UNIVERSIDAD DE CÓRDOBA
ESCUELA POLITÉCNICA SUPERIOR
GRADO EN INGENIERÍA INFORMÁTICA

TRABAJO FIN DE GRADO

Algoritmos de entrenamiento de redes neuronales artificiales para la biblioteca Orca-Python

Manual Técnico y de Usuario

Autor: Adrián López Ortiz

Directores:
Pedro Antonio Gutiérrez Peña
David Guijo Rubio

Algoritmos de entrenamiento de redes neuronales artificiales para la biblioteca Orca-Python

Firmado en Córdoba, 9 de septiembre de 2021

Directores:

Firmado por GUIJO RUBIO DAVID - 45886352H el día 09/09/2021 con un certificado emitido por AC FNMT Usuarios

Fdo: Dr. Pedro Antonio Gutiérrez Peña

Fdo: Dr. David Guijo Rubio

Autor:

Fdo: Adrián López Ortiz

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

Informa:

Que el presente Trabajo Fin de Grado titulado *Algoritmos de entrenamiento de redes neuronales artificiales para la biblioteca Orca-Python*, constituye la memoria presentada por **Adrián López Ortiz** para aspirar al título de Graduado en Ingeniería Informática, y ha sido realizado bajo mi dirección en la Escuela Politécnica Superior de la Universidad de Córdoba reuniendo, a mi juicio, las condiciones necesarias exigidas en este tipo de trabajos.

Y para que conste, se expide y firma el presente certificado en Córdoba, 9 de septiembre de 2021.

El Director:

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

David Guijo Rubio, Profesor Sustituto Interino de la EPSC del Dpto. de Informática y Análisis Numérico y miembro investigador del grupo AYRNA.

Informa:

Que el presente Trabajo Fin de Grado titulado *Algoritmos de entrenamiento de redes neuronales artificiales para la biblioteca Orca-Python*, constituye la memoria presentada por **Adrián López Ortiz** para aspirar al título de Graduado en Ingeniería Informática, y ha sido realizado bajo mi dirección en la Escuela Politécnica Superior de la Universidad de Córdoba reuniendo, a mi juicio, las condiciones necesarias exigidas en este tipo de trabajos.

Y para que conste, se expide y firma el presente certificado en Córdoba, 9 de septiembre de 2021.

El Director:

```
Firmado por GUIJO RUBIO DAVID - 45886352H el día
09/09/2021 con un certificado emitido por AC
FNMT Usuarios
```

Fdo: Prof. Dr. David Guijo Rubio

Agradecimientos

En primer lugar, agradecer a mi familia por animarme y apoyarme siempre para finalizar el grado.

También me gustaría agradecer a mis directores de proyecto, que me enseñaron y me animaron a realizar este Trabajo para finalizar mis estudios y me han ayudado en todo momento con las diferentes dificultades que surgieron en la realización de este.

Por último, me gustaría dar gracias a todos los amigos, compañeros y profesores que me han acompañado a lo largo del grado, que hicieron que esta etapa fuera más fácil y llevadera.

Índice General

Índice de Figuras

Índice de Tablas

Parte I

Introducción

Capítulo 1

Introducción

Hoy en día, se están transfiriendo, almacenando y procesando ingentes cantidades de datos a cada instante a través de sistemas informáticos. En un principio, este volumen de datos no tiene valor alguno para las empresas y organizaciones a las que pertenecen, pero, si estos datos son tratados de la forma adecuada, se podría sacar información con valor. Este proceso es conocido como Descubrimiento de Conocimiento en Bases de Datos (en inglés, *Knowlege Discovery in Databases, KDD*). La extracción de información de valor o útil de grandes de cantidades de datos almacenadas mediante la búsqueda de patrones es denominada Minería de Datos (en inglés, *Data-Mining*). Unas ramas del campo de la informática como la Inteligencia Artificial (IA) y el Aprendizaje Automático (en inglés, Machine Learning) que filtran datos para obtener información de estos. El proceso estandarizado que se sigue en la extracción de información sobre datos se puede observar en la Figura 1.1.

Minería de Datos (en inglés, *Data-Mining*), es la exploración y análisis, por medios automáticos o semiautomáticos, de grandes cantidades de datos para descubrir patrones significativos. De esta forma, una vez obtenido esos patrones de datos útiles, será el turno del Aprendizaje Automático.

El Aprendizaje Automático (en inglés, *Machine Learning*), es un campo de las Ciencias de la Computación cuyo objetivo es generar modelos ma-

Figura 1.1: Proceso de obtención de conocimiento.

temáticos con la finalidad de generar conocimiento o información. Este está dividido en dos categorías: aprendizaje supervisado y no-supervisado. En el aprendizaje supervisado, un experto proporciona una etiqueta a cada patrón para el entrenamiento, en el no-supervisado no ocurre esto, por lo que los patrones carecen de etiqueta. Tras el entrenamiento, aplicando el método de aprendizaje supervisado que consideremos se generará un resultado en la clasificación de los patrones, así cómo un modelo matemático que podemos aplicar a otro conjunto de patrones para obtener una predicción de la clasificación de estos. El funcionamiento del método de aprendizaje supervisado se puede observar en la Figura 1.2.

En regresión los valores de los conjuntos que se desean reproducir han de tener un valor continuo, siendo este el valor que se desea reproducir. Mientras tanto, la clasificación tiene la tarea de asignar una clase, es decir predecir a que clase pertenece un conjunto de datos.

En concreto, en este trabajo se estudiarán problemas de clasificación ordinal o regresión ordinal [1]. Estos son problemas de clasificación multiclase (más de dos clases o categorías) en los que dichas clases presentan una relación de orden entre ellos.

Por ejemplo, en un problema en el que se quiera discernir el estadio de una enfermedad, siendo varios los posibles (leve, medio, grave, extremo), habrá una categoría o clase para cada estadio. En este caso, se puede observar de forma bastante obvia, que la diferencia del error de clasificar una enfermedad

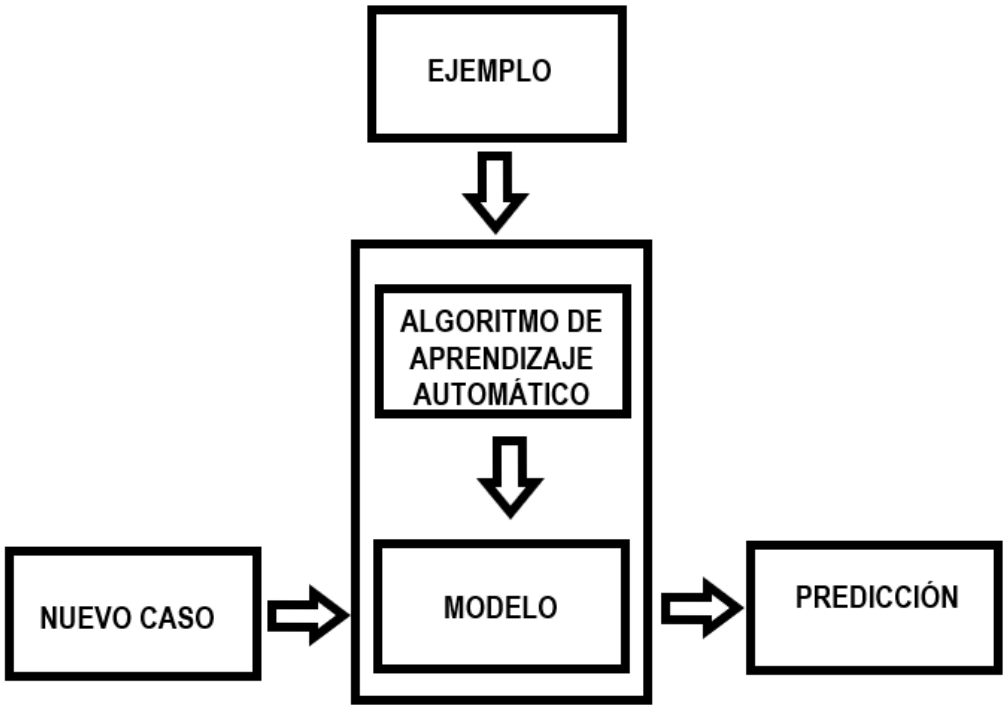

Figura 1.2: Esquema del funcionamiento básico del *Machine Learning*

de estadio leve como medio, no es la misma que si se clasifica como severo. De esta forma, teniendo en cuenta la gravedad del error o la diferencia de error, se podrá penalizar en menor o mayor medida al clasificador para corregir el modelo de aprendizaje.

En este punto de vista ordinal entre clases, hay distintas "distancias.entre clases mientras que si se observa el problema desde un punto de vista nominal, la distancia entre todas las clases es la misma, siendo el mismo error cometido en los dos supuestos anteriores.

Al ser la regresión ordinal un reciente hallazgo dentro del campo del Aprendizaje Automático, no hay muchas herramientas de software que puedan usarse.

En las herramientas existentes, se incluye Weka [2], que permite el uso de varios algoritmos de Aprendizaje Automático, pero que tan solo dispone de un algoritmo de clasificación ordinal.

Otra herramienta tener muy en cuenta es ORCA (*Ordinal Regression and Classification Algorithms*) [3]. Este *framework* desarrollado por el grupo AYRNA [4] implementa una serie de algoritmos para la resolución de problemas de clasificación ordinal, complementándolos a su vez con diversas mejoras como paralelismo, varias métricas de rendimiento disponibles y la capacidad de programar ejecuciones automáticas. Este proyecto está implementado en MATLAB [5], siendo también compatible con GNU Octave [6].

Se utilizará el software de ORCA como referencia para el desarrollo de nuestro proyecto, obteniendo las mismas funcionalidades de este pero implementándolo en el lenguaje de programación Python [7], que es un lenguaje más moderno y con más proyección en el ámbito académico y laboral, lo que permitiría un mayor uso y propagación del *framework* dentro de la comunidad científica.

Orca-Python [8], es una adaptación y ampliación en Python del proyecto ORCA, realizado en un Trabajo de Fin de Grado anterior. Con este

framework se permite aprovechar las ventajas que ofrece el lenguaje de programación Python, así como la cantidad de paquetes y algoritmos que ya existen para este, además de fomentar una mayor extensión de este entre la comunidad científica.

En conclusión, este Trabajo de Fin de Grado se basará en la adaptación de dos de los algoritmos pertenecientes al proyecto original ORCA [3], en concreto, dos algoritmos que trabajan con redes neuronales, Red neuronal del modelo de probabilidades proporcionales, en inglés, *Proportional Odds Model Neural Network* (NNPOM) [9] y Red neuronal con particiones ordenadas, en inglés, *Neural Network with Ordered Partitions* (NNOP) [10] [1].

Capítulo 2

Definición del Problema

En este Capítulo se definirá de forma concisa el problema a solucionar. De esta forma, estableceremos y expondremos los objetivos de este. Este apartado del documento se dividirá en dos partes:

- La **identificación del problema real**, en la que se definirá el problema desde el punto de vista del usuario final.
- La **identificación del problema técnico**, en la que por el contrario, se definirá el problema desde el punto de vista del desarrollador del proyecto.

2.1. Identificación del Problema Real

El problema real, que es planteado por el grupo de investigación AYRNA, a través de los directores del Trabajo, es el siguiente:

Implementación de los dos algoritmos de redes neuronales para problemas de clasificación ordinal [1] en Python [7], previamente escritos en MATLAB [5]. Los algoritmos en cuestión son los siguientes: Red neuronal del modelo de probabilidades proporcionales, en inglés, *Proportional Odds Model Neural*

Network (NNPOM) [9] y Red neuronal con particiones ordenadas, en inglés, *Neural Network with Ordered Partitions* (NNOP) [10].

2.2. Identificación del Problema Técnico

Una vez identificado el problema real, a continuación, debemos de identificar el problema técnico, es decir, elaborar una descripción técnica y detallada del problema en cuestión. Para ello, nos serviremos de la metodología *Product Design Specification* (PDS) [11]. La PDS detalla los requisitos técnicos que deben cumplirse, respondiendo a una serie de preguntas fundamentales, para garantizar que el producto sea exitoso.

2.2.1. Funcionamiento

Dado que el cometido es adaptar dos algoritmos funcionales del *framework* ORCA [3], al *framework* Orca-Python [8], habrá algunas funciones que no serán necesarias realizar, como la interfaz o la representación de los datos obtenidos al final de la ejecución, que están ya implementadas en el nuevo *framework*, así como ciertas mecánicas restrictivas que habrá que respetar a la hora de implementar los algoritmos en cuestión. Es por eso que a continuación explicaremos el funcionamiento del nuevo entorno al ejecutar los algoritmos a desarrollar.

- Introducción. Previo a la ejecución de estos, habría que proporcionar cierta información para el funcionamiento de estos, en concreto, los archivos con las bases de datos que se vayan a usar para el experimento y la configuración de este.
- Base de datos. La primera parte de esta información consta de todos los archivos con las bases de datos pertinentes que se quieran utilizar en un experimento. Cada una de estas bases de datos se deberán almacenar en

un fichero de texto en formato CSV [12], todos ellos en una carpeta raíz cuya localización deberá estar especificada en el fichero de configuración del experimento.

- Archivo de configuración. Precisamente, la otra parte de la información requerida por el algoritmo es este fichero de configuración, que contendrá varios parámetros, como el citado directorio donde se almacenan las bases de datos para el experimento, si se requiere un prepocesamiento de datos, en cuyo caso sea afirmativo se especificará si se hará una estandarización o una normalización de los datos de entrenamiento, etc.

Al comenzar la ejecución del algoritmo, se ha de introducir el archivo de configuración que contendrá las instrucciones a seguir y los parámetros necesarios para el experimento. Al final de la ejecución, los resultados se almacenarán en la carpeta especificada por el usuario para su posterior visualización y estudio.

2.2.2. Entorno

Para el desarrollo de este proyecto se ha utilizado el Sistema Operativo (SO, de ahora en adelante), *Ubuntu 20.04.2 LTS* [13], siendo el *framework* Orca-Python [8] compatible con el SO.

Además, para el implementación y control de versiones de paquetes necesarios usados en el desarrollo se ha utilizado *Anaconda3 v. 4.9.2* [14], una conocida distribución de *Python* [7], que es muy utilizada en los campos de *Data Science* y *Machine Learning*.

Los usuarios finales, necesitarán un intérprete de órdenes para la visualización de los resultados de las pruebas así como para la ejecución de los distintos experimentos con los algoritmos se ha usado el intérprete *bash* [15].

2.2.3. Vida esperada

En este caso, el cálculo de la vida útil del producto es algo relativo y bastante complejo de calcular, debido a diversos factores.

Cabe destacar, dado que el origen del software pertenece al ámbito de la investigación, en el que la evolución y mejora del software es rápida y continua, hace pensar que la vida esperada del software será corta. Aun así, ese periodo de tiempo dependerá en gran medida de la velocidad con la que se consigan hallar soluciones resuelvan los problemas para los que los algoritmos en cuestión fueron diseñados. Estas soluciones pasan probablemente por el desarrollo de nuevos algoritmos que mejoren los resultados obtenidos con los implementados en este proyecto.

Por otro lado, también hay que tener en cuenta que el *framework* Orca-Python no es estático en cuanto a su desarrollo, por lo que bien podrían añadirse nuevos algoritmos o mejoras en los existentes que hagan que la vida útil de este sea incrementada.

2.2.4. Ciclo de mantenimiento

Este punto tiene una importante correlación con la Subsección 2.2.3, ya que la adaptación y mejora de los algoritmos ya presentes en el *framework*, así como las posibles inclusiones de nuevos algoritmos en favor del aumento de la vida útil de este, constarán a su vez como parte del ciclo de mantenimiento.

