Grado en Ingeniería Informática

Escuela Politécnica Superior de Jaén

**UNIVERSIDAD DE JAÉN**
*EPSJ*

Trabajo de Fin de Grado

# IMPLEMENTACIÓN DE MODELOS PARA *DATA SCIENCE*


**Alumno:** **Ramón Díaz Valenzuela**

Tutor: Prof. D. Antonio Jesús Rivera Rivas
Francisco Javier Pulgar Rubio

Dpto: Informática


**Noviembre, 2016**

Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Informática

Don Antonio Jesús Rivera Rivas y Don Francisco Javier Pulgar Rubio, tutores del Trabajo de Fin de Grado titulado: Implementación de Modelos para *Data Science*, que presenta Ramón Díaz Valenzuela, autorizan su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, Septiembre de 2016

El alumno:

Ramón Díaz Valenzuela

Los tutores:

NOMBRE RIVERA RIVAS ANTONIO JESUS - NIF 26015863B

Firmado digitalmente por NOMBRE RIVERA RIVAS ANTONIO JESUS - NIF 26015863B
Fecha: 2016.10.31 10:03:46 +01'00'

Francisco Javier Pulgar Rubio

Firmado digitalmente por Francisco Javier Pulgar Rubio
Nombre de reconocimiento (DN): cn=Francisco Javier Pulgar Rubio, o=Universidad de Jaén, ou, email=fpulgar@ujaen.es, c=ES
Fecha: 2016.10.31 10:12:49 +01'00'

Antonio Jesús Rivera Rivas

y

Francisco Javier Pulgar Rubio

Índice

# 1. INTRODUCCIÓN

En este primer capítulo se pretende hacer una introducción al trabajo de fin de grado. Se trata de un proyecto de tipo **teórico-experimental** centrado en algunas de las técnicas incluidas en el área de la Ciencia de Datos (del inglés *Data Science*). El objetivo del trabajo es el estudio de estas técnicas para su posterior implementación. Finalmente, se llevará a cabo una experimentación tomando como base la implementación de los métodos estudiados en la fase anterior, así como, un análisis de los resultados obtenidos.

## 1.1 Introducción al Trabajo

En la era actual, la cantidad de datos que se guardan en dispositivos de almacenamiento está en constante aumento. Estos datos provienen de muy diversas fuentes: bancos, hospitales, centrales meteorológicas, Facebook... [datacenterknowledge, 2013] En general, se almacenan datos de forma electrónica en todos los ámbitos de la sociedad.

Este hecho ha dado lugar a la aparición del término “BIG DATA”. Dicho término, hace referencia a las técnicas y procedimientos utilizados para tratar grandes cantidades de datos de manera que pueda obtenerse provecho de los mismos. La dificultad del trabajo con este tipo de problemas se suele describir mediante tres V's: *volumen* de datos a tratar, *velocidad* de generación de los mismos y *variedad* de sus formatos.

Los algoritmos de aprendizaje automático tradicionales se deben adaptar para poder abordar el problema. Una de las estrategias más utilizadas en el tratamiento de grandes volúmenes de información es la de distribuir el tratamiento de los datos sobre un conjunto de unidades de procesamiento (cluster) en lugar de procesarlos sobre uno solo.

En este contexto, se define el objetivo principal del proyecto a realizar, que consiste en adaptar dos técnicas de aprendizaje automático de forma que pueda distribuirse su ejecución sobre un conjunto de unidades de procesamiento de forma concurrente.

De estas dos técnicas, la primera utilizará un enfoque basado en la metaheurística de evolución diferencial y la otra se basará en la optimización mediante enjambres de partículas. Ambas técnicas serán utilizadas para optimizar los parámetros que definen una red neuronal de base radial.

