo 4

Antecedentes

4.1. Regresión ordinal

Como se explico anteriormente la regresión ordinal es un tipo de aprendizaje supervisado en el que las etiquetas muestran un orden natural entre sí. Esto hace que el error cometido al clasificar no tenga el mismo peso. En el apartado se decía que un servicio clasificado como deficiente siendo excelente, no tendría la misma penalización que si se cometiera el error de clasificarlo como deficiente o decente, esto es porque contra más alejado este de la etiqueta correcta el error tendrá un mayor peso y por lo tanto la penalización será mayor.

Una vez hecho la introducción a lo que es la regresión ordinal se abordará el problema de forma más teórica. El problema de regresión ordinal consiste en predecir una etiqueta y de un vector de entrada $\mathbf{x}$, donde $x \in X \subseteq \mathbf{R}^K$ e $y \in \gamma = \{C_1, C_2, ..., C_Q\}$, en este caso x es un patrón perteneciente al espacio de entrada K-dimensional e y es una etiqueta perteneciente a un espacio Q conformado por diferentes etiquetas. Estas etiquetas forman categorías o grupos de patrones, tal que el objetivo es encontrar una regla de clasificación o una función que $r : X \to \gamma$ que pueda predecir las categorías de nuevos patrones, dado un conjunto de entrenamiento de N puntos, $D = \{(x_i, y_i), i = 1, ..., N\}$. Dado que se trata de un problema de regresión ordinal hay que considerar que las etiquetas tienen un orden natural, $C_1 \prec C_2 \prec ... \prec C_Q$, donde $\prec$ es un orden de relación. Muchos algoritmos de regresión ordinal y métricas consideran el rango de la etiqueta la posición de la misma en la escala ordinal, la cual viene dada por la función $\mathcal{O}(.)$, tal que $\mathcal{O}(C_q) = q, q = 1, ..., Q$. La suposición de un orden entre las etiquetas de las clases hace

que dos elementos diferentes γ puedan ser comparados usando el operador de relación $\prec$, mientras que en el caso de la clasificación nominal no sería posible.

4.2. Taxonomía

En esta sección se propondrá una taxonomía de la regresión ordinal que está basada en la del artículo de investigación del grupo `AYRNA` [16].

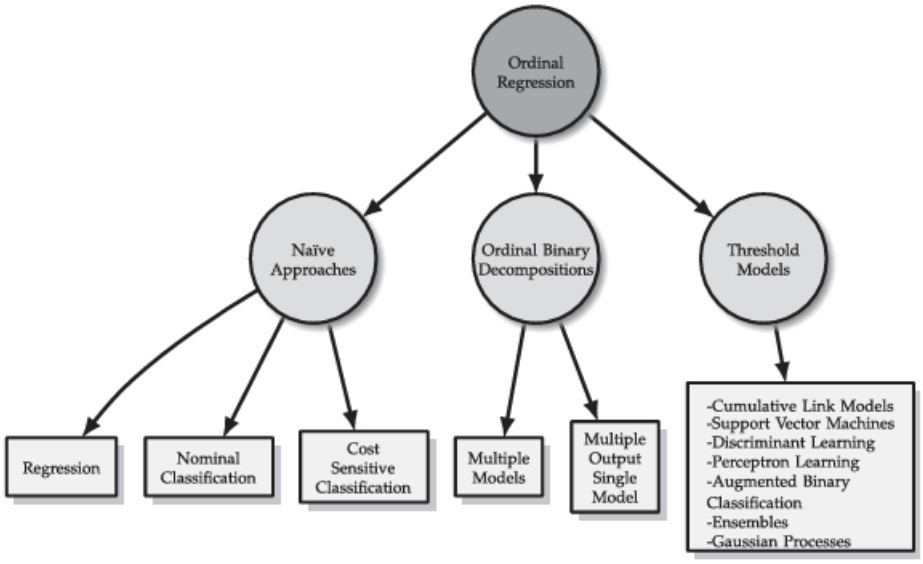

Figura 4.1: Taxonomía de regresión ordinal

Como se ve en la figura 4.1 la regresión ordinal se puede dividir en tres grupos en función el enfoque que se tome: aproximación ingenua, descomposición binaria ordinal y modelos de umbral. A continuación se verán con mas detalle cada una de ellas.

- **Aproximación ingenua.** Estos métodos tratan de resolver el problema usando para ello otros algoritmos de *machine learning* simplificando el problema en otros problemas estándar mediante la aceptación de una serie de conjeturas. Por ejemplo si se asignará a cada clase un valor real sería posible aplicar técnicas de regresión para resolver el problema, si en cambio se ignorará el orden de entre las clases se

podrían aplicar algoritmos de clasificación nominal y por el último método sería considerar el aprendizaje sensible al coste considerando aplicar diferentes tipos de coste por errores al clasificar. No obstante, estos enfoques tienen algunos problemas y es que por ejemplo en el caso de la regresión o la clasificación sensible al coste se necesita información a priori.

- **Descomposición ordinal binaria.** La idea principal de este método es que para resolver el problema este se dividirá en varios problemas binarios. La estimación se realizará por uno o múltiples modelos. Los principales problemas a los que se enfrenta este enfoque es que algunos se basan en la idea de diferentes modelos para resolver cada subproblema, mientras que otros se basan en un único modelo para resolver todos los subproblemas y que no solo hay que definir como se dividirá el problema sino que también es importante definir una regla que permita predecir patrones nuevos.

- **Modelos de umbral.** A menudo en el paradigma de la regresión es posible pensar que una variable continua no observada subyace la respuesta al problema, a esta variable se la conoce con el nombre de variable latente y los métodos basados en esta suposición se conocen como modelos de umbral. Este enfoque puede ser visto como una extensión de los modelos de regresión ingenua, la principal diferencia es que la distancia de las clases no esta definida a priori en el modelo umbral.

4.3. MATLAB

`MATLAB` [5] se trata de un lenguaje de programación de alto nivel diseñado para profesionales de ingeniería y ciencias que permite trabajar con las matrices y *arrays* directamente. Su nombre proviene de la abreviatura de *matrix laboratory* e incluye operaciones básicas, como la creación de variables, la indexación de *arrays*, operaciones aritméticas y tipos de datos. Entre sus principales características se encuentran que:

- Es un lenguaje de programación de alto nivel que esta basado en matrices

- Es un entorno de escritorio optimizado para la exploración iterativa, el diseño y la solución de problemas.

- Permite el uso de gráficas para visualizar los datos y herramientas para crear diagramas personalizados
- Aplicaciones para ajustar curvas, clasificar datos, analizar señales, ajustar sistemas de control, etc.
- Herramientas para crear aplicaciones con interfaces de usuario personalizadas
- Interfaces para otros lenguajes de programación como C/C++, Java, etc

4.4. ORCA

`ORCA` [2] (*Ordinal Regression and Classification Algorithms*) es un *framework* de `MATLAB` [5] que implementa e integra una amplía gama de métodos de regresión ordinal y de métricas de rendimiento. Fue desarrollado por el grupo de investigación AYRNA [16], que pertenece al Departamento de Informática y Análisis Numérico de la Universidad de Cordoba (UCO).

Las principales virtudes del *framework* es que simplifica la tareas de experimentación enormemente, permitiendo al usuario:

- Definir experimentos a partir de un simple fichero de configuración
- Ejecutar automáticamente diferentes particiones de datos
- Paralelizar las ejecuciones
- Generar reportes de una variedad de métricas de rendimiento
- Incluir nuevos algoritmos usando una interfaz intuitiva.

A continuación se verán estas características de manera más extensa.

La clase de los algoritmos define una `API` que incluye diferentes métodos tales como `fit` y `predict`, los cuales procesan las particiones de entrenamiento y test para unos hiperparámetros especificados en la configuración del clasificador seleccionado. Por lo que añadir un nuevo método es relativamente fácil, el usuario tan solo necesita añadir una nueva clase implementando los métodos `fit` y `predict`.

Todas las métricas incluyen un método `calculateMetric`, el cual puede ser ejecutado usando los verdaderos y las etiquetas predichas o la matriz de confusión.

`ORCA` [2] automatiza el proceso de ejecución para uno o varios métodos de un conjunto de base de datos. Ejecuta cada experimento entrenando un modelo con los datos de entrenamiento (incluye la selección del modelo a través de *cross-validación*), evaluando el error del test y creando reportes de rendimiento para todas las métricas.

Para la ejecución de dichos experimentos, `ORCA` [2] puede ser usado directamente interactuando con la `API` o a través de los archivos de configuración en formato `INI`.

A continuación se explicaran con más detalles los algoritmos de regresión ordinal que se van a desarrollar en `ORCA-Python` [3, 4].

El aprendizaje discriminante se ha reformulado para abordar regresión ordinal. Esta metodología es conocida como `KDLOR` [8] (*Kernel Discriminant Learning for Ordinal Regression*). El análisis discriminante no suele ser considerado como una técnica de clasificación por si misma, sino más bien como una reducción de la dimensionalidad supervisada. En general, permite el calculo del óptimo mapeo unidimensional de los datos, mediante el análisis de dos objetivos: la maximización de la distancia entre clases, y la disminución de la distancia intraclase, usando para ello las matrices de varianza-covarianza y el coeficiente de Rayleigh. Para tener en cuenta los problemas de regresión ordinal, una restricción de orden sobre las clases contiguas es impuesta sobre el promedio de los patrones proyectados de cada clase, lo que lleva al algoritmo a ordenar los patrones proyectados según sus etiquetas. Esto preservará la información ordinal y evitará errores graves de clasificación.

Dentro de los enfoques ingenuos tenemos dos formulaciones *OneVsOne* y *OneVsAll* (`SVC1V1` y `SVC1VA`) [9] que son las principales formulaciones en SVM cuando se trabaja con problemas multiclase. Aunque estos métodos consideran la descomposición binaria, han sido incluidos dentro de los grupos nominales de clasificación debido a que no se toma en cuenta el orden.

El método *OneVsOne* construye $K(K-1)/2$ clasificadores donde cada uno es entrenado con datos de dos clases. Se pueden considerar diferentes

métodos para la fase de generalización una vez que se han construido los $K(K-1)/2$ clasificadores. Un ejemplo de estos métodos sería el siguiente: si $\sin((w^{ij})^T\phi(x) + b^{ij})$ dice que x pertenece a la clase i entonces se aumenta en un voto la clase i, en caso contrario, se aumenta en un voto la clase j. X pertenecerá a la clase con más votos. El enfoque de votación descrito es llamado como la estrategia de *Max Win*.

El método *OneVsAll* construye K clasificadores, uno por cada clase. Ese clasificador se entrena tomando como etiquetas positivas las muestras pertenecientes a esa clase y como negativas las del resto de clases. Después de resolver el problema, tendremos k funciones de decisión y x pertenecerá a la función de decisión con el mayor valor:

4.5. Python

`Python` es un lenguaje de programación de propósito general muy poderoso y flexible, a la vez que sencillo y fácil de aprender, que fue creado a principios de los noventa por Guido Van Rossum. Es un lenguaje de alto nivel, que permite procesar fácilmente todo tipo de estructuras de datos, tanto numéricas como de texto.

4.6. ORCA-Python

`ORCA-Python` [3, 4] es un *framework* escrito en `Python` integrado con los módulos de `Scikit-Learn` [6] y `sacred` [7], que busca automatizar los experimentos de *machine learning* mediante ficheros de configuración fáciles de entender. Fue desarrollado como Trabajo Fin de Grado del alumno Iván Bonaque Muñoz y posteriormente ampliado por el alumno Ángel Heredia Pérez y se trata de una ampliación y adaptación del *framework* `ORCA` [2] al lenguaje de programación `Python`.

Este *framework* es compatible con cualquier algoritmo que se encuentre implementado en `Scikit-Learn` o bien creado por el propio usuario siempre que siga las reglas de compatibilidad de esta biblioteca. Si bien, como ya se ha mencionado, este está escrito en `Python`, mientras que algunos algoritmos ya implementados en `ORCA-Python`, como es el caso de `REDSVM` o `SVOREX`, están implementados en un lenguaje de programación diferente, `C++` y `C`, respectivamente.