2.2.5. Competencia

Algunas de las competencias de Orca-Python [8] son su predecesor, ORCA [3] y la biblioteca de Pyhton `mord` [16]. A pesar de tener competencia, este *framework* tiene ciertas ventajas que hacen que sea mejor que otras librerías.

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 pueden diferenciar tres partes.

La primera, como no puede ser de otra manera, será la aplicación desarrollada. Además de está, se almacenará en la memoria del dispositivo físico en cuestión, el presente Manual técnico y de Usuario [17]. Además de este, se proporcionará un documento anexo a este, un Manual de Código, dirigido a los usuarios finales deseen realizar alguna consulta sobre parte o la totalidad del código, para el uso de este o para la realización de algún tipo de modificación/ampliación.

Ambos documentos tendrán una presentación clara, con ilustraciones y estarán redactados en un lenguaje poco técnico y de sencilla comprensión por parte del lector, todo esto para favorecer la asimilación de conceptos por parte del usuario final.

Por último, es recomendable señalar que el código de la aplicación estará totalmente disponible en un repositorio de la plataforma GitHub [18], que en conjunto con la herramienta Git [19], facilita el control de versiones y la posibilidad de colaboración durante el desarrollo de aplicaciones. Tanto Git como GitHub han sido usados durante todo el desarrollo del código por parte de los desarrolladores de este.

2.2.7. Estandarización

Para el proceso de estandarización del proyecto, el alumno desarrollador del software seguirá el estándar enseñado a lo largo del Grado en Ingeniería Informática, respectivo a la programación orientada a objetos, la declaración y nomenclatura de clases y de las distintas propiedades y partes que las conforman.

En lo referente al lenguaje de programación usado para el proyecto, es decir, Python, se seguirá la guía de estilo PEP8 [20]. Se seguirá la metodología especificada en esta guía para proveer de una mayor consistencia al software. Algunas de las indicaciones de está guía que han sido seguidas y que son destacables reseñar, respecto con la nomenclatura son:

- Las constantes son representadas en mayúsculas y con guiones bajos.
- Se seguirá la convención *CapsWords*, en la que el primer carácter del nombre de una clase/objeto será representada en mayúscula.
- Los nombres de métodos y variables correspondientes a una clase y que sean de carácter privado, comenzarán por un guión bajo.
- La primera letra de un nombre de una función, variable o paquete será minúscula.
- Los nombres de todos y cada uno de los elementos del código han de ser lo más descriptivo posible, para obtener el mayor entendimiento posible del código a simple vista.

2.2.8. Calidad y fiabilidad

La calidad del proyecto será supervisada y verificada tanto en primera instancia por el autor de este así como por los directores del mismo, que supervisarán la calidad del trabajo realizado por el autor y le asesorarán para asegurarla.

Posterior a la entrega de este, el propio proyecto pasará un control de calidad durante la defensa/lectura del mismo, frente al Tribunal designado para este propósito por la Escuela Politécnica Superior de Córdoba.

Por último, cabe destacar que durante el desarrollo del proyecto así como durante la ejecución de las pertinentes pruebas de los algoritmos implementados, se seguirán los estándares de creación de software asimilados durante

el Grado en Ingeniería Informática, que fueron explicitados en el apartado anterior, de forma que la aplicación de estas "buenas prácticas" en el desarrollo de software hagan más sencilla la tarea de proporcionar calidad al software creado.

En cuanto a la fiabilidad, se tendrá presente la inclusión de pruebas consistentes para comprobar que el flujo de ejecución que tenga lugar sea óptimo y se implementarán mensajes de error concisos que serán reproducidos en la terminal, para que el usuario sea consciente en caso de error del motivo por el cual ha tenido lugar este, ya sea por una errónea introducción de los argumentos previo a la ejecución o por otro motivo.

2.2.9. Programa de tareas

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

- **Fase de preparación.** Durante la primera fase o la más superficial respecto al desarrollo de software, se realiza un estudio teórico del problema, en el que se identifica y se estudian las metodologías, conceptos y conocimientos para abordarlo. En este caso, se tendrá que identificar y estudiar el problema en cuestion, que es desarrollar dos algoritmos de redes neuronales para la librería Orca-Python. Además, se tendrá que conocer el funcionamiento de la librería Orca-Python, así como el de su precursora, ORCA, para poder integrar los mencionados métodos.

  Es por ello, que se requerirán conocimientos tanto del lenguaje Octave/MATLAB como de Python y de los paquetes que se usarán de este último lenguaje de programación, entre otros, `numpy` [21], `scikit-learn` [22] y `pandas` [23].

- **Fase de diseño.** En esta segunda fase, tras un profundo estudio del problema y sus características, se procederá al diseño del sistema, de forma que se planteé este para una futura fase de codificación o implementación.

- **Fase de implementación.** A continuación, tiene lugar la fase de implementación, en la que se codificarán los algoritmos NNPOM (*Proportional Odds Model Neural Network*) y NNOP (*Neural Network with Ordered Partitions*) en Python [7], con las modificaciones que sean necesarias para que los algoritmos sean integrados dentro del código de Orca-Python y den un resultado óptimo.

- **Fase de pruebas.** En esta parte del programa se harán los ensayos y comprobaciones pertinentes al software desarrollado para comprobar que tanto el funcionamiento de los algoritmos es correcto y robusto así como su integración en la librería.

  Esta etapa no es una etapa aislada del resto, también está presente durante la fase de implementación, ya que durante esta fase es más fácil ir comprobando conforme se implementa que el funcionamiento del código es el esperado y a su vez es más fácil localizar los posibles errores surgidos en este y subsanarlos.

- **Fase de aplicación.** Tras la fase de pruebas, llega el turno de la fase de aplicación, en la que tendrán lugar los experimentos con distintos *datasets* para la comprobación del funcionamiento y del rendimiento de los algoritmos. Con los resultados obtenidos y el análisis de estos, tendrá lugar la conclusión del proyecto.

- **Fase de documentación.** En último lugar, nos encontramos con la fase de documentación, donde lo más apropiado es que esta fase tenga lugar a lo largo de todo el proyecto, y se vaya desarrollando de forma paralela conforme vamos pasando por el resto de fases.

  Esta etapa estará comprendida el desarrollo de esta memoria así como del manual anexo a esta memoria, el manual de código y de usuario.

2.2.10. Pruebas

En la etapa o fase de pruebas mencionada en el apartado anterior, tendrán lugar numerosas y distintas pruebas, para asegurar el funcionamiento y la robustez de los métodos añadidos al *framework* de Orca-Python añadido a la librería de Python ya existente.

Se realizarán pruebas unitarias a nivel de módulos para la comprobación del correcto funcionamiento de los algoritmos. También se harán pruebas de integración de los algoritmos a la librería para la verificación de la integración del nuevo código implementado con el resto de la librería. De igual forma, se comprobará que el proyecto cumpla los objetivos definidos para este y tendrán lugar algunas pruebas de estrés para comprobar que el funcionamiento es óptimo bajo una carga de trabajo a considerar.

Los resultados de todas estas pruebas serán comparados con los resultados obtenidos en pruebas similares realizadas con el proyecto ORCA [3].

2.2.11. Seguridad

Se verificará que el código implementado es seguro mediante el diseño de pruebas que verifiquen la integridad de los datos de entrada y que el manejo de grandes cantidades de datos en memoria también sea el adecuado.

Capítulo 3

Objetivos

El objetivo principal de este proyecto es migrar del *framework* ORCA [3], que está codificado en MATLAB [5] / Octave [6], los algoritmos NNPOM[9] y NNOP[10] al *framework* Orca-Python [8], el cual utiliza el lenguaje de programación Python [7].

Neural Network based on Proportional Odd Model, NNPOM: algoritmo basado en un modelo de posibilidades proporcionales (en inglés, *Proportional Odds Model*, POM), que implementa un modelo de red neuronal para la regresión ordinal. El modelo está compuesto por una capa oculta y una capa de salida, ésta última solo contiene una neurona de salida. La salida se divide por medio de umbrales, habiendo tantos como número de clases menos uno. En esta neurona, se aplica el modelo estándar POM, con la finalidad de proporcionar salidas probabilísticas.

Neural Network with Ordered Partitions, NNOP: algoritmo basado en el esquema de codificación de particiones ordenadas (NNOP), que implementa otro modelo alternativo de red neuronal para regresión ordinal. Este modelo considera el esquema de codificación de *OrderedPartitions* [1], para las etiquetas y una regla para las decisiones basadas en el primer nodo cuya salida es superior a un umbral predefinido (T = 0.5). El modelo tiene una capa oculta y una capa de salida con tantas neuronas como el número de clases

menos una.

3.1. Objetivos de formación

A continuación, en esta Sección del Capítulo 3, se citarán los objetivos de carácter formativo que tienen lugar en la realización del siguiente trabajo:

- Análisis del *framework* ORCA, para la posterior migración de los algoritmos al nuevo *framework* Orca-Python.
- Estudio del lenguaje MATLAB, para conocer y entender los diferentes principios y conceptos asociados al código del *framework* ORCA.
- Estudio teórico de los métodos NNPOM y NNOP.
- Análisis y estudio del presente *framework* Orca-Python.
- Estudio del lenguaje de programación Python, así como de la bibliotecas pertenecientes a este y que se usarán en la realización del trabajo, como son, `numpy` [21], `pandas` [23], `matplotlib` [24] y `scikit-learn` [22].

3.2. Objetivos operacionales

Tras revisar la Sección anterior, se mostrarán los objetivos operacionales, los cuales son una serie de requisitos que se han de cumplir para lograr el correcto funcionamiento y la mejora de este sobre el *framework* Orca-Python.

Se han de desarrollar dos nuevos algoritmos de clasificación ordinal basados en redes neuronales para el *framework* Orca-Python ya existente, dado que el objetivo final de este será migrar los algoritmos desde una biblioteca a otra.

Los algoritmos en cuestión a migrar desde el lenguaje de programación MATLAB [5] a Python [7] son NNPOM y NNOP, descritos al inicio del capítulo. Además de la migración de estos también se generarán archivos de prueba con los que comprobar el correcto funcionamiento de estos.

Capítulo 4

Antecedentes

4.1. Regresión ordinal

La regresión ordinal es un tipo de aprendizaje supervisado en el que las clases a predecir tienen cierta relación de orden entre sí. El orden referente a estas clases es una fuente de información sobre los patrones con relevancia de cara al proceso de aprendizaje. Es por eso, que como las clases están ordenadas, la penalización en el error de estas variará en función de la "lejanía"de la clase estimada con la clase a la que pertenece cada patrón.

4.1.1. Definición del problema

Los problemas de regresión ordinal, consisten en predecir el valor de una etiqueta y a partir de un conjunto de patrones pertenecientes a un espacio k-dimensional, $x \in X \subseteq \mathbb{R}^K$. La etiqueta pertenecerá a un conjunto de Q etiquetas o clases distintas, $y \in \gamma = C_1, C_2, ..., C_Q$. Para conseguir realizar la predicción mencionada se debe generar una regla $r : X \to \gamma$ a partir de un conjunto de patrones que tienen etiquetas conocidas. Esta regla debería de predecir las etiquetas de los patrones ya entrenados y además, tener la capacidad de predecir las etiquetas de nuevos patrones presentados.

Presentado esto, podemos observar que el procedimiento corresponde con el de problema de aprendizaje de clasificación corriente, pero al ser un problema de regresión ordinal habría que contemplar que las etiquetas tienen un orden, $C_1 \prec C_2 \prec ... \prec C_Q$ donde $\prec$ es la relación de orden entre las clases.

También, existen muchos algoritmos y métricas tienen en cuenta el orden de cada etiqueta en una escala ordinal. Esto se puede expresar por la función $\Theta(\cdot)$ de tal forma que $\Theta(C_q) = q$, $q = 1, ..., Q$.

4.1.2. Taxonomía de los métodos de regresión ordinal

El estudio el que nos hemos basado es el desarrollado por el grupo de investigación AYRNA [4] de la Universidad de Córdoba, el cual propone una taxonomía o clasificación de los distintos métodos de clasificación ordinal.

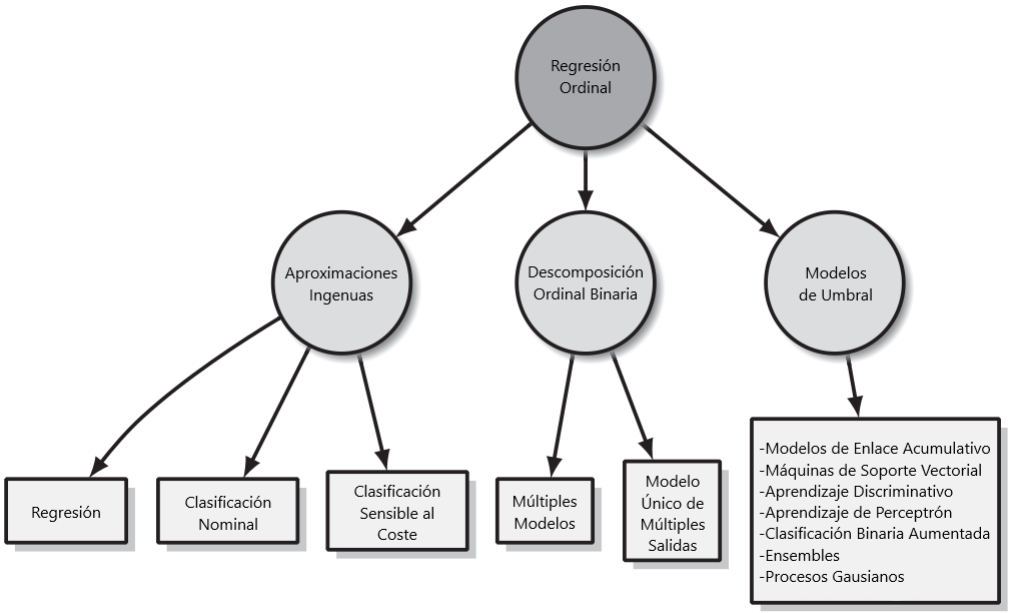

Figura 4.1: Taxonomía de métodos de regresión ordinal

Esta clasificación esta organizada en base a si los métodos realizan una descomposición del sistema, se basan en sistemas de valores o al propósito

para el que han sido diseñados. En la Figura 4.1 se muestra un esquema que representa esta taxonomía:

- **Aproximaciones ingenuas.** Una metodología para la resolución de problemas de regresión ordinal es simplificar estos problemas a problemas estándar, bajo la aceptación de supuestos teóricos que lo hacen posible. Los métodos que usan aproximaciones ingenuas, si los supuestos asumidos son aceptables, pueden llegar a ser muy competitivos, ya que su rendimiento será el correspondiente al de modelos sofisticados. Métodos de regresión, clasificación nominal y clasificación sensible al coste pertenecen a esta metodología.

- **Métodos de descomposición ordinal binaria.** Básicamente, esta metodología consiste en descomponer las etiquetas ordinales del problema de regresión en etiquetas binarias. Para la estimación de estas nuevas etiquetas se podrán aplicar uno o varios modelos. Múltiples modelos y modelo único de múltiples salidas son métodos de descomposición ordinal binaria.

- **Modelos de umbral.** Los métodos de umbral, están principalmente basados en la idea de que hay una variable continua latente, esto quiere decir, que esta variable contiene la respuesta del problema ordinal.

  Estos métodos son los más populares a la hora de abordar problemas de regresión ordinal. Algunos de ellos pueden ser: modelos de enlace acumulativo, máquinas de vectores soporte (cuyas siglas en inglés son *SVM* [25]), aprendizaje discriminatorio, aprendizaje del Perceptrón, clasificación binaria aumentada, *ensembles*, y los procesos gaussianos.