Para realizar la implementación de los métodos mencionados se hará uso de la plataforma Spark. Ésta tiene como principal objetivo proporcionar una interfaz para la programación de cluster de computación; simplificando y optimizando la tarea de distribuir el procesamiento sobre distintas máquinas en una red. Para ello, la plataforma hace uso del paradigma Map-Reduce.*[Dean y Ghemawat,2008]*

Una vez finalizada la implementación se realizarán una serie de experimentos utilizando distintos conjuntos de datos. La experimentación se realizará en un clUster propiedad de la Universidad de Jaén. El objetivo principal de esta experimentación es comparar los tiempos de ejecución entre un enfoque tradicional y otro enfoque basado en computación distribuida.

## 1.2 Motivación

Como se ha destacado anteriormente, en los últimos años la cantidad de datos que se recogen y almacenan mediante dispositivos electrónicos está creciendo exponencialmente. Baste como ejemplo la bolsa de Nueva York, que genera entre 4 y 5 terabytes de datos cada día.[White, 2014]. Empresas y organizaciones pueden sacar provecho de estos datos si consiguen extraer conocimiento útil de los mismos. Por este motivo, la demanda de profesionales formados en el campo de la ciencia de datos está también en ascenso en la actualidad.

Los enfoques tradicionales de algoritmos de aprendizaje automático tienen limitaciones para abordar problemas que involucren las cantidades de datos a las que se hace referencia, entendiendo por tradicionales aquellas que se ejecutan sobre una única unidad de procesamiento. Una posible solución a este problema es adaptar esos algoritmos para que puedan distribuir su procesamiento sobre una red de unidades de procesamiento en lugar de ejecutarse sobre una sola unidad. Esta es la estrategia que se sigue en el proyecto.

Para la implementación de los modelos de este trabajo se ha utilizado el framework Spark. Se trata de un framework de código abierto que simplifica mucho la tarea de repartir el procesamiento en un conjunto de nodos. El framework se distribuye junto con una biblioteca con diferentes funcionalidades y algoritmos de aprendizaje(MLLib). A día de comienzo del proyecto, esta librería no contiene ningún algoritmo de aprendizaje basado en redes neuronales de base radial. Por este motivo, resulta interesante la implementación de algún método que utilice ese tipo de redes. En el trabajo que se trata, se desarrollarán dos variantes basadas en esta técnica.

## 1.3 Propósito

El propósito principal del trabajo es la implementación de algoritmos de aprendizaje automático capaces de distribuir el procesamiento de datos en distintas máquinas de forma paralela.

Como se ha mencionado en la introducción, se implementarán dos métodos: uno basado en la metaheurística evolución diferencial y el otro en optimización mediante enjambres de partículas.

## 1.4 Objetivos

1. Estudiar la definición del problema de "Big Data" y las técnicas de Ciencia de Datos.

2. Estudiar las herramientas disponibles para tratar la escalabilidad horizontal en problemas de "Big Data", particularmente utilizando el paradigma Map-Reduce en entornos con Spark.

3. Instalación del software necesario para desarrollar y ejecutar estos algoritmos (Scala, IntelliJ, Apache Spark, Hadoop, HDFS, etc.).

4. Implementación de algoritmos de aprendizaje automático o minería de datos utilizando las herramientas y paradigmas mencionados.

5. Analizar las mejoras obtenidas con las implementaciones realizadas en comparación con los enfoques tradicionales.

6. Redactar una memoria que recoja el trabajo realizado.

## 1.5 Metodología

1. Estudiar documentación reciente sobre la temática del TFG.

2. Instalación y configuración del software que se va a utilizar (herramientas, bibliotecas y código anterior).

3. Implementación de los modelos elegidos.

4. Evaluación del comportamiento de los algoritmos usando distintos conjuntos de datos y comparando el rendimiento de estos con el enfoque tradicional.

5. Redacción de la memoria incluyendo el proceso de documentación, diseño, implantación y evaluación de los modelos.

## 1.6 Resultados Esperados

Con la elaboración de este trabajo se pretenden obtener los siguientes resultados:

1. Informe que documente la información obtenida como resultado de la investigación de las diferentes temáticas que se abordan: BIG DATA, Minería de Datos, Redes Neuronales, Algoritmos Evolutivos, optimización mediante Evolución Diferencial y optimización mediante Enjambres de Partículas principalmente. Dicho informe se incluirá en la memoria final.