La correcta ejecución del *framework* requiere de la instalación de las siguientes dependencias de `Python`:

- `NumPy`
- `Pandas`
- `Scikit-Learn`
- `Scipy`
- `Sacred`
- `cvxopt`

4.6.1. Numpy

`Numpy` [12] es una biblioteca para `Python` que facilita el trabajo con *arrays*. Esta biblioteca permite declarar *arrays* con distintas dimensiones capaces de albergar gran cantidad de datos del mismo tipo y relacionados entre sí. Además, provee numerosos métodos para manipular los *arrays*; y para acceder a la información y procesarla de forma muy eficiente.

4.6.2. Pandas

`Pandas` [13] es una biblioteca para `Python`, que proporciona una serie de estructuras de datos y operaciones para manipulación y el análisis de los mismos. `Pandas` permite importar varios formatos de archivos (`csv, excel, sql, json, parquet, etc`).

4.6.3. Scikit-learn

`Scikit-Learn` [6] es una biblioteca de software libre para `Python`. Cuenta con varios algoritmos como máquina de vectores soporte, *random forest* y *K-neighbours*. También admite bibliotecas numéricas y científicas como `NumPy` y `SciPy`.

4.6.4. Scipy

`SciPy` [14] es un software de código abierto que nació a partir de la colección original de Travis Oliphant y que consistía en módulos de extensión para `Python`. La biblioteca de `SciPy` depende de `Numpy`, proporcionando una manipulación de vectores N-dimensionales de una forma cómoda y rápida.

Además contiene diversos módulos para optimización, álgebra lineal, integración y otras tareas relacionadas con la ciencia e ingeniería.

4.6.5. Sacred

`Sacred` [7] es una herramienta que permite configurar, organizar, registrar y reproducir experimentos computacionales. Está diseñado para introducir la mínima sobrecarga, y, al mismo tiempo, fomentar la modularidad y facilitar la configuración de los experimentos. La capacidad para hacer experimentos.

La capacidad para hacer experimentos configurables es el corazón de `Sacred` y permite, entre otros:

- Realizar un seguimiento de todos los parámetros de tu experimento.
- Guardar la versión del código que se ha lanzado, para evitar problemas de saber que código has utilizado para un determinado experimento.
- Ejecutar tus experimentos con diferentes configuraciones.
- Guardar configuración para ejecuciones individuales en ficheros o en una base de datos.
- Reproducir tus resultados.

4.6.6. Cvxopt

`Cvxopt` [15] es un paquete de software para la optimización convexa basado en `Python`. Se puede usar con el interprete interactivo de `Python`, en la línea de comandos mediante el uso scripts o integrado en otro software a través de módulos de extensión de `Python`. Su principal propósito es facilitar el desarrollo de software para aplicaciones de optimización convexas utilizando para ello la extensa biblioteca estándar de `Python` y las fortalezas de `Python` al ser un lenguaje de programación de alto nivel.

4.7. Métricas

Las métricas en clasificación son medidas utilizadas para evaluar el rendimiento de los modelos de aprendizaje automático. Estas métricas se utilizan para comparar diferentes modelos y seleccionar el que tenga un mejor rendimiento en base a las necesidades y objetivos del problema a resolver.

Normalmente para calcular las métricas se suele utilizar una herramienta que se muestra en la figura 4.2 llamada matriz de confusión que evalúa el rendimiento del modelo de aprendizaje.

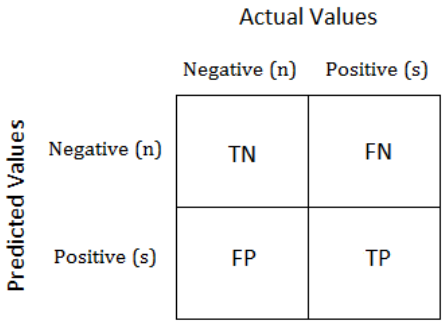

Figura 4.2: Matriz de confusión

- Verdaderos positivos (TP): número de casos positivos que el modelo ha clasificado correctamente.
- Falsos positivos (FP): número de casos negativos que el modelo ha clasificado incorrectamente como positivos.
- Verdaderos negativos (TN): número de casos negativos que el modelo ha clasificado correctamente.
- Falsos negativos (FN): número de casos positivos que el modelo ha clasificado incorrectamente como negativos.

Con la matriz de confusión se pueden calcular diferentes métricas, a continuación veremos algunas de ellas:

- *Sensitivity* = $\frac{TP}{TP+FN}$, es el ratio de verdaderos positivos.
- Specificity = $\frac{TN}{TN+FP}$, es el ratio de verdaderos negativos.
- `Precision` = $\frac{TP}{TP+FP}$, representa los verdaderos positivos de entre todos los casos que el modelo ha predicho como positivos
- `Accuracy` = $\frac{TN+TP}{TN+FP+TP+FN}$, los casos que el modelo ha clasificado correctamente

4.7.1. *Mean Absolute Error* (MAE)

El error medio absoluto (MAE) es la media de todos los errores absolutos de predicción, donde el error de predicción es la diferencia entre el valor real y el valor predicho.

MAE = $\frac{1}{N}\sum_{i=1}^{N}|\mathcal{O}(y_i) - \mathcal{O}(y_i^*)|$, donde y_i y y_i^* representan la clase real y la clase predicha respectivamente.

4.7.2. *Mean Zero-One Error* (MZE)

También se le conoce como el ratio de error del clasificador y es la medida complementaria de la *accuracy*:

MZE = 1-Acc

Capítulo 5

Restricciones

En este capítulo se expondrán todas las restricciones de diseño, que condicionarán las fases posteriores del proyecto. Estos se pueden dividir en dos grupos: los factores dato y factores estratégicos.

- **Factores dato.** Son todas aquellas restricciones que vienen impuestas por la propia naturaleza del problema, y por tanto no pueden modificarse durante su desarrollo y son de obligado cumplimiento. Suelen venir dadas por el cliente, en este caso el grupo de investigación `AYRNA`.
- **Factores estrátegicos.** Son restricciones que no vienen impuestas, elegidas por el desarrollador. Como por ejemplo la elección de una herramienta frente a otra véase, la elaboración de la documentación de este proyecto usando para eso la herramienta LaTeX`Texmaker`.

5.1. Factores dato

Como se ha comentado en el apartado anterior los factores dato son un grupo de restricciones que han de cumplirse durante el desarrollo del proyecto.

- **Restricciones hardware.** El alumno encargado de realizar el proyecto hará uso de su propio equipo informático.
- **Restricciones software.** Se deben implementar tres algoritmos de clasificación ordinal en `Python` para el *framework* `ORCA-Python` y deben ser compatibles tanto con el *framework* como con `scikit-learn`.

- **Restricciones temporales.** Estas restricciones vendrán establecidas por la fecha límite de entrega, que tendrá lugar durante el presente curso académico 2023/24.
- **Restricciones humanas.** Dado que el proyecto es un Trabajo de Fin de Grado, estás limitaciones vendrán determinadas por los directores del mismo.
- **Restricciones económicas.** Al tratarse de un Trabajo de Fin de Grado, se priorizará el uso de herramientas de software libre.

5.2. Factores estratégicos

Los factores estratégicos son restricciones impuestas por el propio desarrollador y los directores del proyecto.

- Se integrará el programa con la biblioteca `scikit-learn` ya que los algoritmos de maquina de vectores soporte se implementarán usando para ello dichas bibliotecas.
- Se utilizará el sistema de composición LaTeXpara la generación de documentos de este trabajo. Tal y como se ha explicado previamente se utilizará `Texmaker` que es una aplicación gratuita multiplataforma que permite la generación de documentación en LaTeXpara diferentes sistemas operativos como linux o windows.
- Para los diagramas de clases se utilizará la herramienta *online* `GenMyModel`.
- Para el control de versiones se utilizará `Git` acompañada de `GitHub` para subir el código una vez terminado el proyecto.

Capítulo 6

Recursos

En esta sección se mencionaran los recursos utilizados por el autor del proyecto durante el desarrollo del mismo.

6.1. Recursos hardware

Para el desarrollo de este proyecto se ha utilizado el equipo personal del autor, que consta de las siguientes características:

- Procesador Intel (R) Core (TM) i7-8750H CPU @ 2.20 GHz
- Memoria RAM 16 GB DDR4 x2
- Controlador gráfico Geforce GTX 1060 6GB GDDR5
- Discor duro HGST Travelstar 7K1000 1 TB + SSD de 256 GB

6.2. Recursos software

- Sistema Operativo (SO): Se ha trabajado en Ubuntu y Windows para verificar el correcto funcionamiento del software desarrollado.
- Lenguaje de programación `Python`.
- Documentación: se ha utilizado `Textmaker` y `basic-miktex` para generar la documentación en LaTeX.
- Visual Studio Code.

6.3. Recursos humanos

- **Cristian Torres Pérez de Gracia:** Alumno del grado de Ingeniería Informática con mención en computación, en la Escuela Politécnica Superior de la Universidad de Córdoba (EPSC). Será el encargado del diseño, implementación y documentación del Trabajo de Fin de Grado.
- **David Guijo Rubio:** Investigador postdoctoral en la Universidad de Córdoba del Dpto. Informática y Análisis Numérico y miembro investigador del grupo AYRNA y encargado de dar soporte técnico y teórico para las tareas de análisis, diseño y programación.
- **Pedro Antonio Gutiérrez Peña:** Profesor Catedrático de la Universidad de Córdoba del Dpto. Informática y Análisis Numérico y miembro investigador del grupo AYRNA y encargado de dar soporte técnico y teórico para las tareas de análisis, diseño y programación.

Parte II

Requisitos del Sistema

Capítulo 7

Especificación de requisitos

7.1. Introducción

7.2. Participantes del trabajo

Participante	Pedro Antonio Gutiérrez Peña
Organización	AYRNA
Rol	Director
Desarrollador	No
Cliente	Si
Usuario	Si
Comentarios	Catedrático de la Universidad de Córdoba del Dpto. Informática y Análisis Numérico y miembro investigador del grupo AYRNA.

Cuadro 7.1: Director Pedro Antonio Gutiérrez Peña

Participante	David Guijo Rubio
Organización	AYRNA
Rol	Director
Desarrollador	No
Cliente	Si
Usuario	Si
Comentarios	Profesor Titular de la Universidad de Córdoba del Dpto. Informática y Análisis Numérico y miembro investigador del grupo AYRNA.

Cuadro 7.2: Director David Guijo Rubio

Participante	Cristian Torres Pérez de Gracia
Organización	EPS UCO
Rol	Alumno desarrollador
Desarrollador	Si
Cliente	No
Usuario	No
Comentarios	Alumno de Grado en Ingeniería Informática con mención en computación de la Escuela Politécnica Superior de la Universidad de Cordoba.