4.2. ORCA

ORCA (*Ordinal Regression and Classification Algorithms*) [3], es un *framework* de MATLAB desarrollado por el grupo de investigación AYRNA

[4], perteneciente al Dpto. de Informática y Análisis Numérico de la Escuela Politécnica Superior de Córdoba (EPSC), Universidad de Córdoba.

Este implementa e integra una amplia gama de métodos de regresión ordinal [1] y métricas de rendimiento del artículo *"Ordinal regression methods: Survey and experimental study"* publicado en *IEEE Transactions on Knowledge and Data Engineering* [26].

ORCA también ayuda a acelerar la comparación experimental del clasificador con la ejecución automática de diferentes *folds*, la paralelización de experimentos y los informes de rendimiento.

4.3. MATLAB

MATLAB (abreviatura de MATrix LABoratory) [5], es un sistema de cómputo numérico que ofrece un entorno de desarrollo integrado (IDE) con un lenguaje de programación propio (lenguaje M). Está disponible para las plataformas Unix, Windows, macOS y GNU/Linux [27].

Como se ha mencionado anteriormente, el proyecto ORCA está hecho en el lenguaje de programación M, perteneciente a este IDE.

4.4. Orca-Python

Orca-Python [8], es un *framework* escrito en el lenguaje de programación Python, completamente integrado con los módulos `scikit-learn` y `sacred`, cuyo objetivo es el de automatizar la ejecución de experimentos de *machine learning* utilizando ficheros de configuración fáciles de entender. Fue desarrollado como Trabajo de Fin de Grado por el alumno Iván Bonaque Muñoz[8].

Este *framework* es compatible con cualquier algoritmo que se encuentre implementado en `scikit-learn` o bien creado por el usuario siempre que

siga las reglas de compatibilidad con dicha biblioteca.

La herramienta en cuestión, implementa cuatro métodos de descomposición ordinal:

- *Ordered partitions.*
- *One vs next.*
- *One vs previous.*
- *One vs follower.*

Además posee las métricas de rendimiento del *framework* en el que está basado, ORCA.

4.5. Python

Python [7], es un lenguaje de programación interpretado dinámico y multiplataforma, de propósito general, cuya filosofía hace hincapié en la legibilidad de su código. Este es un lenguaje de programación multiparadigma, es decir, que está basado en varios paradigmas de programación, como son: la orientación a objetos, la cual soporta parcialmente, la programación imperativa y la programación funcional, aunque está última en menor medida que las anteriores.

La licencia de este lenguaje de programación, *Python Software Foundation License*, es de código abierto, uno de los muchos motivos de su creciente popularidad tanto en la comunidad científica, como a nivel empresarial y de usuario/desarrollador medio. Otros de los motivos de su auge es que es fácilmente legible, ya que forma parte de su filosofía, su facilidad de aprendizaje por parte de nuevos usuarios y la capacidad que posee de ampliación, mediante sus numerosas bibliotecas.

Centrándonos en la capacidad de ampliación debida a sus diversas bibliotecas, nos encontramos con muchas de estas dedicadas a facilitar la realización de tareas en el ámbito científico de las Ciencias de la Computación.

A continuación, en las siguientes Subsecciones, se introducirán algunas de estas bibliotecas, que han sido usadas durante el desarollo del proyecto.

4.5.1. Matplotlib

`Matplotlib` [24], es una biblioteca del lenguaje de programación Python, cuya finalidad es la generación de gráficos a través de datos almacenados. Esta biblioteca posee integración con la biblioteca `Numpy`.

Además, proporciona una API, pylab, cuyo diseño recuerda a la de MATLAB.

4.5.2. Numpy

`Numpy` [21] es un proyecto de código abierto que tiene como objetivo permitir la computación númerica con Python. Esta biblioteca está publicada bajo una licencia de código abierto BSD modificada.

4.5.3. Pandas

`Pandas` [23] es una biblioteca de software desarrollada como extensión de la biblioteca Numpy para la manipulación y el análisis de datos, dentro del lenguaje de programación Python. Ofrece estructuras de datos y operaciones para manipular tablas numéricas y series temporales.

Esta biblioteca se distribuye bajo una licencia de software libre BSD modificada.

4.5.4. Scikit-learn

`Scikit-learn` [22] es una biblioteca para aprendizaje automático para el lenguaje de programación Python, programada sobre las bibliotecas `Numpy`, `Matplotlib` y `Scipy`.

Incluye varios algoritmos de clasificación, regresión y análisis de grupos (*clustering*) y prepocesamiento. Está diseñada para operar junto con las bibliotecas `Numpy` y `SciPy`.

Posee una licencia de código abierto BSD [28].

4.5.5. Scipy

`Scipy` [29] es una biblioteca libre y de código abierto para Python. Se compone de herramientas y algoritmos matemáticos.

Capítulo 5

Restricciones

En este Capítulo, se muestran todas las restricciones de diseño que a lo largo del proyecto, condicionan a las fases de desarrollo antes vistas. Estos factores se pueden clasificar en dos grupos: los factores dato y los factores estratégicos.

A continuación, se explicará brevemente en qué consisten estos factores y las restricciones que los componen:

- **Factores dato.** Los factores dato están compuestos por todas las restricciones inherentes al proyecto y que vienen dadas por el cliente, que en este caso vendría a ser el grupo de investigación AYRNA [4], perteneciente a la UCO.
- **Factores estratégicos.** Los factores estratégicos son las restricciones que el desarrollador debe satisfacer durante el diseño, generalmente a través de la elección de algún tipo de herramienta/software que elimina por tanto la posibilidad de usar otra alternativa. Un ejemplo bastante fácil de comprender sería la elección del tipo de editor de texto que se usará para redactar el documento actual. En nuestro caso, hemos optado por usar LaTeX [30], dentro de la plataforma on-line *OverLeaf* [31].

Tras está breve introducción a los dos tipos de factores que limitan el proyecto. A continuación se expondrán todos los factores de cada tipo que componen este proyecto.

5.1. Factores dato

Los factores dato son un grupo restricciones inherentes al proyecto que se han de cumplir durante su realización. Estas restricciones se podrían dividir en cinco grupos a la hora de describirla, como se hará a continuación:

- **Restricciones hardware.** El autor del proyecto usará su equipo personal, por lo que las restricciones del hardware vendrán determinadas por las condiciones de este equipo, sin que ello llegue en ningún momento a perjudicar la calidad del proyecto.

- **Restricciones software.** Se deben de adaptar dos algoritmos de clasificación ordinal con redes neuronales del lenguaje de programación M [5] al lenguaje Python [7], pudiéndose ejecutar desde el propio *framework* Orca-Python [8], y, siendo este compatible. Por todo esto, se hará uso de todas las herramientas software y lenguajes de programación necesarios para el desarrollo y ejecución óptimo del proyecto.

- **Restricciones temporales.** Estas restricciones vendrán establecidas por el tiempo que será necesario utilizar por parte del autor del proyecto para desarrollado con la calidad suficiente, teniendo como objetivo personal del autor que la entrega tenga lugar antes de la finalización del curso presente, es decir, 2020/21.

- **Restricciones humanas.** Dado que el proyecto es un Trabajo de Fin de Grado universitario, esta restricción vendrá determinada por colaboración de los directores del mismo, así como del grupo de investigación al que pertenecen.

- **Restricciones económicas.** Al ser un proyecto informático de la especialidad de Computación, estas restricciones vendrán dadas por el precio resultante del uso del software empleado en este, por lo que se ha primado la utilización de Software Libre en la medida de lo posible y en su defecto, se empleará software de carácter gratuito.

5.2. Factores estratégicos

Los factores estratégicos corresponden a las restricciones impuestas para el proyecto por parte del alumno y sus directores de proyecto.

En primer lugar, indicar la utilización del gestor de paquetes `Conda` [14], mediante el cual es posible una fácil instalación, ejecución y actualización de los paquetes necesarios para el proyecto a través de la creación especifica de un entorno o *enviroment*.

Como editor de código se utilizará Visual Studio Code, cuya abreviatura es VSCode [32]. Este editor de código, que ha alcanzado gran popularidad en los últimos meses dentro del entorno del desarrollo de software, destaca por su facilidad de uso y la virtud de que en él se puede programar con casi cualquier lenguaje de programación. Posee numerosos complementos, como paquetes de *IntelliSense*, depuración y ejecución de la mayoría de lenguajes de programación, así como de un terminal integrado para la ejecución del código.

Además de los paquetes ya mencionados en el Capítulo 4, para la realización de las pruebas se usará el paquete `unittest` [33], que permite la creación de pruebas dentro del código y la agrupación de estas.

Dentro del proyecto, también se hace uso de la herramienta Make [34], para la simplificación de la instalación y compilación del *framework* Orca-Python.

Para el control de versiones tanto de forma local como de forma on-line,

se utilizará la herramienta Git [19] acompañada de GitHub [18], no tan solo para el control de versiones y la publicación de los avances en el *framework*, sino también para facilitar la revisión de este durante la fase de desarrollo por parte de los directores del proyecto.

Para la implementación de los documentos, manuales y/o anexos del proyecto, se usará la herramienta LaTeX [30], cuya principal ventaja encontrada por parte del proyectista es la facilidad de generación de textos, inclusión de figuras y tablas, índices dinámicos y gestión del diseño gráfico de los documentos (márgenes, estilos, bibliografía, etc.).

El desarrollo de los diagramas de casos de uso y de clases se llevará a cabo con la herramienta online GenMyModel [35], de la que tenemos acceso a través de internet y de forma gratuita.

Por último, para la edición de imágenes y figuras, se usará la herramienta de Sofware Libre que incluye el Sistema Operativo (SO) Ubuntu, *GNU Image Manipulation Program, GIMP* [36].

Capítulo 6

Recursos

En esta Sección se detallan los recursos usados por el autor del proyecto durante el desarrollo de este. Estos recursos se pueden agrupar en tres clases bien diferenciadas: recursos hardware, software y humanos.

6.1. Recursos hardware

Señalar que para el desarrollo del proyecto se ha usado el equipo personal del autor, dicho equipo es un ordenador personal de sobremesa de las siguientes características:

- Procesador: Intel(R) Core(TM) i7-3770 CPU 3.40GHz.
- Memoria RAM: Memoria RAM de 8GB DDR3 a 1333 MT/s
- Tarjeta gráfica: Nvidia GeForce GTX 750 con 1GB GDDR5 a 5,012 GHz.
- Disco duro: Disco duro ATA Disk Samsung SSD 840 (120GB).

6.2. Recursos software

- **Sistema Operativo (SO):** Ubuntu 20.04.2 LTS
- **Python:** Python 3.7.8.
- **Octave:** Octave 4.2.1.
- **Editor de código:** Visual Studio Code 1.49.3.
- **Herramienta de documentos de textos:** Overleaf, editor y entorno online para la elaboración de una documentación en LaTeX.
- **Herramienta de edición de imágenes:** GNU Image Manipultion Program, GIMP. Versión 2.10.18.
- **Herramienta de modelado:** GenMyModel, plataforma de modelado online. Usado para el diseño de diagramas de clases y casos de uso.

6.3. Recursos humanos

- Autor: **Adrián López Ortiz.** Alumno del grado de Ingeniería Informática con mención en computación, en la Escuela Politécnica Superior de la Universidad de Córdoba (EPSC).
- Directores:
  Pedro Antonio Gutiérrez Peña. Profesor Titular de la Universidad de Córdoba del Dpto. de Informática y Análisis Numérico y miembro investigador del grupo AYRNA [4] y encargado de dar soporte técnico y teórico para las tareas de análisis, diseño y programación.

  David Guijo Rubio. Profesor Sustituto Interino de la Universidad de Córdoba del Dpto. de Informática y Análisis Numérico y miembro investigador del grupo AYRNA [4] 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

En este Capítulo se ofrecerá una vista con más detalle de los requisitos necesarios para el desarrollo del presente proyecto. Como se vió en Capítulos anteriores del documento, el objetivo primordial del proyecto es la adaptación de dos funciones de aprendizaje automático con redes neuronales al *framework* Orca-Python [8], migrando estos métodos ya existentes en el *framework* anterior ORCA [3].

Estos métodos ya mencionados con anterioridad de regresión ordinal [1] son NNPOM [9] y NNOP [10], que están implementados en el lenguaje M (MATLAB / Octave) [5] y que se han de migrar al lenguaje de programación Python [7] a través del uso de bibliotecas propias de este lenguaje como `Numpy` [21], `Scipy` [29] y `Scikit-learn` [22], entre otras.

Todo lo mencionado en los dos párrafos anteriores será supervisado durante su desarrollo por el grupo de investigación AYRNA [4].

7.2. Participantes del trabajo

Los participantes del trabajo de fin de grado serán los dos directores de este y el alumno que realizará el proyecto bajo su supervisión, cuya información ya mostrada en el Capítulo 6 del presente trabajo será ampliada en las Tablas mostradas a continuación:

Participante	Pedro Antonio Gutiérrez Peña
Organización	AYRNA
Rol	Director
Desarrollador	No
Cliente	Si
Usuario	Si
Comentarios	Profesor Titular de la Universidad de Córdoba del Dpto. de 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.

Tabla 7.1: Director Pedro Antonio Gutiérrez Peña

Participante	Divid Guijo Rubio
Organización	AYRNA
Rol	Director
Desarrollador	No
Cliente	Si
Usuario	Si
Comentarios	Profesor Sustituto Interino de la Universidad de Córdoba del Dpto. de 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.

Tabla 7.2: Director David Guijo Rubio

Participante	Adrián López Ortiz
Organización	EPS UCO
Rol	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 Córdoba, cuyo rol será el de desarrollador del software del proyecto

Tabla 7.3: Alumno Adrián López Ortiz

7.3. Catálogo de objetivos del sistema

A continuación, en esta Sección mostraremos una Tabla para cada objetivo del sistema a conseguir:

OBJ-001	Desarrollo de una biblioteca software en Python
Versión	1.0
Autores	Adrián López Ortiz
Fuentes	Pedro Antonio Gutiérrez Peña David Guijo Rubio
Descripción	Desarrollo de parte de una biblioteca software en Python que implemente distintos algoritmos de clasificación ordinal. Se migrarán dos de los clasificadores de regresión ordinal, más concretamente los de redes neuronales, pertenecientes al *framework* ORCA.
Subobjetivos	OBJ-002 Migración del clasificador NNPOM OBJ-003 Migración del clasificador NNOP OBJ-005 Facilidad de uso
Comentarios	Ninguno.

Tabla 7.4: Objetivo relativo al desarrollo de parte de la biblioteca software Orca-Python.

OBJ-002	Migración del clasificador NNPOM
Versión	1.0
Autores	Adrián López Ortiz
Fuentes	Pedro Antonio Gutiérrez Peña David Guijo Rubio
Descripción	Efectuar la migración al lenguaje de programación Python, con los cambios que sean necesarios en el código del algoritmo NNPOM para permitir su ejecución desde el *framework* Orca-Python.
Subobjetivos	Ninguno.
Comentarios	El código del algoritmo se implementará de forma que sea compatible con la librería `scikit-learn`.

Tabla 7.5: Objetivo relativo a la migración del código del clasificador NNPOM al *framework* Orca-Python.

OBJ-003	Migración del clasificador NNOP
Versión	1.0
Autores	Adrián López Ortiz
Fuentes	Pedro Antonio Gutiérrez Peña David Guijo Rubio
Descripción	Efectuar la migración al lenguaje de programación Python, con los cambios que sean necesarios en el código del algoritmo NNOP para permitir su ejecución desde el *framework* Orca-Python.
Subobjetivos	Ninguno.
Comentarios	El código del algoritmo se implementará de forma que sea compatible con la librería `scikit-learn`.

Tabla 7.6: Objetivo relativo a la migración del código del clasificador NNOP al *framework* Orca-Python.