2. Implementación de dos modelos de aprendizaje automático; uno de ellos hará uso de optimización mediante Evolución Diferencial y el otro usará Enjambres de Partículas.

3. Informe que contenga los resultados de la experimentación realizada sobre los dos algoritmos anteriores, así como, la comparativa de resultados según

se ejecuten en una sola unidad de procesamiento o varias de forma distribuida.

4. Memoria del trabajo.

## 1.7 Estructura del documento

La estructura de este trabajo intenta seguir, en la medida de lo posible, la recomendada en la página web de la Escuela Politécnica de la Universidad de Jaén.

En el capítulo 2 (Antecedentes), se pretende plasmar los conocimientos adquiridos en la fase de investigación de este proyecto. Son los fundamentos sobre los que posteriormente se ha realizado la implementación de los modelos.

En el capítulo 3 (Material y Métodos), se describe el proceso de implementación de los modelos, así como, de las herramientas, lenguajes y tecnologías utilizadas.

El capítulo 4 (Resultados y Discusión), está reservado para mostrar los resultados de la experimentación con los algoritmos implementados.

En capítulo 5 (Conclusiones) se pretende reflejar lo aprendido en el proceso de desarrollo de este proyecto.

En el capítulo 6 se muestra un listado con la bibliografía a la que se hace referencia en este trabajo.

Finalmente, en el anexo 1, se describe de forma breve la estructura del repositorio que contiene el código fuente de los modelos; y en el anexo 2 se explica la configuración para la ejecución.

## 1.8 Agenda y planificación del proyecto

### 1.8.1 Planificación del proyecto

El proyecto se inicia con fecha a 19 de Febrero de 2016 y tiene fecha de finalización prevista para el 28 de Octubre de 2016. Este proyecto tiene asignado un tiempo de trabajo de 12 créditos ECTS. Como cada crédito ECTS hace referencia a

un tiempo de trabajo de entre 25 y 30 horas, entonces, el tiempo total del que se dispone para llevar a término el proyecto es de 300-360 horas. Estas se irán repartiendo a lo largo del periodo de desarrollo del mismo.

El proceso de realización del trabajo podría hacerse identificando tareas que puedan ejecutarse de forma concurrente en el tiempo, el uso de diagramas PERT es muy útil en estos casos. Este enfoque es el adecuado si el trabajo va a ser realizado por un grupo de personas. En este caso, no obstante, como el trabajo va a ser realizado por una sola persona, se va a realizar una asignación de tareas secuencialmente.

#### 1.8.1.1 Identificación de tareas

El primer paso para realizar la planificación del proyecto es identificar las tareas que han de realizarse para llevar a término el mismo, las dependencias que existen entre las mismas (para poder ordenarlas en el tiempo de forma adecuada) y el tiempo necesario para realizar cada una de ellas. Estos datos están recogidos en la tabla 1.1.