Cuadro 7.3: Alumno Cristian Torres Pérez de Gracia

7.3. Catálogo de objetivos del sistema

OBJ-001	Desarrollo de una biblioteca escrita en `Python`
Versión	1.0
Autor	Cristian Torres Pérez de Gracia
Fuentes	Pedro Antonio Gutiérrez Peña David Guijo Rubio
Descripción	Desarrollo de una biblioteca escrita en `Python` que implemente los algoritmos de clasificación ordinal propuestos.Se implementarán y adaptaran los algoritmos `KDLOR`, `SVC1V1` y `SVC1VA`
Subojetivos	OBJ-002 Implementación y adaptación del clasificador `SVC1V1` OBJ-003 Implementación y adaptación del clasificador `SVC1VA` OBJ-004 Implementación y adaptación `KDLOR` OBJ-005 Facilidad de uso
Comentarios	Ninguno

Cuadro 7.4: Desarrollo de la biblioteca para el *framework* `ORCA-Python`

OBJ-002	Implementación y adaptación del clasificador `SVC1V1`
Versión	1.0
Autor	Cristian Torres Pérez de Gracia
Fuentes	Pedro Antonio Gutiérrez Peña David Guijo Rubio
Descripción	Implementación del clasificador `SVC1V1` usando el lenguaje de programación `Python` y la librería de scikit-learn para permitir su ejecución desde el *framework* `ORCA-Python`
Subojetivos	Ninguno
Comentarios	Ninguno

Cuadro 7.5: Implementación del código del clasificador `SVC1V1` para el *framework* `ORCA-Python`

OBJ-003	Implementación y adaptación del clasificador `SVC1VA`
Versión	1.0
Autor	Cristian Torres Pérez de Gracia
Fuentes	Pedro Antonio Gutiérrez Peña David Guijo Rubio
Descripción	Implementación del clasificador `SVC1VA` usando el lenguaje de programación `Python` y la librería de scikit-learn para permitir su ejecución desde el *framework* `ORCA-Python`
Subojetivos	Ninguno
Comentarios	Ninguno

Cuadro 7.6: Implementación del código del clasificador `SVC1VA` para el *framework* `ORCA-Python`

OBJ-004	Implementación y adaptación `KDLOR`
Versión	1.0
Autor	Cristian Torres Pérez de Gracia
Fuentes	Pedro Antonio Gutiérrez Peña David Guijo Rubio
Descripción	Implementación del clasificador `KDLOR` usando el lenguaje de programación `Python`, con los cambios pertinentes para permitir su ejecución desde el *framework* `ORCA-Python`
Subojetivos	Ninguno
Comentarios	El código deberá ser compatible con la librería scikilearn

Cuadro 7.7: Implementación y adaptación `KDLOR`

OBJ-005	Facilidad de uso
Versión	1.0
Autor	Cristian Torres Pérez de Gracia
Fuentes	Pedro Antonio Gutiérrez Peña David Guijo Rubio
Descripción	Facilitar al usuario final en la medida de lo posible el uso de los algoritmos implementados para el *framework* `ORCA-Python`
Subojetivos	Ninguno
Comentarios	Ninguno

Cuadro 7.8: Facilidad de uso

7.4. Catálogo de requisitos del sistema

7.4.1. Requisitos de información

IRQ-001	Configuración de parámetros de `SVC1V1`
Versión	1.0
Autor	Cristian Torres Pérez de Gracia
Fuentes	Pedro Antonio Gutiérrez Peña David Guijo Rubio
Dependencias	OBJ-002 Implementación y adaptación del clasificador de `SVC1V1`
Descripción	Los parámetros del algoritmo son: **kernel_type.** Tipo de kernel utilizado por el clasificador **degree.** Parámetro utilizado en la función kernel. **gamma.** Parámetro utilizado en la función kernel. **coef0.** Parámetro utilizado en la función kernel. **cost.** Parámetro de regularización **cachesize.** Tamaño de la memoria cache **epsilon.** Parámetro de regularización. **shrinking.** Especifica si se utilizará la heurística *shrinking* o no.
Comentarios	Ninguno

Cuadro 7.9: Requisito de información de la configuración de los parámetros de `SVC1V1`

IRQ-002	Configuración de parámetros de `SVC1VA`
Versión	1.0
Autor	Cristian Torres Pérez de Gracia
Fuentes	Pedro Antonio Gutiérrez Peña David Guijo Rubio
Dependencias	OBJ-003 Implementación y adaptación del clasificador de `SVC1VA`
Descripción	Los parámetros del algoritmo son: **kernel_type.** Tipo de kernel utilizado por el clasificador **degree.** Parámetro utilizado en la función kernel. **gamma.** Parámetro utilizado en la función kernel. **coef0.** Parámetro utilizado en la función kernel. **cost.** Parámetro de regularización **cachesize.** Tamaño de la memoria cache **epsilon.** Parámetro de regularización. **shrinking.** Especifica si se utilizará la heurística *shrinking* o no.
Comentarios	Ninguno

Cuadro 7.10: Requisito de información de la configuración de los parámetros de `SVC1VA`

IRQ-003	Configuración de los parámetros de `KDLOR`
Versión	1.0
Autor	Cristian Torres Pérez de Gracia
Fuentes	Pedro Antonio Gutiérrez Peña David Guijo Rubio
Dependencias	OBJ-004 Implementación y adaptación del clasificador de `KDLOR`
Descripción	**kernel_type.** Tipo de kernel utilizado por el clasificador **cost.** Parámetro de regularización **u.** Parámetro de regularización **k.** Parámetro del kernel
Comentarios	Ninguno

Cuadro 7.11: Requisito de información de la configuración de los parámetros de `KDLOR`

7.4.2. Requisitos funcionales

FRQ-001	Entrenamiento del clasificador `SVC1V1`
Versión	1.0
Autor	Cristian Torres Pérez de Gracia
Fuentes	Pedro Antonio Gutiérrez Peña David Guijo Rubio
Objetivos Asociados	OBJ-001 Desarrollo de una biblioteca escrita en `Python` OBJ-002 Implementación y adaptación del clasificador `SVC1V1`
Requisitos Asociados	Ninguno
Descripción	Se utilizará el *framework* `ORCA-Python` para entrenar el modelo de `SVC1V1`, utilizando para ello una clase local donde se almacenará dicho modelo con el fin de predecir resultados en futuras iteraciones. Para dicho entrenamiento es necesario proporcionar los parámetros con los que se va a ejecutar el algoritmo y las bases de datos utilizadas para el mismo.
Comentarios	La clase local deberá cumplir con los estándares especificados en la documentación de `scikit-learn.`

Cuadro 7.12: Requisito funcional entrenamiento del clasificador `SVC1V1`

FRQ-002	Fase de predicción del clasificador `SVC1V1`
Versión	1.0
Autor	Cristian Torres Pérez de Gracia
Fuentes	Pedro Antonio Gutiérrez Peña David Guijo Rubio
Objetivos Asociados	OBJ-001 Desarrollo de una biblioteca escrita en `Python` OBJ-002 Implementación y adaptación del clasificador `SVC1V1`
Requisitos Asociados	FRQ-001 Entrenamiento del clasificador `SVC1V1`
Descripción	Una vez terminada la fase de entrenamiento se cargará de la clase local el modelo previamente entrenado y se procederá a clasificar los patrones del dataset de test.
Comentarios	La clase local deberá cumplir con los estándares especificados en la documentación de `scikit-learn`.

Cuadro 7.13: Requisito funcional predicción del clasificador `SVC1V1`

FRQ-003	Entrenamiento del clasificador `SVC1VA`
Versión	1.0
Autor	Cristian Torres Pérez de Gracia
Fuentes	Pedro Antonio Gutiérrez Peña David Guijo Rubio
Objetivos Asociados	OBJ-001 Desarrollo de una biblioteca escrita en `Python` OBJ-003 Implementación y adaptación del clasificador `SVC1VA`
Requisitos Asociados	Ninguno
Descripción	Se utilizará el *framework* `ORCA-Python` para entrenar el modelo de `SVC1VA`, utilizando para ello una clase local donde se almacenará dicho modelo con el fin de predecir resultados en futuras iteraciones. Para dicho entrenamiento es necesario proporcionar los parámetros con los que se va a ejecutar el algoritmo y las bases de datos utilizadas para el mismo.
Comentarios	La clase local deberá cumplir con los estándares especificados en la documentación de `scikit-learn`.

Cuadro 7.14: Requisito funcional entrenamiento del clasificador `SVC1VA`

FRQ-004	Fase de predicción del clasificador `SVC1VA`
Versión	1.0
Autor	Cristian Torres Pérez de Gracia
Fuentes	Pedro Antonio Gutiérrez Peña David Guijo Rubio
Objetivos Asociados	OBJ-001 Desarrollo de una biblioteca escrita en `Python` OBJ-003 Implementación y adaptación del clasificador `SVC1VA`
Requisitos Asociados	FRQ-003 Entrenamiento del clasificador `SVC1VA`
Descripción	Una vez terminada la fase de entrenamiento se cargará de la clase local el modelo previamente entrenado y se procederá a clasificar los patrones del *dataset* de test.
Comentarios	La clase local deberá cumplir con los estándares especificados en la documentación de `scikit-learn`.

Cuadro 7.15: Requisito funcional predicción del clasificador `SVC1VA`

FRQ-005	Entrenamiento del clasificador `KDLOR`
Versión	1.0
Autor	Cristian Torres Pérez de Gracia
Fuentes	Pedro Antonio Gutiérrez Peña David Guijo Rubio
Objetivos Asociados	OBJ-001 Desarrollo de una biblioteca escrita en `Python` OBJ-004 Implementación y adaptación `KDLOR`
Requisitos Asociados	Ninguno
Descripción	Se utilizará el *framework* `ORCA-Python` para entrenar el modelo de `KDLOR`, utilizando para ello una clase local donde se almacenará dicho modelo con el fin de predecir resultados en futuras iteraciones. Para dicho entrenamiento es necesario proporcionar los parámetros con los que se va a ejecutar el algoritmo y las bases de datos utilizadas para el mismo.
Comentarios	La clase local deberá cumplir con los estándares especificados en la documentación de `scikit-learn.`

Cuadro 7.16: Requisito funcional entrenamiento del clasificador `KDLOR`

FRQ-006	Fase de predicción del clasificador `KDLOR`
Versión	1.0
Autor	Cristian Torres Pérez de Gracia
Fuentes	Pedro Antonio Gutiérrez Peña David Guijo Rubio
Objetivos Asociados	OBJ-001 Desarrollo de una biblioteca escrita en `Python` OBJ-004 Implementación y adaptación `KDLOR`
Requisitos Asociados	FRQ-005 Entrenamiento del clasificador `KDLOR`
Descripción	Una vez terminada la fase de entrenamiento se cargará de la clase local el modelo previamente entrenado y se procederá a clasificar los patrones del dataset de test.
Comentarios	La clase local deberá cumplir con los estándares especificados en la documentación de `scikit-learn.`

Cuadro 7.17: Requisito funcional predicción del clasificador `KDLOR`

7.4.3. Requisitos no funcionales

NFR-001	Implementación y adaptación del clasificador `SVC1V1` a `Python`
Versión	1.0
Autor	Cristian Torres Pérez de Gracia
Fuentes	Pedro Antonio Gutiérrez Peña David Guijo Rubio
Dependencias	OBJ-002 Implementación y adaptación del clasificador `SVC1V1`
Descripción	El clasificador `SVC1V1` se desarrollará y adaptará al *framework* `ORCA-Python`, usando para ello el lenguaje de programación `Python`.
Comentarios	Ninguno

Cuadro 7.18: Requisito no funcional implementación y adaptación del clasificador `SVC1V1`

NFR-002	Compatibilidad del clasificador `SVC1V1` a `ORCA-Python`
Versión	1.0
Autor	Cristian Torres Pérez de Gracia
Fuentes	Pedro Antonio Gutiérrez Peña David Guijo Rubio
Dependencias	OBJ-002 Implementación y adaptación del clasificador `SVC1V1`
Descripción	El clasificador `SVC1V1` deberá adecuarse al correspondiente *framework*, respetando el diseño del mismo para su correcto funcionamiento.
Comentarios	El clasificador deberá ser compatible con `scikit-learn`.

Cuadro 7.19: Requisito no funcional compatibilidad del clasificador `SVC1V1`

NFR-003	Implementación y adaptación del clasificador `SVC1VA` a `Python`
Versión	1.0
Autor	Cristian Torres Pérez de Gracia
Fuentes	Pedro Antonio Gutiérrez Peña David Guijo Rubio
Dependencias	OBJ-003 Implementación y adaptación del clasificador `SVC1V1`
Descripción	El clasificador `SVC1VA` se desarrollará y adaptará al *framework* `ORCA-Python`, usando para ello el lenguaje de programación `Python`.
Comentarios	Ninguno

Cuadro 7.20: Requisito no funcional implementación y adaptación del clasificador `SVC1VA`

NFR-004	Compatibilidad del clasificador `SVC1VA` a `ORCA-Python`
Versión	1.0
Autor	Cristian Torres Pérez de Gracia
Fuentes	Pedro Antonio Gutiérrez Peña David Guijo Rubio
Dependencias	OBJ-003 Implementación y adaptación del clasificador `SVC1VA`
Descripción	El clasificador `SVC1VA` deberá adecuarse al correspondiente *framework*, respetando el diseño del mismo para su correcto funcionamiento.
Comentarios	El clasificador deberá ser compatible con `scikit-learn`.