OBJ-004	Facilidad de uso
Versión	1.0
Autores	Adrián López Ortiz
Fuentes	Pedro Antonio Gutiérrez Peña David Guijo Rubio
Descripción	Facilitar en la medida de lo posible el uso de los algoritmos implementados para el *framework* ORCA-Python.
Subobjetivos	Ninguno.
Comentarios	Ninguno.

Tabla 7.7: Objetivo relativo a mantener la facilidad de uso de los algoritmos implementados dentro del *framework* ORCA-Python

7.4. Catálogo de requisitos del sistema

7.4.1. Requisitos de información

El *framework* Orca-Python se encargará del tratamiento de toda la información que se procesa con los algoritmos desarrollados. Los requisitos de información serán definidos a continuación en una serie de Tablas:

IRQ-001	Configuración de parámetros de NNPOM
Versión	1.0
Autores	Adrián López Ortiz
Fuentes	Pedro Antonio Gutiérrez Peña David Guijo Rubio
Dependencias	OBJ-002 Migración del clasificador NNPOM
Descripción	Los parámetros del algoritmo NNPOM son: ▪ **epsilonInit**. Rango de inicialización de los pesos del modelo de red neuronal. ▪ **iter.** Número de iteraciones para el algoritmo de retropropagación del error. ▪ **hiddenN**. Número de neuronas que componen la capa oculta del modelo. ▪ **lambda.** Parámetro de regularización.
Comentarios	Ninguno.

Tabla 7.8: Requisito de información relativo a la configuración de los parámetros del algoritmo NNPOM

IRQ-002	Configuración de parámetros de NNOP
Versión	1.0
Autores	Adrián López Ortiz
Fuentes	Pedro Antonio Gutiérrez Peña David Guijo Rubio
Dependencias	OBJ-003 Migración del clasificador NNOP
Descripción	Los parámetros del algoritmo NNOP son: ▪ **epsilonInit**. Rango de inicialización de los pesos del modelo de red neuronal. ▪ **iter.** Número de iteraciones para el algoritmo de retropropagación del error. ▪ **hiddenN**. Número de neuronas que componen la capa oculta del modelo. ▪ **lambda.** Parámetro de regularización.
Comentarios	Ninguno.

Tabla 7.9: Requisito de información relativo a la configuración de los parámetros del algoritmo NNOP

7.4.2. Requisitos funcionales

Esta Subsección versa sobre los requisitos funcionales del proyecto, los cuales determinarán el comportamiento de los algoritmos a implementar así como las mejoras a desarrollar en estos. Estos requisitos se encuentran recogidos uno a uno a en Tablas individuales que se mostrarán en esta Sección.

FRQ-001	Entrenamiento del clasificador NNPOM
Versión	1.0
Autores	Adrián López Ortiz
Fuentes	Pedro Antonio Gutiérrez Peña David Guijo Rubio
Objetivos Asociados	OBJ-001 Desarrollo de una biblioteca software en Python OBJ-002 Migración del clasificador NNPOM
Requisitos Asociados	Ninguno.
Descripción	El *framework* Orca-Python será utilizado para realizar el entrenamiento del modelo de NNPOM mediante el uso de una clase local, implementada por el desarrollador del proyecto, donde estará definido el modelo y el entrenamiento. De esta forma se ejecutará el algoritmo con los parámetros seleccionados por el usuario para entrenar el modelo añadiendo también por su parte los *datasets* de entrenamiento que haya considerado. El modelo resultante se almacenará en la clase para usarlo en futuras ejecuciones con el objetivo de predecir resultados.
Comentarios	La clase local donde se encontrará definido NNPOM cumplirá con el estándar especificado por la documentación de `scikit-learn`.

Tabla 7.10: Requisito funcional relativo al proceso de entrenamiento del clasificador NNPOM

FRQ-002	Fase de predicción del clasificador NNPOM
Versión	1.0
Autores	Adrián López Ortiz
Fuentes	Pedro Antonio Gutiérrez Peña David Guijo Rubio
Objetivos Asociados	OBJ-001 Desarrollo de una biblioteca software en Python OBJ-002 Migración del clasificador NNPOM
Requisitos Asociados	FRQ-001 Entrenamiento del clasificador NNPOM
Descripción	La fase de predicción del clasificador en cuestión, al igual que el entrenamiento se realizará a través de una clase local, perteneciente al *framework* Orca-Python. En esta clase tras realizar el entrenamiento y haber obtenido el modelo de red con los pesos ya ajustados, este se cargará durante la fase de predicción y recibirá los parámetros y el *dataset* de test con los patrones a clasificar mediante el archivo de configuración del *framework*.
Comentarios	La clase local donde se encontrará definido NNPOM cumplirá con el estándar especificado por la documentación de `scikit-learn`.

Tabla 7.11: Requisito funcional relativo a la fase de predicción del clasificador NNPOM

FRQ-003	Entrenamiento del clasificador NNOP
Versión	1.0
Autores	Adrián López Ortiz
Fuentes	Pedro Antonio Gutiérrez Peña David Guijo Rubio
Objetivos Asociados	OBJ-001 Desarrollo de una biblioteca software en Python OBJ-003 Migración del clasificador NNOP
Requisitos Asociados	Ninguno.
Descripción	El *framework* Orca-Python será utilizado para realizar el entrenamiento del modelo de NNOP mediante el uso de una clase local, implementada por el desarrollador del proyecto, donde estará definido el modelo y el entrenamiento. De esta forma se ejecutará el algoritmo con los parámetros seleccionados por el usuario para entrenar el modelo añadiendo también por su parte los *datasets* de entrenamiento que haya considerado. El modelo resultante se almacenará en la clase para usarlo en futuras ejecuciones con el objetivo de predecir resultados.
Comentarios	La clase local donde se encontrará definido NNOP cumplirá con el estándar especificado por la documentación de `scikit-learn`.

Tabla 7.12: Requisito funcional relativo al proceso de entrenamiento del clasificador NNOP

FRQ-004	Fase de predicción del clasificador NNOP
Versión	1.0
Autores	Adrián López Ortiz
Fuentes	Pedro Antonio Gutiérrez Peña David Guijo Rubio
Objetivos Asociados	OBJ-001 Desarrollo de una biblioteca software en Python OBJ-003 Migración del clasificador NNOP
Requisitos Asociados	FRQ-003 Entrenamiento del clasificador NNOP
Descripción	La fase de predicción del clasificador en cuestión, al igual que el entrenamiento se realizará a través de una clase local, perteneciente al *framework* Orca-Python. En esta clase tras realizar el entrenamiento y haber obtenido el modelo de red con los pesos ya ajustados, este se cargará durante la fase de predicción y recibirá los parámetros y el *dataset* de test con los patrones a clasificar mediante el archivo de configuración del *framework*.
Comentarios	La clase local donde se encontrará definido NNOP cumplirá con el estándar especificado por la documentación de `scikit-learn`.

Tabla 7.13: Requisito funcional relativo a la fase de predicción del clasificador NNOP

7.4.3. Requisitos no funcionales

En esta Subsección se mostrarán a través de Tablas individuales para cada requisito, todo los requisitos no funcionales del proyecto.

NFR-001	Migración del clasificador NNPOM al lenguaje de programación Python
Versión	1.0
Autores	Adrián López Ortiz
Fuentes	Pedro Antonio Gutiérrez Peña David Guijo Rubio
Dependencias	OOBJ-002 Migración del clasificador NNPOM
Descripción	El clasificador NNPOM se desarrolla en el lenguaje de programación Python al igual que el *framework*, por lo que todas las adaptaciones realizadas del clasificador respecto al proyecto ORCA se han de realizar en este lenguaje de programación.
Comentarios	Ninguno.

Tabla 7.14: Requisito no funcional relativo a la migración del clasificador NNPOM.

NFR-002	Migración del clasificador NNOP al lenguaje de programación Python
Versión	1.0
Autores	Adrián López Ortiz
Fuentes	Pedro Antonio Gutiérrez Peña David Guijo Rubio
Dependencias	OOBJ-003 Migración del clasificador NNOP
Descripción	El clasificador NNOP se desarrolla en el lenguaje de programación Python al igual que el *framework*, por lo que todas las adaptaciones realizadas del clasificador respecto al proyecto ORCA se han de realizar en este lenguaje de programación.
Comentarios	Ninguno.

Tabla 7.15: Requisito no funcional relativo a la migración del clasificador NNOP.

NFR-003	Compatibilidad del clasificador NNPOM con Orca-Python
Versión	1.0
Autores	Adrián López Ortiz
Fuentes	Pedro Antonio Gutiérrez Peña David Guijo Rubio
Dependencias	OBJ-002 Migración del clasificador NNPOM
Descripción	El clasificador NNPOM tendrá que tener comptibilidad con el proyecto en sí, es decir, con el *framework* Orca-Python y tener en consideración la estructura de este en diversos niveles, como el código, la lógica, la detección y gestión de errores y la presentación de la misma.
Comentarios	El clasificador deberá ser compatible con `scikit-learn`.

Tabla 7.16: Requisito no funcional relativo a la compatibilidad del clasificador NNPOM con Orca-Python.

NFR-004	Compatibilidad del clasificador NNOP con Orca-Python
Versión	1.0
Autores	Adrián López Ortiz
Fuentes	Pedro Antonio Gutiérrez Peña David Guijo Rubio
Dependencias	OBJ-003 Migración del clasificador NNOP
Descripción	El clasificador NNOP tendrá que tener comptibilidad con el proyecto en sí, es decir, con el *framework* Orca-Python y tener en consideración la estructura de este en diversos niveles, como el código, la lógica, la detección y gestión de errores y la presentación de la misma.
Comentarios	El clasificador deberá ser compatible con `scikit-learn`.

Tabla 7.17: Requisito no funcional relativo a la compatibilidad del clasificador NNOP con Orca-Python.

NFR-005	Usabilidad
Versión	1.0
Autores	Adrián López Ortiz
Fuentes	Pedro Antonio Gutiérrez Peña David Guijo Rubio
Dependencias	OBJ-004 Facilidad de uso
Descripción	Facilitar, a través de los comentarios del código y manuales disponibles, el uso por parte del usuario de los algoritmos implementados, intentando que su uso sea lo más intuitivo posible.
Comentarios	Ninguno.

Tabla 7.18: Requisito no funcional relativo a la usabilidad de los clasificadores adaptados en Orca-Python.

NFR-006	Tolerancia a fallos y optimización
Versión	1.0
Autores	Adrián López Ortiz
Fuentes	Pedro Antonio Gutiérrez Peña David Guijo Rubio
Dependencias	OBJ-001 Desarrollo de una biblioteca software en Python
Descripción	Los clasificadores implementados y las posibles mejoras introducidas ellos dentro del *framework* Orca-Python, deberán tolerar los fallos. Deberán de detectar los errores introducidos por el usuario y reportarlos de forma óptima por medio de la terminal.
Comentarios	Ninguno.

Tabla 7.19: Requisito no funcional relativo a la tolerancia a fallos y la optimización por parte de los nuevos calsificadores implementados al *framework* Orca-Python.

NFR-007	Desarrollo en inglés
Versión	1.0
Autores	Adrián López Ortiz
Fuentes	Pedro Antonio Gutiérrez Peña David Guijo Rubio
Dependencias	OBJ-001 Desarrollo de una biblioteca software en Python
Descripción	Todo el desarrollo del código y los comentarios introducidos en este será en ingles, de forma que se facilite la comprensión y uso del software de forma internacional.
Comentarios	Ninguno.

Tabla 7.20: Requisito no funcional relativo al idioma que se utilizará, en este caso inglés, en el desarrollo del código introducido a Orca-Python.

7.5. Matriz de rastreabilidad

Esta es la última Sección del Capítulo, está compuesta por la matriz de rastreabilidad mostrada en la Tabla 7.21, la cual facilita la visión sobre qué objetivos del sistema están relacionados con cada uno de los requisitos del sistema.

TRM - 001	**OBJ - 001**	**OBJ - 002**	**OBJ - 003**	**0BJ - 004**
IRQ - 001		X		
IRQ - 002			X	
FRQ - 001	X	X		
FRQ - 002	X	X		
FRQ - 003	X		X	
FRQ - 004	X		X	
NFR -001		X		
NFR -002			X	
NFR -003		X		
NFR -004			X	
NFR -005				X
NFR -006	X			
NFR -007	X			

Tabla 7.21: Matriz de rastreabilidad

Parte III

Análisis y Diseño del Sistema

Capítulo 8

Casos de uso

En este Capítulo 8, se especificarán los casos de uso. Estos muestran todas las interacciones que tendrán los diferentes actores del sistema de forma gráfica. La definición de un caso de uso puede estar comprendido por los siguientes componentes: actores, relaciones, casos de uso y límites del sistema.

- **Actores:** Persona o elemento que interacciona con el sistema, de manera que cada actor desempeñará un rol en este.

- **Relaciones:** Tipo de interacción existente entre los actores y el sistema.

- **Casos de uso:** Acciones que pueden realizar los actores dentro de un caso de uso.

- **Límites del sistema:** Los límites del sistema agrupa uno o más casos de uso dentro de un sistema o un subsistema.

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

Este es el primer caso de uso del proyecto software. Este caso de uso muestra el funcionamiento completo del sistema. Para ello, hay que tener en cuenta las siguientes características:

Solo hay un actor, en este caso el usuario, que usa el sistema para realizar un experimento. Dado que los algoritmos que se implementarán en este proyecto se incluirán en la biblioteca o proyecto Orca-Python, serán utilizados a través de la interfaz proporcionada por este. Esto quiere decir, que todos los proyectos que tengan como meta desarrollar algoritmos de regresión ordinal para esta biblioteca, heredarán del proyecto Orca-Python los casos de uso, siendo coincidentes.

Como fue especificado en otros Capítulos, se crearán ficheros de configuración a través de los cuales se podrán aplicar distintos clasificadores/algoritmos sobre uno o varios *datasets* en un mismo experimento, de forma que se genere para cada combinación de estos un modelo ajustado. Esto se podría observar como que se están haciendo varios experimentos en una misma ejecución, de manera que se obtiene para cada experimento (clasificador-*dataset*) un modelo de ese clasificador optimizado.

En el diagrama 8.1 y en la Tabla 8.1, en la que se especifica el Caso de Uso 0, (CU-0) .