| Id | Tarea | Tiempo | Fecha inicio | Fecha fin | Predecesoras |
|---|---|---|---|---|---|
| 1 | Estudio del estado del arte en Redes Neuronales de Base Radial | 50 días | 19/02/16 | 28/04/16 | |
| 2 | Estudio de la sintaxis del lenguaje de programación Scala | 14 días | 29/04/16 | 18/05/16 | 1 |
| 3 | Estudio del funcionamiento de la plataforma Spark | 14 días | 19/05/16 | 07/06/16 | 2 |
| 4 | Definición de los casos de uso | 3 días | 08/06/16 | 10/06/16 | 3 |
| 5 | Realización de los diagramas de clases | 3 días | 13/06/16 | 15/06/16 | 4 |
| 6 | Realización de los diagramas de secuencia | 3 días | 16/06/16 | 20/06/16 | 5 |
| 7 | Implementación del algoritmo 1 (ED) | 15 días | 21/06/16 | 11/07/16 | 6 |
| 8 | Pruebas del algoritmo 1 (ED) | 3 días | 12/07/16 | 14/07/16 | 7 |
| 9 | Implementación del algoritmo 2 (PSO) | 15 días | 15/07/16 | 04/08/16 | 8 |
| 10 | Pruebas del algoritmo 2 (PSO) | 3 días | 05/08/16 | 09/08/16 | 9 |
| 11 | Experimentación en el cluster con el algoritmo 0 (Genético) | 10 días | 10/08/16 | 23/08/16 | 10 |
| 12 | Experimentación en el cluster con el algoritmo 1 (ED) | 10 días | 24/08/16 | 06/09/16 | 11 |
| 13 | Experimentación en el cluster con el algoritmo 2 (PSO) | 10 días | 07/09/16 | 20/09/16 | 12 |
| 14 | Análisis de los resultados obtenidos en el cluster | 2 días | 21/09/16 | 22/09/16 | 13 |
| 15 | Informe de resultados de la experimentación | 1 día | 23/09/16 | 23/09/16 | 14 |
| 16 | Análisis de resultados | 1 día | 26/09/16 | 26/09/16 | 15 |
| 17 | Revisión de la memoria | 24 días | 27/09/16 | 28/10/16 | 16 |
| 18 | Final de proyecto | 0 días | 28/10/16 | 28/10/16 | 17 |

**Tabla 1.1: Tareas del Trabajo**

El diagrama de Gantt que las representa se muestra en la siguiente página.

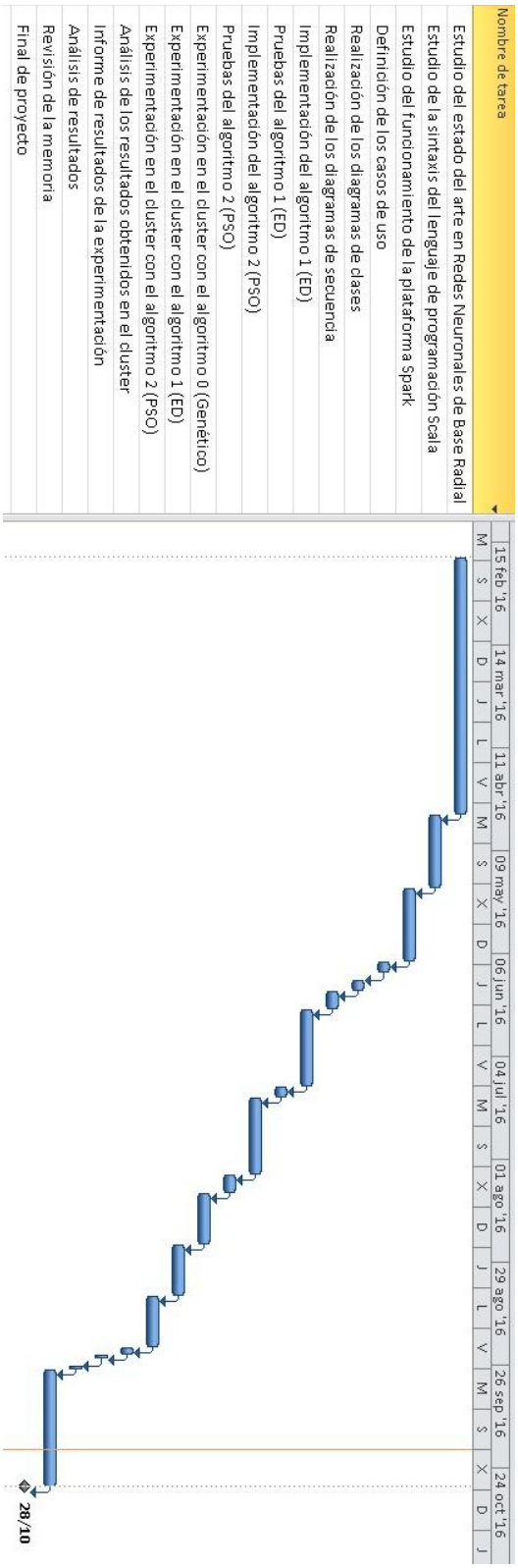



**Ilustración 1.1: Diagrama de Gantt del Trabajo**

## 1.8.2 Estimación de costes

Para estimar el coste del proyecto, es necesario tener en cuenta tres conceptos:

- El equipo de desarrollo del proyecto.
- El tiempo de uso del cluster para realizar las pruebas y experimentaciones.
- El tiempo de trabajo del encargado de llevar a cabo el proyecto.