Cuadro 7.21: Requisito no funcional compatibilidad del clasificador `SVC1VA`

NFR-005	Implementación y adaptación del clasificador `KDLOR` a `Python`
Versión	1.0
Autor	Cristian Torres Pérez de Gracia
Fuentes	Pedro Antonio Gutiérrez Peña David Guijo Rubio
Dependencias	OBJ-004 Implementación y adaptación del clasificador `SVC1V1`
Descripción	El clasificador `KDLOR` se desarrollará y adaptará al *framework* `ORCA-Python`, usando para ello el lenguaje de programación `Python`.
Comentarios	Ninguno

Cuadro 7.22: Requisito no funcional implementación y adaptación del clasificador `KDLOR`

NFR-006	Compatibilidad del clasificador `KDLOR` a `ORCA-Python`
Versión	1.0
Autor	Cristian Torres Pérez de Gracia
Fuentes	Pedro Antonio Gutiérrez Peña David Guijo Rubio
Dependencias	OBJ-004 Implementación y adaptación del clasificador `SVC1VA`
Descripción	El clasificador `KDLOR` deberá adecuarse al correspondiente *framework*, respetando el diseño del mismo para su correcto funcionamiento.
Comentarios	El clasificador deberá ser compatible con `scikit-learn`.

Cuadro 7.23: Requisito no funcional compatibilidad del clasificador `KDLOR`

NFR-007	Usabilidad
Versión	1.0
Autor	Cristian Torres Pérez de Gracia
Fuentes	Pedro Antonio Gutiérrez Peña David Guijo Rubio
Dependencias	OBJ-005 Facilidad de Uso
Descripción	Se debe facilitar el uso y entendimiento del algoritmo al usuario final a través de la documentación aportada.
Comentarios	Ninguno

Cuadro 7.24: Requisito no funcional usabilidad

NFR-008	Tolerancia a fallos y optimización
Versión	1.0
Autor	Cristian Torres Pérez de Gracia
Fuentes	Pedro Antonio Gutiérrez Peña David Guijo Rubio
Dependencias	OBJ-001 Desarrollo de una biblioteca escrita en `Python`
Descripción	Los clasificadores implementados en el *framework* `ORCA-Python`, deberán tolerar los errores, detectando y reportándolos de forma clara a través de la terminal.
Comentarios	Ninguno

Cuadro 7.25: Requisito no funcional tolerancia a fallos y optimización.

7.5. Matriz de rastreabilidad

TRM - 001	OBJ - 001	OBJ - 002	OBJ - 003	OBJ - 004	OBJ - 005
IRQ - 001		X			
IRQ - 002			X		
IRQ - 003				X	
FRQ - 001	X	X			
FRQ - 002	X	X			
FRQ - 003	X		X		
FRQ - 004	X		X		
FRQ - 005	X			X	
FRQ - 006	X			X	
NRF - 001		X			
NRF - 002		X			
NRF - 003			X		
NRF - 004			X		
NRF - 005				X	
NRF - 006				X	
NRF - 007					X
NRF - 008	X				

Cuadro 7.26: Matriz de rastreabilidad

Parte III

Análisis y Diseño del Sistema

Capítulo 8

Casos de uso

Un caso de uso narra una historia sobre cómo interactúa un usuario final (que tiene cierto números de roles posibles) con el sistema en circunstancias específicas.

El primer paso para escribir un caso de uso es definir un conjunto de actores que estarán involucrados en la historia. Estos actores son las diferentes personas (o dispositivos) que usan el sistema o producto en el contexto de la función y comportamiento que va a describirse. En resumen, un actor es cualquier cosa que se comunique con el sistema o producto y que sea externo a éste.

Finalmente, una vez identificados los actores, como interactúan con el sistema y los límites del sistema, es posible desarrollar los casos de uso.

8.1. Caso de Uso 0. *Framework* para regresión ordinal

Es el primer caso de uso del proyecto software, que muestra el funcionamiento completo del sistema.

Solo hay un actor, que se corresponde con el usuario, que usa el sistema para realizar un experimento. Debido a que los algoritmos implementados se utilizarán a partir del *framework* `ORCA-Python`, que se utilizarán a través de la interfaz del mismo, todos los algoritmos implementados para este *framework* compartirán los mismos casos de uso.

Como fue explicado anteriormente, para realizar el experimento se crearán ficheros de configuración que especifican que algoritmos y que *datasets* se

van a utilizar a la hora de realizar el experimento.

Caso de Uso	CU-0. *framework* para Clasificación
Descripción	Sistema software que simplifica la realización de experimentos para *machine-learning* utilizando ficheros de configuración.
Actores	Usuario
Casos de Uso	CU-1 Ejecutar Experimento.

Cuadro 8.1: CU-0. *framework* para Clasificación

8.2. Caso de Uso 1. Ejecutar Experimento

Lo primero que el usuario tendrá que hacer es configurar el fichero de configuración especificando los parámetros de los diferentes algoritmos, como deben ser tratado los datos, así como el directorio y los *datasets* que se utilizarán en dicho experimento. Al finalizar el experimento, los resultados obtenidos serán guardados en ficheros en la carpeta especificada en el fichero de configuración.

Caso de Uso	CU-1. Ejecutar experimento
Descripción	El usuario realiza un experimento utilizando el archivo de configuración.
Actores	Usuario
Casos de Uso	CU-1-1 Configurar experimento CU-1-2 Optimizar modelo. CU-1-3 Calcular y guardar métricas. CU-1-4 Guardar resultados.

Cuadro 8.2: CU-1 Ejecutar experimento

8.2.1. Caso de Uso 1-1. Configurar Experimento

Para realizar un experimento, previamente el usuario debe crear un archivo de configuración en el que se especifican los algoritmos y los parámetros de los mismos, así como la ubicación y las bases de datos a utilizar.

La configuración del fichero, viene explicada en mayor profundidad en el VI.

Caso de Uso	CU-1. Configurar experimento
Descripción	El usuario introduce toda la información necesaria para la ejecución del experimento.
Actores	Usuario
Flujo principal	1.El usuario especifica en la configuración general del fichero el directorio de los conjuntos de datos, los conjuntos de datos a utilizar, el tipo de preprocesamiento a realizar, el número de *folds* para la *cross-validation*, el número de trabajos a utilizar por el proceso, el directorio donde se guardarán los resultados y las métricas a calcular. 2. El usuario especifica los diferentes algoritmos con los que clasificar los conjuntos de datos, indicando los parámetros a utilizar por dichos algoritmos.
Condiciones finales	Ninguna
Flujos Alternativos	Ninguno

Cuadro 8.3: CU-1 Configurar experimento

8.2.2. Caso de Uso 1-2. Optimizar modelo

Para conseguir un modelo óptimo se utilizará la *cross-validation* de tipo *k-fold* (siendo k un parámetro indicado por el usuario en el archivo de configuración), considerando únicamente los datos de entrenamiento. Este procedimiento se realizará por cada partición de cada conjunto de datos.

Caso de Uso	CU-1-2. Optimizar modelo
Descripción	Utilizando el archivo de configuración creado por el usuario, se obtiene el mejor modelo de cada algoritmo utilizado, para cada conjunto de datos usado.El método utilizado es el de la *cross-validation* de tipo *k-fold*
Actores	Usuario
Condiciones Iniciales	El fichero de configuración del experimento debe existir El formato de las bases de datos debe ser correcto. El fichero de configuración no debe contener errores de sintaxis.
Flujo principal	1. Se cargan las diferentes bases de datos especificadas en el fichero de configuración. 2. Se recorren cada una de las bases de datos partición a partición. 3.Para cada clasificador se determinan los mejores parámetros mediante *cross-validation*.
Condiciones finales	Se obtienen los mejores modelos por partición
Flujos Alternativos	El proceso se detiene debido a un error.

Cuadro 8.4: CU-1-2. Optimizar modelo

8.2.3. Caso de Uso 1-3. Calcular y guardar métricas

Una vez obtenidos los modelos óptimos de cada clasificador para cada conjunto de datos, se realizará el calculo de las métricas especificado en el fichero de configuración. Además, se realizará el calculo de las medias y desviaciones típicas de las métricas para los conjuntos de entrenamiento y test.

Caso de Uso	CU-1-3. Calcular y guardar métricas
Descripción	Una vez obtenidos los mejores modelos para cada partición se calculan sus métricas de rendimiento.
Actores	Usuario
Condiciones Iniciales	Se han calculado los mejores modelos por partición
Flujo principal	1. Se calculan las métricas de rendimiento que fueron especificadas en el fichero de configuración para cada uno de los modelos obtenidos 2. Para cada conjunto de datos, se calcula su media y desviación típica.
Condiciones finales	Se calculan las métricas especificadas para cada modelo y partición Se calcula la media y la desviación típica de las métricas para cada conjunto de datos.
Flujos Alternativos	El proceso se detiene debido a un error.

Cuadro 8.5: CU-1-3. Calcular y guardar métricas

8.2.4. Caso de Uso 1-4. Guardar resultados

Una vez terminado el experimento, los resultados de los modelos óptimos y los cálculos de las métricas son almacenados en ficheros en la ruta especificada en el fichero de configuración.

Caso de Uso	CU-1-4. Guardar resultados
Descripción	Se guardan las métricas calculadas y los modelos óptimos. Las medias y desviaciones típicas se guardan en dos ficheros uno para el conjunto de entrenamiento y otro para las de test.
Actores	Usuario
Condiciones Iniciales	Se han obtenido las métricas y calculado las medias y desviaciones típicas.
Flujo principal	1. Se guardan las métricas obtenidas y los modelos óptimos en ficheros. 2. Se guardan las medias y las desviaciones típicas en dos ficheros, uno para el conjunto de entrenamiento y otro para el de test.
Condiciones finales	Las métricas de todos los modelos quedan almacenadas. Los modelos óptimos para cada partición son almacenados. Se generan los ficheros resumen con las medias y desviaciones típicas para entrenamiento y test
Flujos Alternativos	El proceso se detiene debido a un error.

Cuadro 8.6: CU-1-4. Guardar resultados

Capítulo 9

Tecnologías

En este apartado, se describirán las herramientas utilizadas en la migración de los algoritmos `SVC1V1`, SV1VA y `KDLOR` para el *framework* `ORCA-Python`.

9.1. Introducción a Python

`Python` es un lenguaje de programación de propósito general muy poderoso y flexible, a la vez que sencillo y fácil de aprender, que fue creado a principios de los noventa por Guido Van Rossum. Es un lenguaje de alto nivel, que permite procesar fácilmente todo tipo de estructuras de datos, tanto numéricas como de texto. La sintaxis de `Python` y su tipado dinámico, junto a su naturaleza interpretada lo convierten en un lenguaje ideal para *scripting* y desarrollo rápido de aplicaciones en diversas áreas.

Es administrado por *Python Software Foundation* y posee una licencia de código abierto, denominada *Python Software Foundation License*. Debido a esto es fácilmente obtenible desde la web de `Python`, donde además es posible encontrar diferente distribuciones y referencias a muchos módulos de `Python` de terceros.

El intérprete de `Python` estándar incluye un modo interactivo en el cual se escriben las instrucciones en una especie de intérprete de comandos: las expresiones pueden ser introducidas una a una, pudiendo verse el resultado inmediato de su evaluación, lo que permite probar porciones de código en el modo interactivo antes de integrarlo como parte de un programa. Además, el intérprete de `Python` es fácilmente extensible con funciones y tipos de datos implementados en C o C++.

Al ser un lenguaje de propósito general, podemos encontrar aplicaciones prácticamente en todos los campos científico-tecnológicos:

- Aprendizaje Automático (*Machine Learning*)
- Inteligencia Artificial (IA)
- *Big data* y Análisis de datos
- Desarrollo web
- Desarrollo móvil
- Bases de datos relaciones

Además, `Python` cuenta con numerosas librería, debido a que en la siguiente sección 9.2 se tratarán con más profundidad las librerías usadas en el desarrollo de este proyecto, aquí solo se mencionarán un par de ellas:

- `Keras:` es una biblioteca de Redes Neuronales de Código Abierto escrita en `Python`. Está especialmente diseñada para posibilitar la experimentación en más o menos poco tiempo con redes de Aprendizaje Profundo. Sus fuertes se centran en ser amigable para el usuario, modular y extensible.
- `Django:` es un *framework* de desarrollo web de código abierto, escrito en `Python`, que permite crear aplicaciones complejas rápidamente.