8.2. Caso de Uso 1. Ejecutar Experimento

El usuario que gestione el experimento tendrá como primera tarea configurar el fichero de configuración correspondiente al experimento, que una vez configurado, será el encargado de indicar los parámetros correspondientes a este, como el/los algoritmos a utilizar, la configuración de estos así como los

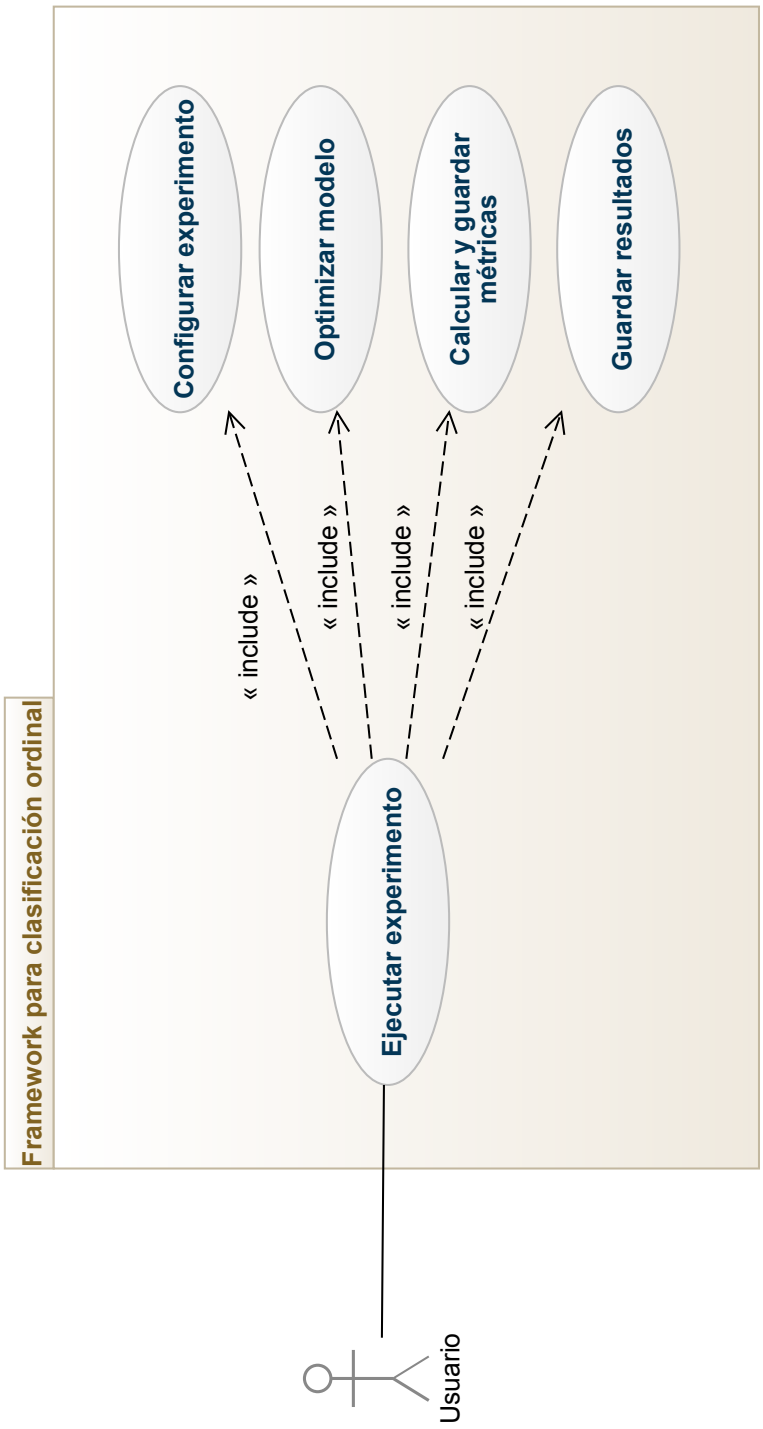

Figura 8.1: Caso de uso CU-0: *Framework* para Clasificación Ordinal

Caso de Uso	CU-0. *Framework* para Clasificación Ordinal.
Descripción	Sistema *software* que simplifica la realización de experimentos de *machine learning* utilizando archivos de configuración de fácil comprensión.
Actores	Usuario
Casos de Uso	▪ CU-1 Ejecutar Experimento.

Tabla 8.1: Especificación del caso de uso CU-0. *Framework* para Clasificación Ordinal

conjuntos de datos con los que estos algoritmos trabajarán.

Al finalizar el experimento, los resultados que se hayan obtenido serán guardados en ficheros. En la Tabla 8.2 se especifica este caso de uso.

Caso de Uso	CU-1. Ejecutar experimento.
Descripción	El usuario realiza un experimento con las pautas introducidas en el archivo de configuración del experimento de forma previa.
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.

Tabla 8.2: Especificación del caso de uso CU-1: Ejecutar experimento

8.2.1. Caso de Uso 1-1. Configurar Experimento

Si el usuario desea ejecutar un experimento, previamente deberá crear un fichero de configuración en el que se especificarán los algoritmos de clasificación y los parámetros de estos, así como los conjuntos de datos sobre los que aplicar dichos clasificadores.

La configuración del fichero consta de dos partes, una general y otra especifica de cada clasificador, explicadas con más detalle en el Anexo A.

Este caso de uso se detalla en la Tabla 8.3.

8.2.2. Caso de Uso 1-2. Optimizar modelo

Para conseguir los parámetros que conformen el modelo óptimo para cada algoritmo se usará la *cross-validation* de tipo *k-fold* (siendo k determinada por el usuario en la configuración general, dentro del archivo de configuración), considerando únicamente los datos de entrenamiento.

Este procedimiento es repetido por cada partición de cada *dataset*. Este caso de uso se detalla en la Tabla 8.4.

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

Tras haber determinado los modelos óptimos de cada clasificador elegido para cada conjunto de datos especificado de forma previa en el archivo de configuración del experimento, tendrá lugar el cálculo de las métricas de rendimiento especificadas para también en el archivo de configuración para cada uno de ellos.

También, se realizará un cálculo de las medias y desviaciones típicas a lo largo de las particiones de las métricas de *train* y *test*.

Este caso de uso se detalla en la Tabla 8.5.

Caso de Uso	CU-1-1. Configurar experimento.
Descripción	El usuario introduce en el fichero de configuración toda la información necesaria para la ejecución del experimento.
Actores	Usuario
Condiciones Iniciales	Ninguna
Flujo Principal	1. El usuario indica la configuración general del experimento, compuesta por: los *datasets*, el número de *folds* con los que realizar la validación cruzada (*cross-validation*), las métricas a calcular, el tipo de preprocesamiento a realizar y el número de trabajos a utilizar por el proceso. 2. El usuario indica los distintos algoritmos con los que clasificar los *datasets*, indicando los hiperparámetros con los que quiere optimizar los modelos durante la fase de *cross-validation*.
Condiciones Finales	Ninguna
Flujos Alternativos	Ninguno

Tabla 8.3: Especificación del caso de uso CU-1-1. Configurar experimento

Caso de Uso	CU-1-2. Optimizar modelo.
Descripción	Usando el archivo de configuración creado por el usuario, se obtiene el mejor modelo de cada algoritmo usado, adecuando los parámetros de estos para obtener un modelo óptimo para cada conjunto de datos usado en el experimento. El método utilizado es el de la validación cruzada 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 el adecuado. Se especifica más acerca de dicho formato en el Anexo A. ▪ La sintaxis y la información del fichero de configuración deben ser correctas.
Flujo Principal	1. Se cargan las diferentes bases de datos especificadas en el archivo de configuración por particiones. 2. Se recorren todas las bases de datos partición a partición. 3. Para cada clasificador se determinan los mejores parámetros por partición mediante validación cruzada.
Condiciones Finales	Se obtienen los mejores modelos por partición.
Flujos Alternativos	El proceso finaliza debido a un error.

Tabla 8.4: Especificación del caso de uso CU-1-2. Optimizar modelo.

Caso de Uso	CU-1-3. Calcular y guardar métricas.
Descripción	Una vez determinados los mejores modelos por 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 indicadas el archivo de configuración para cada uno de los modelos óptimos de cada partición realizada en el experimento. 2. Para cada *dataset*, se toman las métricas de rendimiento de cada modelo (uno por partición) y se determina su media y desviación típica.
Condiciones Finales	▪ Las métricas especificadas en el archivo de configuración son calculadas para cada modelo y partición. ▪ Se obtiene la media y desviación típica de las diferentes métricas en cada *dataset*.
Flujos Alternativos	El proceso finaliza debido a un error.

Tabla 8.5: Especificación del caso de uso CU-1-3. Calcular y guardar métricas.

8.2.4. Caso de Uso 1-4: Guardar resultados.

Las métricas calculadas y los modelos óptimos se almacenan en ficheros, siguiendo la estructura que aparece en el Anexo A.

Este caso de uso se detalla en la Tabla 8.6.

Caso de Uso	CU-1-4. Guardar resultados.
Descripción	Se almacenan las métricas calculadas y los modelos óptimos. Las medias y desviaciones típicas calculadas se guardan en dos ficheros resumen, uno para las particiones de *train* y otro para las de *test*.
Actores	Usuario
Condiciones Iniciales	Se han calculado las métricas de los modelos óptimos de cada algoritmo por partición y las medias y desviaciones típicas de cada conjunto de datos.
Flujo Principal	1. Se almacenan las métricas calculadas y los modelos óptimos en ficheros. En el Anexo A se describirá con más detalle la estructura de ficheros generada. 2. Se guardan las medias y desviaciones típicas en dos archivos resumen, uno para las particiones de *train* y otro para las de *test*.
Condiciones Finales	▪ Las métricas de todos los modelos quedan almacenadas. ▪ Los modelos óptimos para cada partición son almacenados. ▪ Se generan los archivos resumen para las particiones de *train* y *test* con las medias y desviaciones típicas de las métricas calculadas.
Flujos Alternativos	El proceso finaliza debido a un error.

Tabla 8.6: Especificación del caso de uso CU-1-4. Guardar resultados.

Capítulo 9

Tecnologías

En este capítulo 9, se hará una descripción de las principales tecnologías que se han usado para la migración de los algoritmos **NNPOM** y **NNOP** en el *framework* ORCA-Python [8].

9.1. Lenguaje de programación Python

Python [7], es un lenguaje de programación interpretado dinámico y multiplataforma, de propósito general, cuya filosofía hace hincapié en la legibilidad de su código, se podría decir que es bastante parecida a la de Unix. Este es un lenguaje de programación multiparadigma, es decir, que está basado en varios paradigmas de programación, como son: la orientación a objetos [37], la cual soporta parcialmente, la programación imperativa [38] y la programación funcional [39], aunque está última en menor medida que las anteriores.

La licencia de este lenguaje de programación, *Python Software Foundation License* [40], es de código abierto, uno de los muchos motivos de su creciente popularidad tanto en la comunidad científica, como a nivel empresarial y de usuario/desarrollador medio. Otros de los motivos de su auge es que es fácilmente legible, ya que forma parte de su filosofía, su facilidad

de aprendizaje por parte de nuevos usuarios y la capacidad que posee de ampliación, mediante sus numerosas bibliotecas.

Python posee una gran biblioteca estándar o por defecto, la cual es usada para diferentes y variadas tareas. Esto se debe a que se aplica la filosofía "pilas incluidas"(en inglés, *batteries included*), como referencia a los módulos de Python, ya que estos pueden ser mejorados por el usuario con módulos personalizados que pueden estar codificados tanto en Python como en C.

Esto proporciona a Python la habilidad de combinar su ya mencionada clara sintaxis con las ventajas que proporcionan otros lenguajes de un nivel de abstracción más bajo, como pueden ser C y C++. Gracias a esto, se puede interactuar con otras bibliotecas.

La gestión de la memoria en Python es transparente para el progamador, ya que todas las variables del lenguaje son punteros o referencias a objetos. Además, los objetos en Python gestionan internamente una variable contador, la cual se encarga de obtener el número de veces que este objeto está siendo referenciado. De esta manera, cuando el valor de este contador equivale a cero, significa que este objeto no esta siendo usado, por lo que de forma automática el colector de basura libera la memoria del objeto que no está siendo utilizado.

Centrándonos en la capacidad de ampliación debida a sus diversas bibliotecas, nos encontramos con muchas de estas dedicadas a facilitar la realización de tareas en el ámbito científico de las Ciencias de la Computación, lo cual es algo a destacar debido a la naturaleza de este proyecto.

9.1.1. Características del lenguaje

En este apartado enumeraremos y explicaremos las características principales de Python. Algunas de ellas ya han sido mencionadas tanto en el Capítulo 4 como en la Sección 9.1.

Python es un lenguaje interpretado, esto quiere decir, que no es necesario compilar el código fuente, sino que un programa intérprete hace la traducción de código fuente a lenguaje de alto nivel durante la ejecución. Como ventaja tiene que el código es de esta forma más portable entre sistemas pero como desventaja esto también hace que el tiempo de ejecución sea mayor, por razones obvias, ya que en vez de compilarlo y ejecutarlo, la traducción a código fuente se hace de forma simultánea a la ejecución por el interprete.

Además, como se dijo en la Sección anterior, es multiparadigma, ya que no solo es un lenguaje orientado a objetos (POO) [37], también permite otros paradigmas como la programación imperativa [38] y la programación funcional [39] aunque esta última en menor medida.

Otras dos características destacables de Python sobre el tipado, que es fuerte y dinámico.

Cuando se dice que un lenguaje de programación es de tipado fuerte, se quiere decir que cuando una variable es de un tipo concreto, no está permitido usar dicha variable como si fuera de otro tipo. Esta es una característica que a pesar de que a priori puede hacer que sea más complicado de usar por un nuevo usuario, es bastante positiva, ya que a la larga mitiga numerosos problemas y errores que derivan de que este también sea un lenguaje con tipado dinámico, que veremos a continuación en que consiste.

Por otro lado, un lenguaje de tipado dinámico, es aquel que permite a una variable creada desde cero tomar distintos tipos, ya que las comprobaciones del código se hacen durante el tiempo de ejecución, al ser un lenguaje interpretado. Esto hace que el lenguaje sea mucho más versátil que un lenguaje de tipado fijo pero a su vez, conlleva con ello una mayor dificultad de comprensión del código.

Por último, es imprescindible destacar que Python es bastante sencillo de dominar debido a que es bastante intuitivo si tienes un nivel básico de inglés y has trabajado anteriormente con lenguajes orientados a objetos, sin olvidar que para usuarios que parten de cero en el mundo de la programación es un

lenguaje con una curva de aprendizaje poco acentuada.

9.1.2. Bibliotecas utilizadas

En esta Sección se nombrarán y hablarán de las principales bibliotecas de Python que han sido utilizadas para el desarrollo del software.

`Matplotlib` [24], es una biblioteca del lenguaje de programación Python, cuya finalidad es la generación de gráficos a través de datos almacenados. Esta biblioteca posee integración con la biblioteca `Numpy`.

`Numpy` [21] es un proyecto de código abierto que tiene como objetivo permitir la computación numérica con Python.

`Pandas` [23] es una biblioteca de software desarrollada como extensión de la biblioteca `Numpy` para la manipulación y el análisis de datos, dentro del lenguaje de programación Python. Ofrece estructuras de datos y operaciones para manipular tablas numéricas y series temporales.

`Scipy` [29] es una biblioteca libre y de código abierto para Python. Se compone de herramientas y algoritmos matemáticos.

`Scikit-learn` [22] es una biblioteca para aprendizaje automático para el lenguaje de programación Python, programada sobre las bibliotecas `Numpy`, `Matplotlib` y `Scipy`.

Incluye varios algoritmos de clasificación, regresión y análisis de grupos (*clustering*) y prepocesamiento. Está diseñada para operar junto con las bibliotecas `Numpy` y `SciPy`.

Capítulo 10

Arquitectura del Sistema

En este Capítulo, implementaremos una estructura modular del sistema. En esta se representará como interactúan los diferentes módulos entre sí.

10.1. Módulos

Aquí se nombrarán y se explicarán con detalle los dos módulos que componen el Sistema. Cada uno de ellos se corresponde con uno de los clasificadores a desarrollar durante el proyecto. Estos son:

Red neuronal del modelo de probabilidades proporcionales, en inglés, *Proportional Odds Model Neural Network* (NNPOM) [9] y Red neuronal con particiones ordenadas, en inglés, *Neural Network with Ordered Partitions* (NNOP) [10].

10.1.1. Clasificador NNPOM

Este módulo está compuesto por el código del algoritmo adaptado desde el proyecto ORCA [3] al proyecto Orca-Python [8] para el clasificador NNPOM.

El algoritmo NNPOM usa el enfoque de los modelos de umbral (*threshold models*), que son uno de los enfoques más comunes para la regresión ordinal, basado en proyectar patrones en la línea real y dividir esta línea real en intervalos consecutivos, asignando un intervalo para cada clase, por lo que se requieren n-1 umbrales, siendo n el número de clases del problema.

10.1.2. Clasificador NNOP

De forma análoga a la Subsección 10.1.1, este módulo comprenderá el código perteneciente al clasificador NNOP dentro del proyecto Orca-Python.