### 1.8.2.1 El equipo de desarrollo

Para el desarrollo del proyecto no es necesario un equipo especialmente potente. Se va a utilizar un equipo portátil de gama media con las siguientes características:

- Procesador intel i3. 2.3GH
- Cantidad de memoria RAM: 8 GB

El precio del equipo es de **449€**

### 1.8.2.2 Tiempo de uso del cluster

Como se ha comentado anteriormente, se medirá el rendimiento de los algoritmos a implementar en un cluster. Éste es propiedad de la universidad de Jaén, se estima que el precio por hora de uso del mismo será de 1,3 €.

Se necesita usar el cluster durante 30 días como se ha detallado en la sección de tareas. El coste total se calcula como:

TOTAL CLUSTER = 1,3€ * (30 días * 24 horas) = **936€**

### 1.8.2.3 Tiempo de trabajo

El perfil del encargado de desarrollo del proyecto puede enmarcarse como analista-programador. El precio medio por hora de este perfil ronda los 40€. Como se ha indicado anteriormente, el tiempo de proyecto ha de oscilar entre las 300 y 360

horas de trabajo. La media son 330 horas. Por tanto el coste de trabajo del desarrollador será de:

Coste Analista Programador = 330horas * 40€ = **13200€**

#### 1.8.2.4 Total costes

El total de costes estimados queda pues reflejado en siguiente operación:

| | |
|---|---|
| Equipo de desarrollo | +449€ |
| Uso de cluster | +936€ |
| Coste Analista Programador | +13200€ |
| **TOTAL** | **14585€** |

# 2. ANTECEDENTES

El propósito de este capítulo es el de describir la teoría y conceptos en los que se fundamenta el proyecto.

## 2.1 Big data

Este término, también denominado *datos a gran escala*, hace referencia a la gestión y análisis de enormes volúmenes de datos que no pueden ser tratados de manera convencional, ya que superan los límites y capacidades de las herramientas software habitualmente utilizadas para la captura, gestión y procesamiento de los datos.[wikipedia, 1]

## 2.2 Extracción de conocimiento en bases de datos

Tal y como se ha dicho anteriormente, hoy en día se manejan grandes cantidades de datos. Estos vienen de fuentes muy heterogéneas y se generan a gran velocidad (cada día mas).

Sería, por tanto, muy interesante si se pudiera sacar provecho de ellos. Eso es justamente lo que se intenta con un proceso de extracción de conocimiento en bases de datos (en inglés KDD, *knowledge database discovery*). Se define el descubrimiento de conocimiento en bases de datos como " *el proceso no trivial de identificar patrones válidos, novedosos, potencialmente útiles, y, en última instancia, comprensibles a partir de los datos*" [Pérez Godoy, 2010]

Este proceso puede dividirse en una serie de pasos. El mostrar una definición detallada de cada uno de ellos se escapa de los límites de este trabajo; pero se considera oportuno hacer una introducción a los mismos. De forma resumida estos son:

1. Pre-procesamiento de los datos.
2. Minería de Datos.
3. Post-procesamiento de los datos.

Generalmente, cuando se va intentar extraer conocimiento de un conjunto de datos en el mundo real, estos vienen dados en "bruto". Esto es, provienen

probablemente de distintas fuentes y, por tanto, tienen distintos formatos. Además, se puede dar el caso que los datos presenten diferentes problemas, como pueden ser: datos erróneos, datos repetidos o datos perdidos. En el primer paso del proceso de extracción de conocimiento de bases de datos (pre-procesamiento de los datos) se intenta solventar estos problemas "homogeneizando" los datos que se van a analizar, de forma que su tratamiento sea lo más sencillo posible. Este paso puede dividirse a su vez en otras fases tales como limpieza, reducción y selección entre otras.