9.2. Bibliotecas utilizadas

En esta sección se nombrarán las bibliotecas de `Python` que han sido utilizadas para el desarrollo de este proyecto software.

`Numpy` [12] es una biblioteca para el lenguaje de programación `Python` que da soporte para crear vectores y matrices de grandes multidimensionales. Además, cuenta con una variada colección de complejas funciones matemáticas que operan con esas matrices.

`Pandas` [13] es una biblioteca para `Python`, que proporciona una serie de estructuras de datos y operaciones para manipulación y el análisis de los

mismos.

`Scikit-Learn` [6] es una biblioteca de software libre para `Python`. Cuenta con varios algoritmos como máquina de vectores soporte, *random forest* y *K-neighbours*. También admite bibliotecas numéricas y científicas como `NumPy` y `SciPy`.

`SciPy` [14] es un software de código abierto que nació a partir de la colección original de Travis Oliphant y que consistía en módulos de extensión para `Python`. La biblioteca de `SciPy` depende de `Numpy`, proporcionando una manipulación de vectores N-dimensionales de una forma cómoda y rápida.

`Cvxopt` [15] es un paquete de software para la optimización convexa basado en `Python`. Su principal propósito es facilitar el desarrollo de software para aplicaciones de optimización convexas utilizando para ello la extensa biblioteca estándar de `Python` y las fortalezas de `Python` al ser un lenguaje de programación de alto nivel.

Capítulo 10

Arquitectura del sistema

En este capítulo se explicará con más detalle la estructura modular que se ha adaptado para este proyecto. En total se describirán tres módulos que son: `KDLOR`, `SVC1V1` y `SVC1VA`.

10.1. KDLOR

Este módulo esta compuesto por el código del algoritmo `KDLOR` para el *framework* `ORCA-Python`, el cuál es una migración y adaptación de la versión del *framework* `ORCA`.

10.2. SVC1V1

Este módulo está compuesto por el código del algoritmo `SVC1V1` para el *framework* `ORCA-Python`, el cuál es una migración y adaptación de la versión del *framework* `ORCA`.

10.3. SVC1VA

Este módulo está compuesto por el código del algoritmo `SVC1VA` para el *framework* `ORCA-Python`, el cuál es una migración y adaptación de la versión del *framework* `ORCA`.

Capítulo 11

Diagramas de Clases

El objetivo de este apartado es refinar el análisis de los módulos del capítulo anterior, para ello usaremos los diagramas de clases.

Los diagramas de clases UML son una notación gráfica usada en los sistemas orientados a objeto. Un diagrama de clase en UML(*Unified Modeling Language*) es un tipo de estructura estática que se utiliza para definir un sistema mostrando sus clases, sus atributos, sus métodos y las relaciones que existen entre los mismos.

Además de los correspondientes diagramas de clase, también se proporcionará una tabla con la especificación de cada clase. La notación utilizada será similar a la 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
...	...	...

Cuadro 11.1: Ejemplo notación de clase

11.1. Clase KDLOR

En esta sección se muestra la clase que implementa el algoritmo KDLOR:

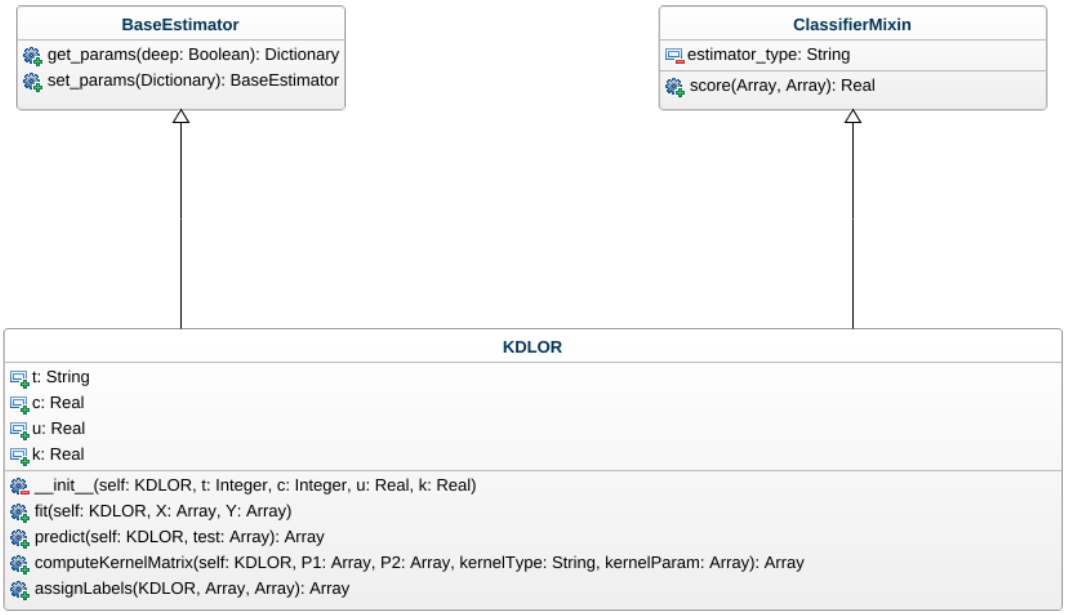

Figura 11.1: Diagrama de clases de la clase KDLOR

Clase: KDLOR		
Clase que contiene el clasificador KDLOR dentro del framework Orca-Python		
Datos		
t(*kernel_type*)	String	Cadena que específica el tipo de kernel a utilizar por el algoritmo
c(cost)	Real	Parámetro usado en la función de optimización.
u	Float	Parámetro usado para evitar matrices mal planteadas.
k	Float	Parametro del kernel.
Métodos		
__init__	Void	Constructor de la clase que inicializa los diferentes parámetros del clasificador
fit	SVC1V1	Se encarga de entrenar el modelo en base al conjunto de entrenamiento de entrada y los valores de los parámetros proporcionados.
predict	Array	Devuelve un array con las etiquetas de las clases predichas para un conjunto de datos.
computeKernelMat	Array	Devuelve un array como resultado de calcular la matriz kernel.
assignLabels	Array	Función auxiliar que devuelve un array para calcular las etiquetas predichas.

Cuadro 11.2: Especificación de la clase SVC1V1

11.2. Clase SVC1V1

En esta sección se muestra la clase que implementa el algoritmo SVC1V1, usando para ello la biblioteca de `scikit-learn`.

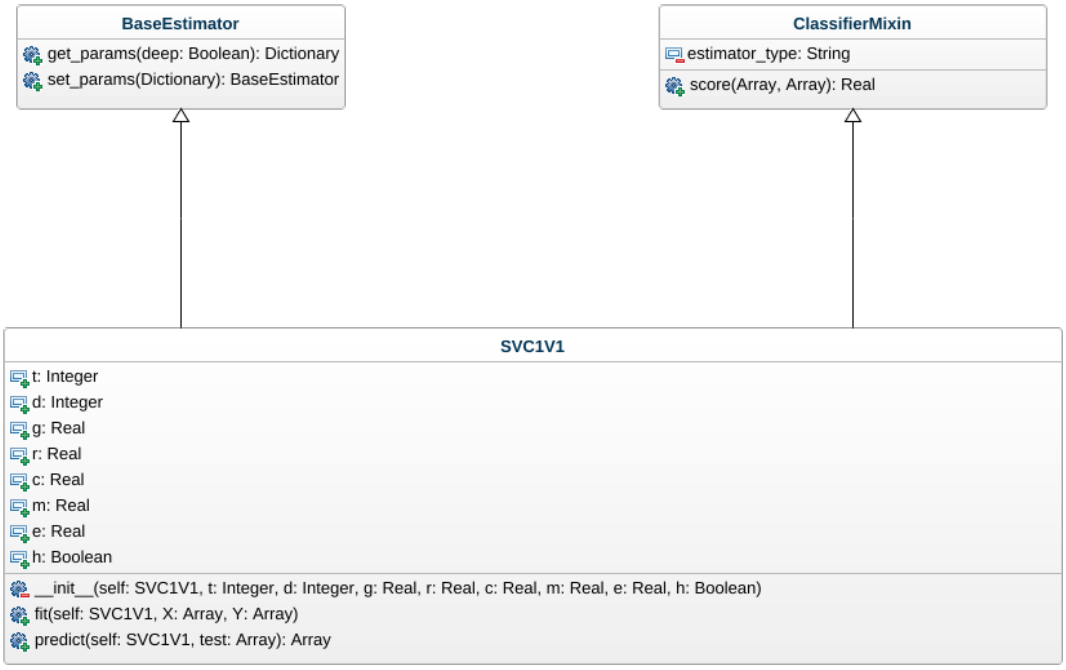

Figura 11.2: Diagrama de clases de la clase SVC1V1

Clase: SVC1V1		
Clase que contiene el clasificador SVC1V1 dentro del framework Orca-Python.		
Datos		
t(kernel_type)	Integer	Número que específica el tipo de kernel a utilizar por el algoritmo.
d(degree)	Integer	Contiene el valor que se utiliza en la función kernel de tipo polinómica.
g(gamma)	Float	Contiene el valor de gamma un coeficiente de la función kernel (rbf,polinómica y sigmoide).
r(coef0)	Float	Termino independiente en la función kernel. Solo es importante en los kernel de tipo polinómico y sigmoide.
c(cost)	Float	Parámetro de regularización.
m(cachesize)	Float	Específica el tamaño de la cache del kernel en MB.
e(epsilon)	Float	Contiene el valor de epsilon, es decir, el criterio de terminación del algoritmo.
h(shrinking)	Boolean	Indica si se utilizará la heurística *shrinking* o no.
Métodos		
__init__	Void	Constructor de la clase que inicializa los diferentes parámetros del clasificador.
fit	SVC1V1	Se encarga de entrenar el modelo en base al conjunto de entrenamiento de entrada y los valores de los parámetros proporcionados.
predict	Array	Devuelve un array con las etiquetas de las clases predichas para un conjunto de datos.

Cuadro 11.3: Especificación de la clase SVC1V1

11.3. Clase SVC1VA

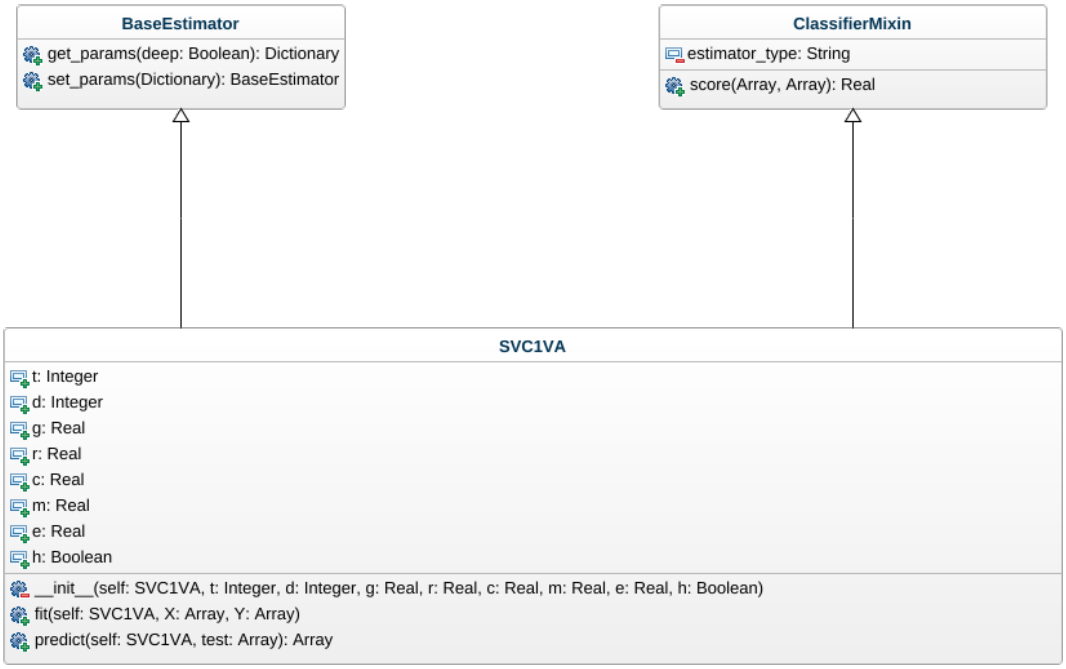

Figura 11.3: Diagrama de clases de la clase SVC1VA

Clase: SVC1VA		
Clase que contiene el clasificador SVC1VA dentro del framework Orca-Python		
Datos		
t(kernel_type)	Integer	Número que específica el tipo de kernel a utilizar por el algoritmo.
d(degree)	Integer	Contiene el valor que se utiliza en la función kernel de tipo polinómica.
g(gamma)	Float	Contiene el valor de gamma un coeficiente de la función kernel (rbf,polinómica y sigmoide).
r(coef0)	Float	Termino independiente en la función kernel. Solo es importante en los kernel de tipo polinómico y sigmoide.
c(cost)	Float	Parámetro de regularización.
m(cachesize)	Float	Específica el tamaño de la cache del kernel en MB.
e(epsilon)	Float	Contiene el valor de epsilon, es decir, el criterio de terminación del algoritmo.
h(shrinking)	Boolean	Indica si se utilizará la heurística *shrinking* o no.
Métodos		
__init__	Void	Constructor de la clase que inicializa los diferentes parámetros del clasificador.
fit	SVC1VA	Se encarga de entrenar el modelo en base al conjunto de entrenamiento de entrada y los valores de los parámetros proporcionados.
predict	Array	Devuelve un array con las etiquetas de las clases predichas para un conjunto de datos.