El algoritmo NNOP usa el enfoque *OrderedPartitions*, en el que la categoría asignada a un patrón es igual al índice anterior al del primer nodo de salida cuyo valor es mayor que un umbral T predefinido, o cuando no quedan nodos.

Capítulo 11

Diagrama de Clases

El objetivo de este Capítulo, es esquematizar y clarificar el diseño del software a realizar en el proyecto a través de la comprensión de las clases que componen el mismo, mediante el uso de diagramas de clase, para cada uno de los dos módulos especificados en el Capítulo anterior.

Además de la inclusión de los diagramas de clase correspondientes, también se proporcionará una tabla con la especificación de cada clase. En esta, se describirán los métodos y variables que componen a la clase, de la forma que viene indicada en la notación de la Tabla 11.1.

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

Tabla 11.1: notación usada para describir las clases del sistema

11.1. Clase NNPOM

En esta Sección, se gestionarán los parámetros del algoritmo NNPOM [9] y se ejecutarán los módulos que se encargarán tanto de ejecutar el algoritmo en sí como de los métodos *fit* y *predict* que conforman *train* y *test* del proceso de aprendizaje de la red neuronal.

Dentro de estos métodos también tendrán lugar las comprobaciones necesarias para aseverar que los procesos de aprendizaje de la red así como la formación del modelo de esta tienen el formato correcto.

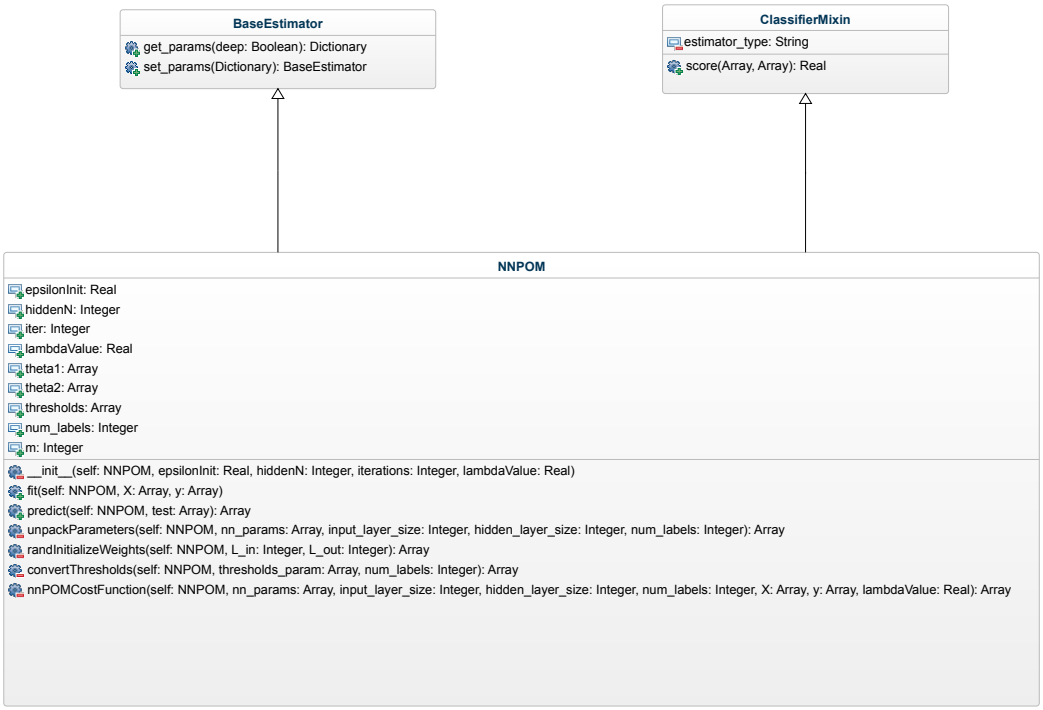

Figura 11.1: diagrama de clases de la clase NNPOM

Clase: NNPOM		
Clase que contiene al clasificador NNPOM dentro de el *framework* Orca-Python.		
Datos		
epsilonInit	*Float*	Contiene el valor de epsilon, es decir, el rango de inicialización de los pesos.
hiddenN	*Integer*	Número de neuronas de la capa oculta del modelo.
iter	*Integer*	Número de iteraciones máximas para el algoritmo fmin_l_bfgs_b.
lambdaValue	*Float*	Parámetro de la regularización.
theta1	*Array*	Array con los pesos de la capa oculta (incluidos sesgos).
theta2	*Array*	Array con los pesos de la capa de salida (sin incluir sesgos, ya que estos serán los umbrales).
thresholds	*Array*	Umbrales de las clases del modelo.
num_labels	*Integer*	Número de etiquetas del problema.
m	*Integer*	Número de patrones del conjunto de patrones de entrenamiento del problema.
Métodos		
__init__	*Void*	Inicializa los diferentes parámetros del clasificador acorde a la información introducida por el usuario y los valores por defecto

fit	*NNPOM*	Se encarga de realizar el entrenamiento del modelo utilizando el conjunto de datos de entrenamiento pasado como argumento.
predict	*Array*	Determina las clases de un conjunto de datos.
__unpackParameters	*Array*	Obtiene las variables con los pesos entre capas, sesgos y umbrales incluidos del array columna *nn_params* que compacta todos estos datos.
__randInitializeWeights	*Array*	Inicializa aleatoriamente los pesos de una capa con *L_in* conexiones entrantes y *L_out* conexiones salientes.
__convertThresholds	*Array*	Transforma los umbrales para realizar una optimización sin restricciones.
__NNPOMCostFunction	*Array*	Implementa la función de coste y obtiene las derivadas correspondientes.

Tabla 11.2: especificación de la clase NNPOM

11.2. Clase NNOP

En esta Sección 11.2, se gestionarán los parámetros del algoritmo NNOP [10] y se ejecutará genererarán los módulos que se encargarán tanto de ejecutar el algoritmo en sí como de los métodos *fit* y *predict* que conforman *train* y *test* del proceso de aprendizaje de la red neuronal.

Dentro de estos métodos también tendrán lugar las comprobaciones ne-

cesarias para aseverar que los procesos de aprendizaje de la red así como la formación del modelo de esta tienen el formato correcto.

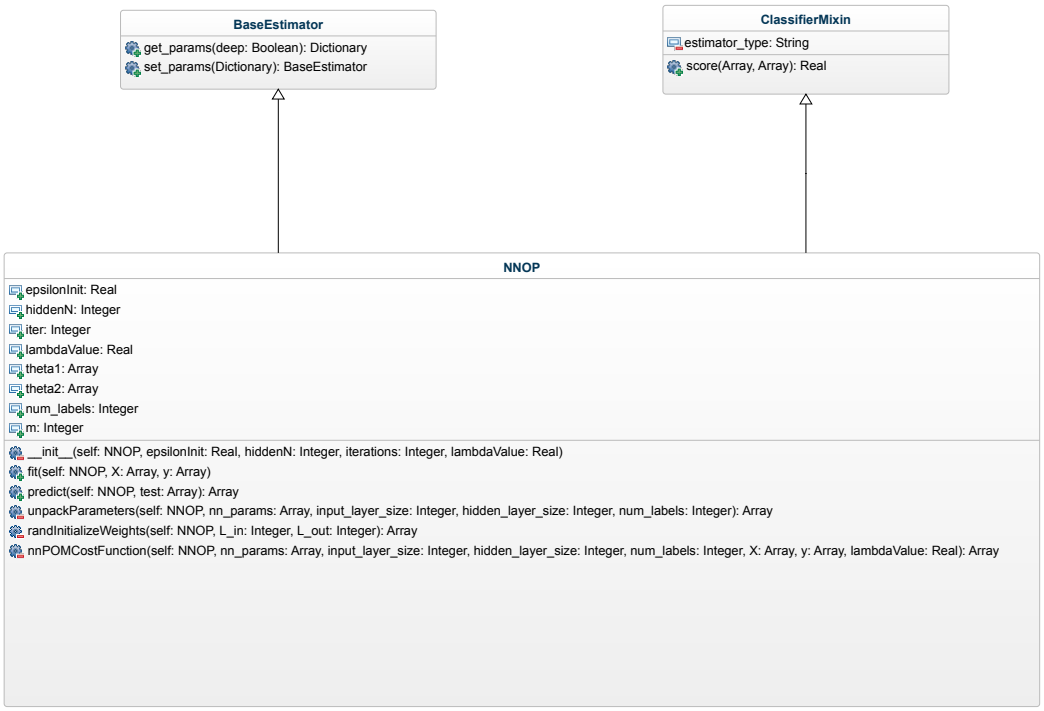

Figura 11.2: diagrama de clases de la clase NNOP

Clase: NNOP		
Clase que contiene al clasificador NNOP dentro de el *framework* Orca-Python.		
Datos		
epsilonInit	*Float*	Contiene el valor de epsilon, es decir, el rango de inicialización de los pesos.
hiddenN	*Integer*	Número de neuronas de la capa oculta del modelo.
iter	*Integer*	Número de iteraciones máximas para el algoritmo fmin_l_bfgs_b.
lambdaValue	*Float*	Parámetro de la regularización.
theta1	*Array*	Array con los pesos de la capa oculta (incluidos sesgos).
theta2	*Array*	Array con los pesos de la capa de salida.
num_labels	*Integer*	Número de etiquetas del problema.
m	*Integer*	Número de patrones del conjunto de patrones de entrenamiento del problema.
Métodos		
__init__	*Void*	Inicializa los diferentes parámetros del clasificador acorde a la información introducida por el usuario y los valores por defecto
fit	*NNOP*	Se encarga de realizar el entrenamiento del modelo utilizando el conjunto de datos de entrenamiento pasado como argumento.
predict	*Array*	Determina las clases de un conjunto de datos.

__unpackParameters	*Array*	Obtiene las variables con los pesos entre capas y los sesgos incluidos del array columna *nn_params* que compacta todos estos datos.
__randInitializeWeights	*Array*	Inicializa aleatoriamente los pesos de una capa con *L_in* conexiones entrantes y *L_out* conexiones salientes.
__NNOPCostFunction	*Array*	Implementa la función de coste y obtiene las derivadas correspondientes.

Tabla 11.3: especificación de la clase NNOP

Parte IV

Pruebas

Capítulo 12

Pruebas

La fase de pruebas vendrá especificada en este Capítulo. Esta fase es imprescindible en cualquier proyecto *software* durante la fase de desarrollo ya que de esta forma el diseñador puede obtener la certeza de que el funcionamiento del sistema es óptimo y, en caso de haber alguna inconsistencia en este, el propio sistema tenga las herramientas necesarias para subsanarlo o notificarlo al usuario. Estas pruebas pueden llegar a detectar errores en todas las fases del desarrollo del software, desde la fase más temprana hasta en la pruebas finales antes de hacer la entrega al cliente.

Como el cometido del proyecto es desarrollar dos algoritmos para el *framework* Orca-Python, se deberá asegurar el correcto funcionamiento de estos y la correcta integración con el *framework*, ya que al formar parte de este, interactuarán con otros métodos del *framework*.

A continuación se detallarán las pruebas realizadas, realizando una introducción teórica previa sobre estas.

12.1. Pruebas unitarias

En esta Sección se explicará el marco teórico sobre las pruebas unitarias y se detallarán cuáles han sido efectuadas durante el desarrollo del proyecto.

Las pruebas unitarias comprueban si es correcto el funcionamiento de la unidad más pequeña del diseño de *software*, el módulo. En un sistema se podría considerar como módulo un fragmento de código, ya este encapsulado o no, en un método o función, que pueda ser ejecutado de manera independiente al resto de código. Este tipo de pruebas se pueden dividir en dos tipos diferenciados (y que explicaremos en las siguientes Subsecciones de esta Sección): 12.1.1 pruebas de caja blanca y 12.1.2 pruebas de caja negra.

12.1.1. Pruebas de caja blanca

Las pruebas de caja blanca se realizan durante la fase de diseño. Su objetivo es asegurar que el funcionamiento óptimo de distintos módulos, cada uno probándolo por separado. Este tipo de pruebas son denominadas también como pruebas estructurales ya que solo sirven para una determinada estructura del código desarrollado, normalmente si el código cambia, estas pruebas tendrán que ser adaptadas a estos cambios.

Estas pruebas son realizadas durante la implementación del código para tener en cuenta futuros errores que pueden darse y como comprobación de que el funcionamiento de lo implementado hasta ahora es el esperado.

Durante la implementación del código se han realizado algunas de estas pruebas comprobando los valores de las distintas variables que cada uno de los métodos de cada algoritmo y comprobando estos valores con los análogos resultantes de la ejecución del proyecto en el que se basa, ORCA.

12.1.2. Pruebas de caja negra

Las pruebas de caja negra, al igual que las pruebas de caja blanca, se realizan de forma paralela al desarrollo del *software*. Su cometido es comprobar que información contenida en las salidas de cada uno de los módulos es la esperada. Es por ello que este tipo de pruebas también es denominado como pruebas funcionales, ya que estas se centran exclusivamente en que los módulos probados cumplen las funcionalidades para las que fueron diseñados.

Durante la implementación del código se han realizado algunas de estas pruebas comprobando los valores de las salidas de cada uno de los módulos de cada clase y comprobando estos valores con los análogos resultantes de la ejecución del proyecto en el que se basa, ORCA.

12.2. Pruebas de integración

Una vez implementadas las pruebas unitarias para los módulos del sistema en desarrollo, tendrán lugar las pruebas de integración.

Las pruebas de integración tienen como objetivo comprobar que la comunicación entre módulo es óptima y que en el flujo de datos no tienen lugar cambios que den lugar a errores. Los métodos codificados para realizar las pruebas de integración vienen explicados en las Tablas 12.1 y 12.2. Estas pruebas verifican que los clasificadores NNPOM [9] y NNOP [10] hacen el entrenamiento y la predicción de forma óptima comparando los resultados obtenidos con los de ejecuciones anteriores que han sido almacenados en ficheros para este cometido. Otra comprobación que realizan es que los algoritmos sean consistentes ante errores, es decir, detecten situaciones que puedan llegar a producir errores y los indica antes de que sucedan, para que estos sean controlados.

Estas pruebas están basadas en pruebas de integración anteriores realizadas para los algoritmos REDSVM y SVOREX.

Caso	Descripción	Resultado Esperado
2.1	Se carga un conjunto de datos y se aplica sobre el mismo el proceso de entrenamiento del clasificador NNPOM varias veces usando un parámetro no válido en cada una de ellas. **Función:** *test_nnpom_fit_not_valid_parameter*	El clasificador lanza una excepción indicando el parámetro que no es válido y devuelve un modelo NNPOM nulo.
2.2	Se carga un conjunto de datos que es modificado para ser inválido para el clasificador. Se intenta ejecutar el entrenamiento del modelo NNPOM utilizando este conjunto de datos erróneo o uno vacío. **Función:** *test_nnpom_fit_not_valid_data*	El clasificador lanza una excepción indicando que los datos de entrada son incorrectos y devuelve un modelo NNOP nulo.
2.3	Se intenta realizar el proceso de predicción del clasificador NNPOM usando un conjunto de patrones no válido o vacío. **Función:** *test_nnpom_predict_not_valid_data*	Se lanza una excepción indicando que los datos son erróneos.

Tabla 12.1: Casos de pruebas de integración para el clasificador NNPOM

Caso	Descripción	Resultado Esperado
3.1	Se carga un conjunto de datos y se aplica sobre el mismo el proceso de entrenamiento del clasificador NNOP varias veces usando un parámetro inválido en cada una de ellas. **Función:** *test_nnop_fit_not_valid_parameter*	El clasificador lanza una excepción indicando el parámetro que es inválido y devuelve un modelo NNPOM nulo.
3.2	Se carga un conjunto de datos que es modificado para ser inválido para el clasificador. Se intenta ejecutar el entrenamiento del modelo NNOP utilizando este conjunto de datos erróneo o uno vacío. **Función:** *test_nnop_fit_not_valid_data*	El clasificador lanza una excepción indicando que los datos de entrada son incorrectos y devuelve un modelo NNOP nulo.
3.3	Se intenta realizar el proceso de predicción del clasificador NNOP usando un conjunto de patrones no válido o vacío. **Función:** *test_nnop_predict_not_valid_data*	Se lanza una excepción indicando que los datos son erróneos.