Cuadro 11.4: Especificación de la clase SVC1VA

Parte IV

Pruebas

Capítulo 12

Pruebas

La fase de pruebas es imprescindible en cualquier proyecto software. Existen muchas estrategias que pueden usarse para probar el software. Un enfoque extremo, puede ser esperarse hasta que el sistema esté completamente construido y luego realizar las pruebas sobre el sistema total, con la esperanza de encontrar errores. Desafortunadamente este enfoque no funciona. En el otro extremo, podrían efectuarse pruebas diariamente, siempre que se construya alguna parte del sistema, este enfoque ,aunque menos atractivo, suele ser muy efectivo. Debido a esto normalmente se suele tomar una solución intermedia. Se toma una visión incremental de las pruebas, comenzando con las unidades individuales, avanza hacia pruebas diseñadas para facilitar la integración de las unidades y culmina con pruebas que ejercitan el sistema construido.

Como el objetivo del proyecto es desarrollar una biblioteca escrita en `Python` para el *framework* `ORCA-Python`, se deberá asegurar el correcto funcionamiento y acoplamiento de los mismos. A continuación se explicaran más detalladamente las pruebas realizadas.

12.1. Pruebas unitarias

Las pruebas de unitarias o de unidad, enfoca los esfuerzos de verificación en la unidad más pequeña del diseño de software: el componente o módulo de software. Al usar la descripción del diseño de componente como guía, las rutas de control importantes se prueban para descubrir errores dentro de la frontera del módulo. Las pruebas unitarias se enfocan en la lógica de procesamiento interno y de las estructuras de datos dentro de las fronteras de un módulo. Este tipo de pruebas pueden realizarse en paralelo para múltiples

módulos y se pueden dividir en dos tipos diferenciados: las pruebas de caja blanca y las pruebas caja negra.

12.1.1. Pruebas de caja blanca

Las cajas de prueba blanca son un tipo de pruebas donde el usuario tiene acceso al código fuente. El objetivo principal de las pruebas de caja blanca es evaluar la lógica interna del software desarrollado. Esto implica probar las diferentes rutas de ejecución, bucles, condicionales y decisiones dentro del código para asegurar su correcto funcionamiento, para ello se probarán diferentes datos de entrada para explorar todas las ramificaciones y condicionales. Al tener acceso al código, es posible comprobar el código e identificar los errores a lo largo del desarrollo del mismo.

12.1.2. Pruebas de caja negra

A diferencia de las pruebas de caja blanca donde se tiene acceso al código, en las pruebas de caja negra lo que se busca es comprobar el correcto funcionamiento del código a través de los valores de entrada y salida sin tener conocimiento de los componentes internos del mismo, es decir, las técnicas de prueba de caja negra permiten derivar conjuntos de condiciones de entrada que revisarán por completo todos los requerimientos funcionales para un programa. Las pruebas de cajas negra no son una alternativa a las de caja blanca, al contrario, son complementarias y posiblemente ayude a descubrir errores diferentes que los métodos de caja blanca.

Las pruebas de caja negra intentan encontrar errores en las categorías siguientes:

- Funciones incorrectas o faltantes
- errores de interfaz
- errores en las estructuras de datos o en el acceso a bases de datos externas
- errores de comportamiento o rendimiento
- errores de inicialización y terminación

A diferencia de las pruebas de caja blanca, que se realizan en fases tempranas del proyecto, las de caja negra tienden a aplicarse durante las últimas

etapas de las prueba. Esto es debido a que no considera la estructura de control, sino que se enfoca en el dominio de la información.

Ambas pruebas se han realizado durante la implementación de los diferentes algoritmos y se han comprobado que los resultados eran los esperados comparándolos con su contraparte en `MATLAB`.

12.2. Pruebas de integración

Una vez que se han comprobado las pruebas unitarias el siguiente paso son las pruebas de integración. En esta fase de pruebas, se comienza a examinar cuidadosamente las conexiones entre cada módulo. El objetivo es tomar los componentes probados de manera individual y construir una estructura de programa que se haya dictado por su diseño.

12.3. Pruebas del sistema

La prueba del sistema es una serie de diferentes pruebas cuyo propósito principal es ejercitar por completo el sistema basado en computadora. Aunque cada prueba tenga un propósito diferente, todo funciona para verificar que los elementos del sistema se hayan integrado de manera adecuada y que se realicen las funciones asignadas. A continuación se explican las diferentes pruebas del sistema que merecen la pena para los sistemas basados en software.

12.3.1. Pruebas de recuperación

Muchos sistemas basados en computadora deben recuperarse de fallas y reanudar el procesamiento con poco o ningún tiempo de inactividad.

La recuperación es una prueba del sistema que fuerza al software a fallar en varias formas y que verifica que la recuperación se realice de manera adecuada.

12.3.2. Pruebas de seguridad

Cualquier sistema basado en computadora que gestione información sensible o cause acciones que puedan dañar (o beneficiar) de manera inadecuada a individuos es un blanco de penetración inadecuada o ilegal.

La prueba de seguridad intenta verificar que los mecanismos de protección que se construyen en un sistema en realidad lo protegerán de cualquier penetración impropia. Con el suficiente tiempo y recursos, las buenas pruebas de seguridad al final de cuentas penetran el sistema por lo que el papel del diseñador de sistemas es hacer que el costo de la penetración sea mayor que el valor de la información que se obtendrá.

12.3.3. Pruebas de esfuerzo

Las pruebas de esfuerzo se diseñan para enfrentar a los programas con situaciones anormales. La prueba de esfuerzo ejecuta un sistema en forma que demanda recursos en cantidad, frecuencia o volumen anormales.

Dentro de las pruebas de esfuerzo existe una variación que se llama prueba de sensibilidad. En algunas situaciones (la más común ocurre en algoritmos matemáticos), un rango muy pequeño de datos contenidos dentro de las fronteras de los datos válidos para un programa pueden causar procesamiento extremo, e incluso erróneo, o profunda degradación del rendimiento. La prueba de sensibilidad intenta descubrir combinaciones de datos que dentro de clases de entrada válidas puedan causar inestabilidad o procesamiento inadecuado.

Dado que se esta trabajando con algoritmos de clasificación ordinal un procesamiento inadecuado puede afectar al correcto funcionamiento del algoritmo de clasificación y por lo tanto se realizarán diferentes pruebas de sensibilidad para asegurar un comportamiento adecuado y estable.

12.3.4. Pruebas de rendimiento

Para sistemas en tiempo real, el software que proporcione la función requerida, pero que no se adecue a los requerimientos de rendimiento, es inaceptable. La prueba de rendimiento se diseña para poner a prueba el rendimiento del software en tiempo de corrida, dentro del contexto de un sistema integrado. Con frecuencia se complementan con las pruebas de esfuerzo.

Dado que se en ocasiones se trabajaran con base de datos grandes es importante que se asegure un buen tiempo de respuesta para un funciona-

miento óptimo de los algoritmos.

12.3.5. Pruebas de despliegue

En muchos casos, el software debe ejecutarse en varias plataformas y bajo más de un entorno. La prueba de despliegue, en ocasiones llamada prueba de configuración, ejercita el software en cada entorno en el que debe operar.

Se llevarán a cabo pruebas de despliegue para verificar el correcto funcionamiento de los algoritmos diseñados en el entorno de Ubuntu y Windows.

12.4. Pruebas de aceptación

Las pruebas de aceptación son las que realizarán los usuarios finales que en este caso, al ser un Trabajo de Fin de Grado, serán los directores del proyecto. El objetivo principal sería evaluar la fiabilidad del sistema y que se cumplan con todos los objetivos del proyecto a través de experimentos que ellos mismos realicen.

Capítulo 13

Resultados experimentales

Una vez desarrollado el sistema y tras superar la fase de prueba, llego el momento de realizar las pruebas experimentales:

- Comprobar el correcto funcionamiento de los diferentes clasificadores implementados `KDLOR`, `SVC1V1` y `SVC1VA`.
- Buscar los parámetros óptimos para los diferentes algoritmos desarrollado, con el objetivo de obtener resultados iguales o mejores que los proporcionados por el *framework* `ORCA-Python`.
- Realizar pruebas de estrés para garantizar la fiabilidad del sistema y que el tiempo de ejecución es óptimo.

13.1. Diseño

En esta sección, se explicará en detalle el diseño elegido a la hora de realizar los experimentos, para verificar la validez de los resultados obtenidos en los mismos. Para ello dividiremos esta sección en dos apartados uno destinado a las bases de datos utilizadas 13.1.1 y otro dirigido a los valores que tomaran los parámetros durante los experimentos 13.1.2.

13.1.1. Conjuntos de datos

En este apartado se detallarán las bases de datos a utilizar durante los experimentos. En nuestro caso se han elegido las bases de datos de clasificación ordinal que son un total de 17 problemas reales a los que se le han particionado usando un 30-*holdout*. Todos ellos se han conseguido en la página web del grupo `AYRNA` [16].

Dataset	#Inst.	#Atr.	#Clase	Distribución de la clase
contact-lenses(CL)	24	6	3	(15, 5, 4)
pasture(PA)	36	25	3	(12, 12, 12)
squash-stored(SS)	52	51	3	(23, 21, 8)
squash-unstored(SU)	52	52	3	(24, 24, 4)
tae(TA)	151	54	3	(49, 50, 52)
newthyroid (NT)	215	5	3	(30, 150, 35)
balance-scale (BS)	625	4	3	(288, 49, 288)
SWD (SW)	1000	10	4	(32, 352, 399, 217)
car (CA)	1728	21	4	(1210, 384, 69, 65)
bondrate (BO)	57	37	5	(6, 33, 12, 5, 1)
toy (TO)	300	2	5	(35, 87, 79, 68, 31)
eucalyptus (EU)	736	91	5	(180, 107, 130, 214, 105)
LEV (LE)	1000	4	5	(93, 280, 403, 197, 27)
automobile (AU)	205	71	6	(3, 22, 67, 54, 32, 27)
winequality-red (WR)	1599	11	6	(10, 53, 681, 638, 199, 18)
ESL (ES)	488	4	9	(2, 12, 38, 100, 116, 135, 62, 19, 4)
ERA (ER)	1000	4	9	(92, 142, 181, 172, 158, 118, 88, 31, 18)

Cuadro 13.1: Problemas de regresión ordinal

13.1.2. Parámetros de los experimentos

Una vez seleccionados las bases de datos para los experimentos el siguiente paso a realizar será elegir los parámetros tanto de los algoritmos como de los ficheros de configuración para los experimentos a realizar. En primer lugar las métricas que se han elegido a mejorar en los experimentos son el MAE y el MZE, el número de *folds* que se utilizarán en la validación cruzada es 5 y el número de jobs será 10. Todos estos parámetros se explicarán en

su respectiva sección del Apéndice A.6.1.