Tabla 12.2: Casos de pruebas de integración para el clasificador NNOP

12.3. Pruebas del sistema

Una vez hayan sido obtenidos unos resultados satisfactorios en las pruebas de las Secciones anteriores, se puede asegurar que el sistema desarrollado está completo siempre y cuando estas pruebas estén diseñadas de forma adecuada. Es entonces cuando se realizan las pruebas del sistema.

Las pruebas del sistema tiene como objetivo comprobar que el sistema cumple los requisitos exigidos para este, es decir, que cumple con las funcionalidades que se le pide al *software* diseñado. Este tipo de pruebas pueden dividirse en función del objetivo que tengan que comprobar, habiendo pruebas de funcionalidad, de rendimiento, de estabilidad, etc.

Se comprobará mediante pruebas de funcionalidad si los algoritmos de generación de modelos de redes neuronales para clasificación ordinal que han sido adaptados tienen un funcionamiento óptimo.

12.4. Pruebas de aceptación

Las pruebas de aceptación son las que realizan los usuarios finales de producto *software* desarrollado. En este caso, al ser un Trabajo de Fin de Grado (TFG), los usuarios finales serán los directores del proyecto, que a través de experimentos lanzados por ellos mismos, comprobarán el funcionamiento del sistema e indicarán las futuras posibles mejoras de este.

Capítulo 13

Resultados experimentales

Una vez desarrollado el sistema y este haya pasado las pruebas unitarias y de sistema primarias, el sistema está preparado para los experimentos o pruebas experimentales que confirmen definitivamente que el funcionamiento del sistema es óptimo, comparando los resultados obtenidos en nuestro *framework* Orca-Python, con los resultados obtenidos para unos experimentos idénticos con el *framework* antecesor, ORCA.

Antes de mostrar los resultados obtenidos, se explicará el diseño de los experimentos a realizar y una vez detallados se indicarán los resultados obtenidos con ambos sistemas y se compararán los resultados.

13.1. Diseño

En esta Sección, se detallará toda la información correspondiente a los experimentos a realizar, sin dejar lugar a dudas sobre estos, para que queden claras todas las variables que influyen en este y se tenga constancia de la robustez y validez del experimento. Esta Sección será dividida en dos Subsecciones para facilitar la claridad en cuanto a la exposición de la información que contiene esta Sección.

La Subsección 13.1.1, en la que se detallarán los *datasets* que serán utilizados durante los experimentos y las características de estos. Por otro lado, la Subsección 13.1.2, detallará los parámetros de los algoritmos y los valores que estos tomarán en los experimentos, así como los procesos de validación que serán tomados.

13.1.1. Conjuntos de datos

En esta Subsección, tal y como explicamos en los párrafos anteriores, se determinarán los *datasets* que serán utilizados en los experimentos y las características de estos.

Las bases de datos utilizadas son de clasificación ordinal, es decir, consisten en problemas de clasificación ordinal reales. Estas han sido divididas en 30 particiones aplicando *30-holdouts* sobre la base de datos. Cada partición de dichas base de datos estará compuesta de una parte de entrenamiento y otra de generalización (*train* y *test*).

Los seis *datasets* utilizados en estos experimentos han sido obtenidos de la página web del departamento AYRNA [4]. En la Tabla 13.1, se especifican las características de estos datasets.

Bases de datos de clasificación ordinal				
Nombre BBDD	#Inst.	#Atr.	#Clas.	Distribución Clas.
balance-scale (**BS**)	625	4	3	(288, 49, 288)
contact-lenses (**CL**)	24	6	3	(15, 5, 4)
tae (**TA**)	151	54	3	(49, 50, 52)
car (**CA**)	1728	21	4	(1210, 384, 69, 65)
winequality-red (**WR**)	1599	11	6	(10, 53, 681, 638, 199, 18)
ERA (**ER**)	1000	4	9	(92, 142, 181, 172, 158, 118, 88, 31, 18)

Tabla 13.1: Características de los *datasets* utilizados en los experimentos

13.1.2. Parámetros de los experimentos

Una vez determinados los *datasets* en el apartado anterior, se procederá a especificar las variables restantes que son necesarias para el lanzamiento de un experimento. Para ello, se requieren dos tipos de parámetros, los parámetros generales y los específicos del entrenamiento de los modelos de redes neuronales. El rango de valores de estos parámetros será el mismo que el usado en el artículo [1] con el *framework* ORCA [3], para hacer una comparación de los resultados obtenidos con nuestro *framework*.

En cuanto a los parámetros iniciales, se marcará como activa la opción de estandarizar los datos sacados de los *datasets* que se usarán en los experimentos y también se usará un proceso anidado de validación cruzada de tipo *k-fold* de 5-*folds*. Las métricas empleadas serán *MAE* (*Mean Absolute Error*) y *MZE* (*Mean Zero-One Error*), las cuales se explicarán a continuación.

- ***MAE*:** calcula la media de las diferencias relativas entre las posiciones de la clase predicha y la real:

$$MAE = \frac{1}{n}\sum_{q,k=1}^{Q}|\mathcal{O}(y_i) - \mathcal{O}(y_i^*)|n_{qk} \quad = \quad \frac{1}{n}\sum_{i=1}^{n} e(\mathbf{x_i}) \tag{13.1}$$

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

- ***MZE*:** se trata de la medida complementaria del *Correct Classification Rate* (CCR) y es también conocida como ratio de error:

$$MZE = 1 - CCR \tag{13.2}$$

Será necesario realizar un experimento por métrica a resaltar.

En cuanto a los clasificadores, a continuación se mostrarán el rango de valores escogido para cada parámetro de estos. Dado que ambos tienen los mismos parámetros de entrada, y, tanto para NNPOM como para NNOP

se han escogido los mismos valores, esta información se proporcionará en el documento una sola vez, siendo el lector consciente de que estos valores son válidos para las ejecuciones realizadas con ambos modelos de redes neuronales:

- **epsilonInit**: [0, 5]
- **hiddenN**: [5, 10, 20, 30, 40, 50]
- **iterations**: [250, 500]
- **lambdaValue**: [0, 0.01, 1]

En caso de requerir más información sobre los parámetros enumerados anteriormente, se hace referencia al Capítulo 11, donde se explican tanto los parámetros de cada algoritmo como los métodos que este emplea y su funcionamiento.

13.2. Resultados

En esta Sección se mostrarán mediante Tablas los resultados de los experimentos y su comparación con los resultados obtenidos para el *framework* ORCA, para cada una de las métricas seleccionadas y con los mismos parámetros y *datasets* para cada sistema. La Tabla 13.2, muestra los resultados obtenidos considerando la métrica MAE durante el procedimiento de optimización mientras que la Tabla 13.3, muestra los resultados considerando la métrica MZE.

Se pueda observar en las tablas que salvo en el *dataset ER* el *framework* Orca-Python suele ser superior a ORCA. Dado que estamos tratando con *datasets* un tamaño relativamente pequeño y se lanzan muchas ejecuciones, con distintas configuraciones, permutando entre los valores indicados en el archivo de configuración, que la generación aleatoria de los pesos no influye

Bases de datos de clasificación ordinal ($\text{Media}_{\text{Desviacion Tipica}}$)				
	$\text{NNPOM}_{\text{ORCA}}$	$\text{NNPOM}_{\text{ORCA-Python}}$	$\text{NNOP}_{\text{ORCA}}$	$\text{NNOP}_{\text{ORCA-Python}}$
BS	$0.11_{0.19}$	$\mathbf{0.02_{0.01}}$	$0.04_{0.01}$	$\mathit{0.03}_{\mathit{0.01}}$
CL	$0.48_{0.23}$	$\mathit{0.46}_{\mathit{0.26}}$	$\mathit{0.46}_{\mathit{0.25}}$	$\mathbf{0.43_{0.28}}$
TA	$0.58_{0.13}$	$\mathit{0.54}_{\mathit{0.08}}$	$\mathit{0.54}_{\mathit{0.11}}$	$\mathbf{0.51_{0.09}}$
CA	$0.12_{0.03}$	$\mathbf{0.00_{0.00}}$	$0.03_{0.01}$	$\mathit{0.00}_{\mathit{0.00}}$
WR	$0.45_{0.03}$	$\mathbf{0.41_{0.01}}$	$0.44_{0.02}$	$\mathit{0.41}_{\mathit{0.02}}$
ER	$1.26_{0.06}$	$\mathit{1.24}_{\mathit{0.05}}$	$\mathbf{1.18_{0.02}}$	$\mathit{1.24}_{\mathit{0.05}}$

Tabla 13.2: comparación de los métodos en términos de *MAE*. En la Tabla se muestran los valores de la forma $Media_{DesviacionTipica}$.

Bases de datos de clasificación ordinal ($\text{Media}_{\text{Desviacion Tipica}}$)				
	$\text{NNPOM}_{\text{ORCA}}$	$\text{NNPOM}_{\text{ORCA-Python}}$	$\text{NNOP}_{\text{ORCA}}$	$\text{NNOP}_{\text{ORCA-Python}}$
BS	$0.062_{0.048}$	$\mathbf{0.018_{0.012}}$	$0.039_{0.013}$	$\mathit{0.023}_{\mathit{0.021}}$
CL	$0.356_{0.143}$	$\mathit{0.327}_{\mathit{0.148}}$	$\mathbf{0.283_{0.125}}$	$0.333_{0.131}$
TA	$0.453_{0.090}$	$0.427_{0.071}$	$\mathit{0.415}_{\mathit{0.060}}$	$\mathbf{0.406_{0.070}}$
CA	$0.106_{0.022}$	$\mathbf{0.006_{0.003}}$	$\mathit{0.026}_{\mathit{0.013}}$	$\mathbf{0.006_{0.003}}$
WR	$0.401_{0.022}$	$\mathit{0.385}_{\mathit{0.019}}$	$0.402_{0.018}$	$\mathbf{0.382_{0.019}}$
ER	$\mathit{0.727}_{\mathit{0.028}}$	$\mathit{0.727}_{\mathit{0.024}}$	$\mathbf{0.708_{0.017}}$	$0.732_{0.025}$

Tabla 13.3: comparación de los métodos en términos de *MZE*. En la Tabla se muestran los valores de la forma $Media_{DesviacionTipica}$.

en los resultados. La razón que puede haber detrás de este comportamiento es que se utilizan algoritmos de optimización distintos, ya que en ORCA se usa `iRprop+` y en Orca-Python `fmin_l_bfgs_b`.

Esta puede ser la razón principal de porqué Orca-Python trabaja mejor con los *datasets* más pequeños mientras que ORCA es ligeramente superior en el más grande de estos, ER.

Por lo demás, los resultados en ambos *framework* son muy similares siendo incluso de forma general los resultados obtenidos por el *framework* Orca-Python [8] mejores a los que se obtuvieron con ORCA [3].

Parte V

Conclusiones

Capítulo 14

Conclusiones del Autor

Una vez superadas todas las pruebas realizadas que han sido documentadas en los Capítulos 12 y 13, se da por terminado el desarrollo y adaptación de los algoritmos NNPOM y NNOP para el *framework* Orca-Python [8], por lo que procedemos a detallar el grado de cumplimiento de los diferentes objetivos propuestos en el Capítulo 7.

14.1. Objetivos de formación alcanzados

En esta Sección, serán enumerados los conocimientos adquiridos por el alumno en la realización del TFG:

- Manejo y comprensión en profundidad de los *frameworks* ORCA y Orca-Python.
- Aprendizaje de los lenguajes de programación MATLAB/Octave y Python, haciendo hincapié en este último.
- Dominio en el uso de librerías de Pyhton usadas en el proyecto, haciendo especial mención a las librerías `numpy` y `scikit-learn`.

- Aprendizaje y profundización en conceptos del campo de las redes neuronales y la regresión ordinal.
- Dominio de la herramienta LaTeX durante la documentación del proyecto.

14.2. Objetivos operacionales alcanzados

Los objetivos operacionales superados por el alumno durante la realización del TFG:

- Implementación con éxito de los clasificadores NNPOM y NNOP en Python e inclusión de estos al *framework* Orca-Python.
- La facilidad de uso de Orca-Python ha sido asegurada tras inclusión de los nuevos algoritmos.
- Implementados distintos *tests* de integración para la comprobación del buen funcionamiento de los algoritmos por parte del desarrollador como del usuario final.
- Superado el estudio que avala que los algoritmos implementados para el *framework* obtiene un rendimiento similar o incluso superior al obtenido por medio del *framework*, ORCA.

Capítulo 15

Futuras Mejoras

Dado que el número de objetivos del proyecto actual es limitado debido a restricciones temporales impuestas por la naturaleza del proyecto, el TFG no ha podido desarrollar algunas de la mejoras que pueden ser implementadas en el sistema y estas deberán de ser desarrolladas mediante otros Trabajos de Fin Grado como mejoras de lo obtenido hasta el momento. En este último Capítulo del documento se presentarán algunas de las mejoras que se podrían llevar a cabo en futuros Trabajos de Fin de Grado.

- Finalizar el desarrollo del*framework* Orca-Python, es decir, migrar el resto de algoritmos del *framework* ORCA. Algunos de estos algoritmos pueden ser ELMOP, POM o SVMOP.
- También es posible incluir la programación en Cython [41]. De esta forma se mejoraría de forma notable los tiempos de ejecución de distintos algoritmos durante los experimentos.

Bibliografía

[1] Pedro Antonio Gutiérrez, María Pérez Ortiz, Javier Sánchez Monedero, Francisco Fernández Navarro, and Cesar Hervás Martínez. Ordinal regression methods: Survey and experimental study. *IEEE Transactions on Knowledge and Data Engineering*, 28, 07 2015.

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

[3] Javier Sánchez Monedero, Pedro Antonio Gutiérrez, and María Pérez Ortiz. Orca: A matlab/octave toolbox for ordinal regression. *Journal of Machine Learning Research, 20*, pages 1–4, 2019.

[4] Departamento de informática y análisis numérico de la universidad de córdoba. aprendizaje y redes neuronales artificiales. `https://www.uco.es/grupos/ayrna/index.php/es`. [Online. Última consulta: 26-08-2021].

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

[6] John W. Eaton. *Gnu Octave 4.2 Reference Manual*. Samurai Media Limited, London, GBR, 2017.

[7] G. van Rossum. *Python tutorial. Technical Report CS-R9526, Centrum voor Wiskunde en Informatica (CWI)*. 5 1995.

[8] Iván Bonaque Muñoz, Pedro Antonio Gutiérrez Peña, and Javier Sánchez Monedero. Framework en python para problemas de clasificación ordinal, 2019.

[9] M. J. Mathieson. Ordinal models for neural networks. *3rd Int. Conf. Neural Netw. Capital Markets*, pages 523–536, 1996.

[10] Z. Wang J. Cheng and G. Pollastri. A neural network approach to ordinal regression. *IEEE Int. Joint Conf. Neural Netw. (IEEE World Congr. Comput. Intell.)*, pages 1279–1284, 2008.

[11] Product design especification. definition. `https://www.sciencedirect.com/topics/engineering/product-design-specification`. [Online. Última consulta: 06-09-2021].

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

[13] Sitio oficial de ubuntu. `https://ubuntu.com/`. [Online. Última consulta: 01-08-2021].

[14] Sitio oficial de anaconda. `https://www.anaconda.com/`. [Online. Última consulta: 01-08-2021].