Por otra parte para los algoritmos se ha utilizado los mismos parámetros que venían en el artículo de investigación del grupo AYRNA para poder comparar los resultados de las métricas obtenidas.

Por una parte para los algoritmos `SVC1V1` y `SVC1VA` se han utilizado los mismos valores, que se describirán a continuación:

- c: [0.001, 0.01, 0.1, 1, 10, 100, 1000]
- g: [0.001, 0.01, 0.1, 1, 10, 100, 1000]

Mientras que para el algoritmo KDLOR se han utilizado lo siguientes parámetros:

- c: [0.1, 1.0, 10.0],
- k: [0.001,0.01,0.1,1, 10, 100, 1000],
- u: [0.01, 0.001, 0.0001, 0.00001, 0.000001]

Dichos parámetros vienen explicados en la sección 11, por lo que para más información sobre los mismos o de los métodos de cada clase, se recomienda consultar dicho apartado.

13.2. Resultados

En esta sección se hará una comparativa de los resultados obtenidos con respecto a su versión de `MATLAB`. Como se menciono anteriormente estos resultados son los obtenidos en el artículo de investigación del grupo AYRNA. A continuación se explicará como se ha procedido para realizar la comparación.

Se mostrarán dos tablas cada una para su respectiva métrica, que en nuestro caso son MZE y MAE. Para cada base de datos se mostrará la versión de `ORCA` versus la de `ORCA-Python`, donde el resultado a mostrar sería la media de la métrica acompañada de su desviación típica.

Se añadirá una casilla que especifique si se ha superado o no el resultado de su respectiva versión de `ORCA`. En el ejemplo de la tabla se puede observar que como el resultado de la métrica1 obtenida es mejor que el de la versión

Dataset	$Algoritmo_{ORCA}$	$Algoritmo_{ORCA-Python}$	Superado
Dataset A	Métrica1	Métrica2	No

Cuadro 13.2: Ejemplo de tabla comparativa

implementada en `ORCA-Python` no se ha superado. Cabe destacar que el nombre de las bases de datos a utilizar que aparecerán, serán las abreviaturas que se mostraron en la tabla 13.1.

Dataset	$SVC1V1_{ORCA}$	$SVC1V1_{ORCA-Python}$	Superado
CL	$0{,}78_{0{,}126}$	$0{,}52_{0{,}17}$	Si
BS	$0{,}028_{0{,}15}$	$0{,}028_{0{,}015}$	Si
CA	$0{,}006_{0{,}005}$	$0{,}005_{0{,}0048}$	Si
TA	$0{,}439_{0{,}59}$	$0{,}436_{0{,}06}$	Si
WI	$0{,}352_{0{,}023}$	$0{,}350_{0{,}0223}$	Si
ER	$736_{0{,}025}$	$0{,}732_{0{,}025}$	Si

Cuadro 13.3: Comparación de `SVC1V1` en términos de MZE. Se muestran los valores $\text{Media}_{Desviaciontipica}$

Dataset	$SVC1VA_{ORCA}$	$SVC1VA_{ORCA-Python}$	Superado
CL	$0{,}278_{0{,}110}$	$0{,}28_{0{,}124}$	No
BS	$0{,}033_{0{,}012}$	$0{,}027_{0{,}014}$	Si
CA	$0{,}014_{0{,}006}$	$0{,}005_{0{,}0048}$	Si
TA	$0{,}448_{0{,}65}$	$0{,}434_{0{,}062}$	Si
WI	$0{,}361_{0{,}024}$	$0{,}348_{0{,}018}$	Si
ER	$0{,}825_{0{,}026}$	$0{,}734_{0{,}029}$	Si

Cuadro 13.4: Comparación de `SVC1VA` en términos de MZE. Se muestran los valores $\text{Media}_{Desviaciontipica}$

Como se puede observar en los algoritmos de `SVC1V1` y `SVC1VA` para la mayoría de casos las versiones de `scikit-learn` son mejores que las versiones del *framework* `ORCA`. En cuanto al algoritmo `KDLOR` posiblemente la diferencia de resultados sea debido a que se esta utilizando un algoritmo de optimización diferente.

Dataset	KDLOR_{ORCA}	$\text{KDLOR}_{ORCA-Python}$	Superado
CL	$0{,}0339_{0{,}155}$	$0{,}388_{0{,}018}$	No
BS	,025,020	.16.028	No
CA	$0{,}047_{0{,}010}$	$0{,}046_{0{,}008}$	Si
TA	$0{,}433_{0{,}055}$	$0{,}440_{0{,}058}$	No
WI	$0{,}350_{0{,}019}$	$0{,}350_{0{,}001}$	Si
ER	$0{,}805_{0{,}031}$	$0{,}820_{0{,}002}$	No

Cuadro 13.5: Comparación de `KDLOR` en términos de MZE. Se muestran los valores $\text{Media}_{Desviaciontipica}$

Dataset	SVC1V1_{ORCA}	$\text{SVC1V1}_{ORCA-Python}$	Superado
CL	$0{,}52_{0{,}22}$	$0{,}52_{0{,}17}$	Si
BS	$0{,}03_{0{,}01}$	$0{,}029_{0{,}013}$	Si
CA	$0{,}01_{0{,}01}$	$0{,}007_{0{,}005}$	Si
TA	$0{,}54_{0{,}1}$	$0{,}58_{0{,}17}$	No
WI	$0{,}41_{0{,}02}$	$0{,}407_{0{,}0217}$	Si
ER	$1{,}29_{0{,}07}$	$1{,}272_{0{,}056}$	Si

Cuadro 13.6: Comparación de `SVC1V1` en términos de MAE. Se muestran los valores $\text{Media}_{Desviaciontipica}$

Dataset	SVC1VA_{ORCA}	$\text{SVC1VA}_{ORCA-Python}$	Superado
CL	$0{,}46_{0{,}19}$	$0{,}455_{0{,}239}$	Si
BS	$0{,}03_{0{,}01}$	$0{,}03_{0{,}014}$	Si
CA	$0{,}01_{0{,}01}$	$0{,}007_{0{,}0059}$	Si
TA	$0{,}50_{0{,}09}$	$0{,}52_{0{,}097}$	No
WI	$0{,}42_{0{,}02}$	$0{,}416_{0{,}024}$	Si
ER	$2{,}02_{0{,}12}$	$1{,}28_{0{,}054}$	Si

Cuadro 13.7: Comparación de `SVC1VA` en términos de MAE. Se muestran los valores $\text{Media}_{Desviaciontipica}$

Dataset	KDLOR_{ORCA}	$\text{KDLOR}_{ORCA-Python}$	Superado
CL	$0{,}52_{0{,}22}$	$0{,}388_{0{,}182}$	Si
BS	$0{,}16_{0{,}02}$	$0{,}16_{0{,}028}$	Si
CA	$0{,}05_{0{,}01}$	$0{,}0464_{0{,}088}$	Si
TA	$0{,}46_{0{,}07}$	$0{,}440_{0{,}058}$	Si
WI	$0{,}390_{0{,}02}$	$0{,}350_{0{,}001}$	Si
ER	$1{,}78_{0{,}1}$	$0{,}820_{0{,}002}$	Si

Cuadro 13.8: Comparación de `KDLOR` en términos de MAE. Se muestran los valores $\text{Media}_{Desviaciontipica}$

Parte V

Conclusiones

Capítulo 14

Conclusiones del Autor

Una vez terminadas todas las pruebas y realizados los experimentos, se da por terminado este proyecto y por tanto el desarrollo de los algoritmos `KDLOR`, `SVC1V1` y `SVC1VA` para el *framework* `ORCA-Python`.

14.1. Objetivos de formación alcanzados

- Conocimientos teóricos y prácticos del uso del *framework* `ORCA`.
- Aprendizaje y profundización de los diferentes métodos a implementar `KDLOR`, `SVC1V1` y `SVC1VA`.
- Dominio del *framework* `ORCA-Python` así como de las diferentes bibliotecas de `scikit-learn`.
- Dominio del lenguaje de programación `Python`.

14.2. Objetivos operacionales alcanzados

- Correcta implementación de los diferentes algoritmos propuestos.
- Comprobación del correcto funcionamiento de dichos algoritmos mediante diversas pruebas **??**.

Capítulo 15

Futuras Mejoras

En este capítulo se citaran brevemente algunas de las futuras mejoras a realizar al *framework* `ORCA-Python`. Como se trata de un Trabajo de Fin de Grado algunas de las posibles mejoras serían:

- Terminar de implementar los diferentes algoritmos que están en `ORCA`, tales como `POM`, `ELMOP`, `HPOLD`, etc.
- Optimización de los algoritmos ya implementados ya sea con nuevas métricas o con funciones que optimicen los tiempos de ejecución de los algoritmos ya existentes.

Bibliografía

[1] Aurélien Géron. *Hands-on Machine Learning with Scikit-Learn, keras TensorFlow*. O'Reilly, 2019, pág. 4.

[2] Javier Sánchez-Monedero, Pedro A. Gutiérrez y María Pérez-Ortiz. "ORCA: A Matlab/Octave Toolbox for Ordinal Regression". En: *Journal of Machine Learning Research* 20.125 (2019), págs. 1-5. URL: http://jmlr.org/papers/v20/18-349.html.

[3] Iván Bonaque Muñoz, Pedro Antonio Gutiérrez Peña y Javier Sánchez Monedero. *Trabajo de Fin de Grado. Framework en Python para problemas de clasificación ordinal*. 2019.

[4] Ángel Heredia Pérez, Pedro Antonio Gutiérrez Peña y Juan Carlos Fernández Caballero. *Trabajo de Fin de Grado. Desarrollo de una biblioteca para algoritmos de clasificación ordinal en Python*. 2020.

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

[6] *Sitio oficial de Scikit-learn*. URL: https://scikit-learn.org/stable/. [Online. Última consulta: 06-10-2021].

[7] *Sitio oficial de Sacred*. URL: https://sacred.readthedocs.io/en/stable/. [Online. Última consulta: 06-10-2021].

[8] B.-Y. Sun, D. D. Wu J. Li, X.-M. Zhang y W.-B. Li. "Kernel discriminant learning for ordinal regression". En: *IEEE Trans. Knowl. Data Eng* 22.6 (2016), págs. 906-910. URL: https://www.researchgate.net/publication/224569314_Kernel_Discriminant_Learning_for_Ordinal_Regression.

[9] C.-W. Hsu y C.-J. Lin. "A comparison of methods for multiclass support vector machines". En: *IEEE Trans. Neural Netw.* 13.2 (2002), págs. 415-425.

[10] *Weka Wiki*. URL: https://waikato.github.io/weka-wiki/documentation/. [Online. Última consulta: 20-8-2024].

[11] *How to Apply Ordinal Logistic Regression for prediction of Materials Properties*. URL: https://bjoyita.github.io/Tut11_OrdReg.html#:~:text=MORD%20is%20a%20Python%20package,%2C%20and%20classification%2Dbased%20models.. [Online. Última consulta: 20-8-2024].

[12] *Sitio oficial de NumPy*. URL: https://numpy.org/. [Online. Última consulta: 06-10-2021].

[13] *Sitio oficial de Pandas*. URL: https://pandas.pydata.org/. [Online. Última consulta: 06-10-2021].

[14] *Sitio oficial de Scipy*. URL: https://www.scipy.org/. [Online. Última consulta: 06-10-2021].

[15] *Sitio oficial de Cvxopt*. URL: https://cvxopt.org/. [Online. Última consulta: 10/4/2022].

[16] *Departamento de Informática y Análisis Numérico de la Universidad de Córdoba. Aprendizaje y redes neuronales artificiales*. URL: https://www.uco.es/grupos/ayrna/index.php/es. [Online. Última consulta: 06-10-2021].

[17] *Acerca de Python*. URL: https://entrenamiento-python-basico.readthedocs.io/es/latest/leccion1/introduccion.html#python-psf. [Online. Última consulta: 06-10-2021].

Parte VI

Apéndices

Apéndice A

Manual de Usuario

A.1. Descripción del producto

Este producto es una ampliación del *framework* `ORCA-Python`[17], que constará con tres nuevos algoritmos implementados previamente en `ORCA` [2], los cuales son `SVC1V1`, `SVC1VA` y `KDLOR`.

A lo largo de este Manual de Usuario se explicará el uso del *framework* con dichas mejoras implementadas.

A.2. Requisitos del sistema

A.3. ¿Qué es ORCA-Python?

`ORCA-Python` es un *framework* escrito en `Python` integrado con los módulos de scikit-Learn y sacred, que busca automatizar los experimentos de *machine learning* mediante ficheros de configuración fáciles de entender.

Este *framework* es compatible con cualquier algoritmo que se encuentre implementado en Scikit-Learn o bien creado por el propio usuario siempre que siga las reglas de compatibilidad de esta biblioteca.

A.4. Instalación

`ORCA-Python` se ha desarrollado y probado en GNU/Linux utilizando `Python` 2 y `Python` 3.

A.4.1. Requisitos para la instalación

Para la correcta ejecución de *framework* requiere de la instalación de las siguientes dependencias de `Python`:

- `NumPy` (probado con la versión 1.18.1)
- `Pandas` (probado con la versión 1.0.1)
- `Scikit-Learn` (probado con la versión 0.8.1)
- `Scipy` (probado con la versión 0.22.1)
- `Sacred` (probado con la versión 1.4.1)
- `Cvxopt` (probado con la versión 1.3.0)

Para la instalación de todas las dependencias se incluye el archivo requirements.txt, que facilitará el proceso de utilizando el gestor de paquetes pip. Al ejecutar el siguiente comando se instalarán todas las dependencias mencionadas anteriormente:

```
pip install -r requirements.txt
```

A.4.2. Descarga de ORCA-Python

Para descargar `ORCA-Python` simplemente se debe clonar desde el repositorio de GitHub usando el siguiente comando:

```
git clone https://github.com/ayrna/orca-python
```

Otra posible alternativa seria usando el botón de *Download Zip* de dicho repositorio de GitHub.

A.4.3. Probando la instalación

Para comprobar que todo ha sido correctamente instalado se proporciona un experimento prefabricado. Para realizar esta prueba se debe ejecutar el siguiente comando desde la raíz del repositorio del *framework*:

```
python config.py with configurations/full_functionality_test.json -l ERROR
```

```
##############################
 Running Experiment
##############################

Running tae dataset
--------------------------
Running LR ...
  Running Partition 0
  Running Partition 1
  ...
  Running Partition 27
  Running Partition 28
  Running Partition 29
...
Running SVOREX ...
  Running Partition 0
  Running Partition 1
  ...
  Running Partition 27
  Running Partition 28
  Running Partition 29

Saving Results...
```

Para explicar la información que proporciona el *framework* sobre el experimento, se analizará la salida generada al ejecutar el experimento prefabricado:

- Running tae dataset indica que el *dataset* tae ha sido cargado y se va a proceder a ejecutar los diferentes algoritmos sobre sus particiones.
- Running SVOREX... indica que se va a empezar a ejecutar la configuración del algortimo SVOREX sobre las diferentes particiones del *dataset* actual
- Running Partition X indica la partición actual sobre la que se está aplicando un determinado algoritmo.
- Por último, al completar el experimento, el mensaje `Saving Results`, indica que se esta generando los archivos resumen con las medias y desviaciones típicas de todas las particiones para cada *dataset* y algoritmo.

A.5. Desinstalación

Si la instalación se ha realizado en un entorno virtual, eliminarlo sera tan sencillo como eliminar la carpeta donde se clonó el repositorio y también la carpeta del propio entorno virtual si no se desea conservar.

En caso de que la instalación se haya realizado sobre la instalación de `Python` del sistema o se quiere conservar el entorno virtual se deberá ejecutar el siguiente comando desde el repositorio raíz del *framework*:

```
pip uninstall --yes -r requirements.txt
```

Tras ejecutar el comando anterior todas las dependencias instaladas serán eliminadas del sistema y solo quedará borrar la carpeta donde el repositorio fue clonado.

A.6. ¿Cómo utilizar ORCA-Python?

Este tutorial usa 3 pequeños *datasets* (balance-scale,contact-lenses y tae) que se encuentran en la carpeta *datasets*. Los *datasets* se encuentran particionados con un *30-holdout*(es decir, 30 particiones de tipo *holdout*), teniendo cada partición una parte de entrenamiento y otra de test.

A.6.1. Archivos de Configuración

Los experimentos se lanzan y configuran a través de ficheros de configuración en formato JSON. Estos archivos se componen de dos secciones:

- **Configuración general:** Llamada *general-conf* en el archivo de configuración, se encarga de indicar información básica acerca del experimento a realizar: localización de los conjuntos de datos, nombres de los diferentes *datasets* que utilizarán,etc
- **Configuraciones:** Llamada *configurations*, indica al *framework* aquellos algoritmos que se utilizarán en el experimento. Además, es posible indicar que parámetros debe utilizar cada algoritmo.

general-conf
A continuación se muestra la sección *general-conf* de los archivos $KDLOR\_test.json, SVC1V1\_test.$

```
    "general_conf": {

    "basedir": "datasets/"",
    "datasets": ["tae", "balance-scale", "contact-lenses"],
    "hyperparam_cv_folds": 5,
    "jobs": 10,
    "input_preprocessing": "std",
    "output_folder": "my_runs/",
    "metrics": ["ccr", "mae", "mze"],
    "cv_metric": "mae"
}
```

- ***basedir:*** Es la ruta a la carpeta que contiene los *datasets*, solo se permite una única ruta. Admite tanto una ruta relativa como una absoluta.
- ***datasets:*** Es el nombre de las bases de datos que se utlizarán en el experimento. El nombre de los conjuntos de datos que se especifiquen aquí deberá corresponderse con una carpeta dentro *basedir* en el que se encuentren sus diferentes particiones.
- ***hyperparam_cv_folds:*** Número de folds que se utilizarán en la validación cruzada cuando se optimicen los hiperparámetros.
- ***jobs:*** Número de procesos que se lanzarán durante la validación cruzada. Si se utiliza -1 se usarán todos los núcleos del procesador por defecto.
- ***input_preprocessing:*** Tipo de preprocesamiento que aplicar a los datos, siento *std* para estandarización y *norm* para normalización. Si se especifica una cadena vacía (") o se omite la opción, el preprocesado de los datos no se realiza.
- ***output_folder:*** Ruta de la carpeta donde almacenar los resultados de los experimentos.
- ***metrics:*** Nombre de las métricas de rendimiento que se utilizarán durante el experimento
- ***cv_metric:*** Es la métrica que utilizará *GridSearchCV* para determinar los mejores parámetros para cada clasificador.

La mayoría de las variables mencionadas anteriormente tienen valores por defecto(especificado en *config.py* pero *basedir* y *datasets* siempre deben ser especificadas.

A.6.2. Configuraciones de los algoritmos

configurations Contiene los diferentes algoritmos que se aplicarán sobre los conjuntos de datos especificados en la sección anterior. Cada algoritmo contendrá los valores que tomarán sus hiperparámetros, ya se un valor fijo o un conjunto de valores. De dicho conjunto de valores se elegirá uno durante la fase de validación cruzada.

A continuación se muestra la sección *configurations* de los archivos $KDLOR\_test.json$, $SVC1V1\_test.json$, $SVC1VA\_test.json$:

```
"configurations": {

 "KDLOR": {

    "classifier": "KDLOR",
    "parameters": {
      "c": [0.1, 1.0, 10.0],
      "k": [0.001, 0.01,  0.1, 1.0, 10.0, 100.0, 1000.0],
      "u": [0.000001,0.00001,0.0001,0.001,0.01]
         }

      }

 },

 "SVC1V1": {

    "classifier": "SVC1V1",
    "parameters": {
      "c": [0.001, 0.01,  0.1, 1, 10, 100, 1000],
      "g": [0.001, 0.01,  0.1, 1, 10, 100, 1000]
         }
      },
```

```
    "SVC1VA": {

       "classifier": "SVC1VA",
       "parameters": {
         "c": [0.001, 0.01,  0.1, 1, 10, 100, 1000],
         "g": [0.001, 0.01,  0.1, 1, 10, 100, 1000]
            }
         }

}
```

El nombre de cada configuración será el que quiera el usuario, y consta de:

- ***classifier***: específica el clasificador a utilizar. Se puede especificar de dos diferentes maneras:
  - Una ruta relativa al algoritmo *scikit-learn*.
  - El nombre de la clase del algoritmo en la carpeta Classifiers de la raíz del *framework*
- ***parameters:*** Hiperparámetros a optimizar durante la validación cruzada. No es necesario especificar un conjunto de valores, puede ser uno solo.

A.6.3. Parámetros de los algoritmos

Mientras que los parámetros para los algoritmos `SVC1V1` y `SVC1VA` son los mismos, estos difieren con respecto al algoritmo `KDLOR` como se ve en el ejemplo de estos en la sección A.6.2. Dichos parámetros se explican detalladamente en su correspondiente sección 11 , por lo que no se profundizará en este anexo.

A.6.4. Formato de las Bases de Datos

Partiendo de la raíz del *framework* tenemos que el conjunto de datos se encontrará en la ruta: *datasets/*.En el interior de dicha carpeta se encontrarán los ficheros que contienen la base de datos. En caso de no ser una base de datos particionada el nombre de los ficheros será: *train_dataset.csv* en caso de ser el fichero que contiene los datos de entrenamiento y *test_dataset.csv* en caso de ser el fichero que contiene los datos de test. En el caso de ser una

base de datos particionada, la extensión del *.csv* de los ficheros cambia por el número de la partición, por ejemplo, *train_toy.10* y *test_toy.10*.

A.6.5. Ejecutando un experimento

Lanzar un experimento es tan simple como ejecutar con el interprete de `Python` el fichero *config.py* y especificar el archivo de configuración que se desea utilizar. El comando a utilizar sería parecido al siguiente:

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

Esta forma de ejecución tiene dos problemas: el primero es que los resultados no se pueden volver a reproducir puesto que la semilla utilizada es aleatoria y el segundo es que `Sacred` muestra mucha información por pantalla que realmente no interesa. Para el primer caso la solución es tan simple como añadir una semilla determinada a la hora de utilizar el comando:

```
python config.py with experiment_file.json seed=12345
```

Y para silenciar `sacred` tan solo tendríamos que añadir -l ERROR al comando anterior:

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

A.6.6. Formato de los resultados

Si al finalizar el experimento no se ha producido ningún error, el *framework* almacenará toda la información producida.

Se supondrá que se ha configurado la opción *output_folder* con la ruta *my_runs*.En la raíz del repositorio se generará dicha carpeta si no existe y será en su interior donde se almacenen los resultados de los experimentos realizados. Cada experimento genera una carpeta con el nombre *exp-año-mes-hora-minuto-segundo*. En su interior se generará subcarpetas BBDD-Clasificador y ficheros resumen.

Se generarán tantas subcarpetas como combinaciones de conjunto de datos y clasificadores haya. Dentro de esta carpeta podemos encontrar una carpeta con los mejores modelos obtenidos en la fase de validación, uno por cada partición del conjunto de datos, así como una carpeta con las predicciones realizadas por el mejor modelo de cada partición, tanto del conjunto de entrenamiento como del de test. Además, dicha carpeta contendrá un archivo CSV con las métricas especificadas en el archivo de configuración y

los tiempos computacionales requeridos en cada fase de entrenamiento, cada fila del fichero se corresponde con una partición.

Al final del experimento se generarán dos ficheros resumen, uno para las métricas obtenidas con los conjuntos de entrenamiento y otro para las de test. En este fichero existirá tantas filas como base de datos y para cada una se calculará la media y desviación típica de cada una de las métricas y tiempos computacionales obtenidos.