[15] Gnu bash. `https://www.gnu.org/software/bash/`. [Online. Última consulta: 01-06-2021].

[16] mord: Ordinal regresssion in python. `https://pythonhosted.org/mord/`. [Online. Última consulta: 06-09-2021].

[17] Repositorio del presente proyecto. `https://github.com/adlopor/orca-python`. [Online. Última consulta: 05-09-2021].

[18] Github: the world's leading software development platform. `https://github.com/`. [Online. Última consulta: 04-09-2021].

[19] Git: –everything-is-local. `https://git-scm.com/`. [Online. Última consulta: 09-06-2021].

[20] PEP 8 – Style Guide for Python Code. `https://www.python.org/dev/peps/pep-0008`. [Online. Última consulta: 09-06-2021].

[21] Sitio oficial de numpy. `https://numpy.org/`. [Online. Última consulta: 15-03-2021].

[22] Sitio oficial de scikit-learn. `https://scikit-learn.org/stable/`. [Online. Última consulta: 15-03-2021].

[23] Sitio oficial de pandas. `https://pandas.pydata.org/`. [Online. Última consulta: 15-03-2021].

[24] Sitio oficial de matplotlib. `https://matplotlib.org/`. [Online. Última consulta: 15-03-2021].

[25] Máquina de vectores de soporte (svm). `https://la.mathworks.com/discovery/support-vector-machine.html`. [Online. Última consulta: 13-03-2021].

[26] Sitio oficial de ieee. `https://www.ieee.org/`. [Online. Última consulta: 01-08-2021].

[27] What is gnu? `https://www.gnu.org/home.en.html`. [Online. Última consulta: 20-05-2021].

[28] Texto de licencia bsd original. `http://www.xfree86.org/3.3.6/COPYRIGHT2.html#6`. [Online. Última consulta: 07-07-2021].

[29] Sitio oficial de scipy. `https://www.scipy.org/`. [Online. Última consulta: 15-03-2021].

[30] Sitio oficial de latex. `https://www.latex-project.org/`. [Online. Última consulta: 23-02-2021].

[31] Sitio oficial de overleaf. `https://es.overleaf.com/`. [Online. Última consulta: 23-02-2021].

[32] Sitio oficial de visual studio code. https://code.visualstudio.com/. [Online. Última consulta: 08-08-2021].

[33] Unit testing framework. https://docs.python.org/3/library/unittest.html. [Online. Última consulta: 15-03-2021].

[34] Gnu make. https://www.gnu.org/software/make/manual/make.html. [Online. Última consulta: 07-04-2021].

[35] Sitio oficial de genmymodel. https://www.genmymodel.com/. [Online. Última consulta: 05-08-2021].

[36] Gnu image manipulation program, gimp. http://www.gimp.org.es/. [Online. Última consulta: 06-09-2021].

[37] A simple explanation of oop. https://medium.com/@richardeng/a-simple-explanation-of-oop-46a156581214. [Online. Última consulta: 1-08-2021].

[38] Imperative programming paradigm. https://www.computerhope.com/jargon/i/imp-programming.htm. [Online. Última consulta: 02-05-2020].

[39] Functional programming paradigm. https://www.geeksforgeeks.org/functional-programming-paradigm/. [Online. Última consulta: 02-05-2020].

[40] History and license. https://docs.python.org/3/license.html. [Online. Última consulta: 02-06-2021].

[41] C-extensions for pyhton. https://cython.org/. [Online. Última consulta: 03-09-2021].

[42] Sitio oficial de sacred. https://sacred.readthedocs.io/en/stable/index.html. [Online. Última consulta: 01-08-2021].

[43] Sitio web de pip. https://pypi.org/project/pip/. [Online. Última consulta: 01-08-2021].

Parte VI

Apéndices

Apéndice A

Manual de Usuario

Este Anexo proporciona la información básica de uso del sistema para el usuario final. Además de este documento, podrá encontrar información útil para el usuario final en el fichero *README.txt* presente en el proyecto software [17], que está disponible en la plataforma Github [18].

A continuación, en los siguientes apartados se proporcionará toda la información necesaria para que el usuario pueda usar el *software* implementado en este TFG.

A.1. Descripción del producto

Este producto es una mejora del *framework* Orca-Python, gracias a la inclusión de los algoritmos NNPOM y NNOP, incluidos previamente en ORCA [3].

A lo largo del Manual de Usuario se detallará el uso del *framework* con las mejoras implementadas.

A.2. Requisitos del sistema

Como requisitos de *software* de este *framework* podrá ser utilizado en cualquier Sistema Operativo (SO), que contenga una versión compatible de Python (versiones 2 y 3) [7] instalada junto con una serie de paquetes y dependencias necesarias para su uso y que serán indicadas en la Sección posterior A.4.1.

Una vez aclarados los requisitos *software* necesarios para el uso del *framework*, especificamos las restricciones de hardware, como tener al menos 4GB de RAM como mínimo o un tamaño de ROM necesarios para almacenar el proyecto y los *datasets* a usar. Como también es obvio, cuanto mejores características de hardware posea el usuario (número de núcleos de CPU, potencia de estos, memoria RAM elevada, etc.), el tiempo de computación podrá verse reducido, lo que facilitaría en cierto modo el uso de este por parte del usuario.

A.3. ¿Qué es ORCA-Python?

Orca-Python es un *framework* desarrollado en el lenguaje de programación Python [7], haciendo uso algunas librerías como `Numpy` [21], `scikit-learn` [22], `Sacred` [42] o `Pandas` [23], entre otras. El objetivo de este es automatizar la ejecución de experimentos de aprendizaje automático para problemas de regresión ordinal mediante el uso de sencillos ficheros de configuración *.json* de fácil comprensión.

A.4. Instalación

Orca-Python ha sido desarrollado y testeado con SO GNU/Linux, usando las versiones 2 y 3 de Python.

A.4.1. Requisitos para la instalación

Para una correcta ejecución del *framework* sin errores será necesaria la previa instalación de los siguientes paquetes de Python.

- `numpy` [21] (probado con la versión 1.18.1).
- `pandas` [23] (probado con la versión 1.0.1).
- `sacred` [42] (probado con la versión 0.8.1).
- `scikit-learn` [22] (probado con la versión 0.22.1).
- `scipy` [29] (probado con la versión 1.4.1).

Para la facilidad de instalación por parte de usuario de todos estos requerimientos cabe la posibilidad del uso del archivo *requeriments.txt*, el cual se encuentra dentro del repositorio y facilitará el proceso de instalación de estos paquetes mediante el uso del gestor de paquetes *pip* [43].

A.4.2. Descarga de Orca-Python

Para descargar el repositorio Orca-Python, será necesario clonar el repositorio de GitHub mediante el siguiente:

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

O bien, descargando el repositorio comprimido en formato zip mediante el botón *Download ZIP* presente en la página del repositorio.

A.4.3. Probando la instalación

En el repositorio hay incluidas varios *datasets* de prueba y un experimento, para comprobar que la instalación se ha realizado de forma óptima. Para

probar si la instalación ha sido correcta basta con ejecutar la siguiente línea de comandos desde la raíz del repositorio.

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

Si ha sido instalado siguiendo los pasos indicados en la Sección A.4 la ejecución debería completarse sin problemas. El procedimiento de ejecución no varía con el experimento. Una vez explicado esto, se mostrará la salida obtenida por el terminal tras realizar la ejecución del comando indicado anteriormente, la cual debería de ser similar a la mostrada a continuación:

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

Running tae dataset
--------------------------
Running LR ...
  Running Partition 0
  Running Partition 1
  ...
  Running Partition 28
  Running Partition 29
Running REDSVM ...
  Running Partition 0
  Running Partition 1
  ...

...

Running contact-lenses dataset
--------------------------
```

```
Running LR ...
  Running Partition 0
  Running Partition 1
  ...
  Running Partition 28
  Running Partition 29
Running REDSVM ...
  Running Partition 0
  Running Partition 1
  ...
  Running Partition 28
  Running Partition 29
Running SVM ...
  Running Partition 0
  Running Partition 1
  ...

...

Saving Results...
```

A.5. Desinstalación

Para la desinstalación del *framework* tan solo es necesario eliminar la carpeta en la que fue clonado el repositorio. En cuanto a las dependencias/requerimientos instalados, en caso de haber utilizado un entorno virtual bastará con esto, mientras que si las dependencias han sido instaladas sobre la instalación de Python del equipo previo al borrado de la carpeta del *framework* se ha de ejecutar el siguiente comando para desinstalar estas dependencias:

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

A.6. ¿Cómo utilizar Orca-Python?

Esta Sección se dividirá en tres Subsecciones que explicarán las configuraciones que se han de tener en cuenta antes de lanzar un experimento.

A.6.1. Archivos de Configuración

Para configurar y lanzar los experimentos, se usan archivos de configuración con formato JSON, los cuales cuentan con dos partes diferenciadas. La configuración general llamada como *general-conf* en el archivo de configuración que contiene la información básica acerca del experimento, como el número de folds, los conjuntos de datos a utilizar, las métricas a medir, entre otros. Por otra parte, se encuentran las configuraciones de los algoritmos llamada como *configurations* en el archivo de configuración, que contiene todos los algoritmos que se usarán y la información acerca de cada uno, es decir, los parámetros de estos, pudiendo utilizar o ajustar entre dos o más valores para un mismo parámetro.

Estas dos partes del archivo de configuración tendrán un diccionario de Python que tendrá como clave los nombres comentados en el anterior párrafo. En la Sección A.4.3, está incluido un archivo de configuración como muestra de lo que devuelve la ejecución de un experimento.

A continuación, se enseña la parte *general-conf* de un archivo de configuración:

```
   "general_conf": {

"basedir": "datasets",
"datasets": ["balance-scale", "contact-lenses", "tae",
"car", "winequality-red", "ERA"],
"hyperparam_cv_nfolds": 5,
"jobs": 10,
```

```
"input_preprocessing": "std",
"output_folder": "my_runs/",
"metrics": ["ccr", "mae", "mze"],
"cv_metric": "mze"

},
```

Ahora, mostraremos que indican cada una de las variables que aparecen en la configuración general:

- **basedir:** ruta que contiene los *datasets*. Admite tanto ruta relativa como absoluta y sólo está permitido adjuntar un valor, es decir, una única ruta.
- **datasets:** nombre de los *datasets* que se emplearán en el experimento. Este nombre deberá ser el nombre de una carpeta contenida en la ruta proporcionada en la variable *basedir*.
- **hyperparam_cv_folds:** número de *folds* que se utilizarán en la validación cruzada cuando se optimicen los hiperparámetros.
- **jobs:** número de procesos que lanzados durante la validación cruzada. Si se utiliza -1 se usarán todos los núcleos del procesador del equipo.
- **input_preprocessing:** tipo de preprocesamiento que aplicar a los datos, ya sea estandarización (*std*) o normalización (*norm*). En caso de no incluir esta variable o poner como valor de esta una cadena vacía no se realizará preprocesado a los datos.
- **output_folder:** ruta en la que se almacenan los resultados obtenidos.
- **metrics:** nombre de las métricas de rendimiento que se usarán en el experimento, pueden incluirse una o más, tal y como viene en el experimento de ejemplo proporcionado antes.

- **cv_metric:** es la métrica que se utilizará para determinar los mejores parámetros para cada clasificador.

Estas variable suelen contener un valor por defecto en el archivo de configuración *config.py*, hay algunas de ellas que es obligatorio cumplimentar como la ruta y los *datasets* a utilizar durante el experimento.

A.6.2. Configuraciones de los algoritmos

Esta parte del fichero de configuración contiene los algoritmos que se usarán en el experimento como sus parámetros con los que configuramos dicho algoritmo, aunque estos suelen tener unos valores por defecto en el constructor de la clase que los contiene. Se puede especificar más de un valor para un parámetro de forma que durante el experimento se probará con todos los valores.

Ahora se mostrará un ejemplo de este apartado del fichero de configuración JSON para los algoritmos realizados en este TFG, NNPOM y NNOP.

```
    "configurations": {

        "NNPOM": {

"classifier": "NNPOM",
"parameters": {
            "epsilonInit": 0.5,
            "hiddenN": [5,10,20,30,40,50],
            "iterations": [250, 500],
            "lambdaValue": [0, 0.01, 1]

            }
        },
```

```
        "NNOP": {

    "classifier": "NNOP",
    "parameters": {
                "epsilonInit": 0.5,
                "hiddenN": [5,10,20,30,40,50],
                "iterations": [250, 500],
                "lambdaValue": [0, 0.01, 1]

                }
            }
        }
```

Por último, procedemos a explicar cada una de las variables que pertenecen a este apartado:

- **classifier:** Clasificador a utilizar. Este se puede especificar con la ruta relativa al algoritmo de *scikit-learn* o con el nombre de la clase que contiene al algoritmo.
- **parameters:** Parámetros a optimizar para obtener un modelo de red neuronal óptimo.

A.6.3. Parámetros de los algoritmos

Los parámetros son los mismos para los algoritmos NNPOM y NNOP y hay un ejemplo de estos en la Sección A.6.2. Además estos han sido especificados y explicados detalladamente en el Capítulo 11, siendo estos los parámetros que se introducen como argumentos de entrada al constructor de la clase *__init__*.

A.6.4. Formato de las Bases de Datos

Las bases de datos vienen incluidas en la carpeta *datasets* del proyecto, dentro de la cada carpeta que contenga una base de datos, estas vendrán con el prefijo *train* si son bases de datos para entrenamiento y *test* si por el contrario son para generalización. Estas vendrán por norma general con el formato *.csv* a excepción de que el archivo se trate de una partición de una base de datos, entonces su formato será el número de partición que es, p.e. *train_contact-lenses.15*.

A.6.5. Ejecutando un experimento

Para lanzar un experimento tan solo hay que ejecutar la siguiente línea de código mediante el intérprete Python indicando en *experiment_file.json* el nombre del archivo de configuración.

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

Esta ejecución puede ocasionar dos posibles problemas:

El primero de ellos es que no se puede reproducir un mismo elemento ya que por defecto la semilla es aleatoria. Esto se puede solucionar fijando el valor de la semilla de la siguiente forma:

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

El segundo es que el paquete *sacred* puede llegar a imprimir mucha información por pantalla que quizá al usuario no le interesa. Esto se puede evitar agregando el parámetro *-l ERROR* a la línea de ejecución:

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

A.6.6. Formato de los resultados

Si durante la ejecución del experimento no se produce ningún error, toda la información producida será almacenada en la carpeta *output_folder* indicada en la configuración. Cada experimento genera una carpeta con un nombre que sigue esta codificación: exp-año-mes-día-hora-minuto-segundo

Dentro de esta carpeta se encontrará una carpeta por cada ejecución cuyo nombre sigue esta codificación: dataset-configuration, siendo configuration el nombre asignado para la dicha configuración en el archivo de configuración.

Dentro de estas carpetas podemos encontrar: la carpeta *models* que contiene un archivo por partición con los mejores modelos obtenidos en la fase de validación, la carpeta *predictions* con un archivo con predicciones realizadas por el mejor modelo de cada partición y un archivo csv que contendrá las métricas especificadas en el archivo de configuración tanto para *train* como para *test* y los tiempos de computación. Cada fila del fichero corresponde con el mejor resultado de cada partición.

Además de las carpetas creadas por cada ejecución, también se podrán encontrar dos ficheros con las métricas obtenidas por los conjuntos de *train* y las obtenidas por los de *test*, respectivamente. Cada fichero contendrá una fila por cada *datasets* y en él se almacenan la media de cada métrica obtenida así como la media de los tiempos de ejecución.

El repositorio vendrá con alguna ejecución almacenada en la carpeta *my_runs* para que el usuario pueda ver de forma gráfica lo explicado hasta ahora y se familiarice con esta organización de ficheros.