Debido a la importancia que tiene la fase de Minería de Datos (segundo paso) en la elaboración del proyecto, existe un apartado completo para profundizar en ella. Como introducción, decir que la Minería de Datos es una disciplina que intenta producir nuevo conocimiento a partir de los datos. Para conseguirlo, se aplican distintos algoritmos a los datos dependiendo del tipo de tarea que se quiera realizar(predicción de series, clasificación, agrupamiento, extracción de reglas...).

El último paso se corresponde al de post-procesamiento de los datos. En esta etapa se intentan interpretar los resultados de los patrones obtenidos en el paso anterior para extraer el conocimiento que implican. En este momento, se mide también la calidad de los patrones obtenidos. Un patrón se considera útil si es fácilmente entendible, potencialmente útil, novedoso y es válido sobre los datos de prueba con algún grado de certeza[Pérez Godoy, 2010].

## 2.3 Minería de datos

Es la fase central y más representativa del descubrimiento de conocimiento en bases de datos, como se ha explicado en el apartado anterior. Su finalidad es conseguir crear nuevo conocimiento útil, para ello construye un modelo a partir de los datos con los que trabaja. Este modelo es una especificación de patrones y relaciones que hay entre los datos. A partir de este modelo se pueden hacer predicciones, entender mejor los datos o dar explicación a situaciones pasadas.

La función que realiza la minería de datos la venían haciendo desde tiempo atrás las técnicas de análisis de datos tradicionales. Sin embargo, como se ha

señalado anteriormente, en los últimos años se ha producido una explosión en la creación y almacenamiento de datos, lo que ha supuesto la generación de demasiados datos para tratarlos mediante enfoques tradicionales, de ahí el nacimiento de la minería de datos.

Se ha dicho anteriormente que la minería de datos tiene como objetivo la búsqueda de conocimiento subyacente en los datos. No obstante, según el objetivo que se pretenda obtener a partir del conocimiento adquirido, se puede clasificar la minería de datos en dos ramas:

- Minería de datos **predictiva**. El objetivo del modelo generado en este caso es el de predecir valores desconocidos (valor de clase) usando una serie de variables conocidas (características).

- Minería de datos **descriptiva**. En este caso, el modelo generado busca el explicar o resumir los datos en lugar de hacer una predicción.

Las tareas que se van a abordar en la fase de implementación y experimentación en este proyecto van a ser de tipo predictivo, por tanto, no se va a profundizar más en la minería de datos descriptiva.

### 2.3.1 Tareas de la minería de datos predictiva

La minería de datos predictiva engloba distintos tipos de tareas. Las principales son clasificación, regresión, predicción de series temporales, categorización y priorización.

#### 2.3.1.1 Clasificación

La tarea que realizarán los algoritmos que van a ser implementados en este trabajo es la de **clasificación**. Es seguramente la tarea más utilizada en minería de datos. Para describir su funcionamiento es necesario definir primero lo que es una instancia. El conjunto de datos con el que se va a generar el modelo puede verse como una tabla, en la que cada fila sería una instancia. Cada una de estas instancias tiene asignada una clase. El objetivo del modelo generado es el de ser capaz de predecir la clase de una instancia.

Una definición más rigurosa de clasificación en minería de datos la encontramos en [Ferry y otros,2007]:

*"Clasificación (o discriminación): los ejemplos se presentan como un conjunto de pares de elementos de dos conjuntos, δ={<e,x> : e ∈ E, s ∈ S}, donde S es el conjunto de valores de salida. Los ejemplos e, al ir acompañados de un valor de S, se denominan comúnmente ejemplos etiquetados <e,s> y, en consecuencia, δ se denomina conjunto de datos etiquetado. El objetivo es aprender una función λ: E -> S, denominada clasificador, que represente la correspondencia existente en los ejemplos, es decir, para cada valor de E tenemos un único valor para S. Además, S es nominal, es decir, puede tomar un conjunto de valores $c_1$, $c_2$, ... , $c_m$, denominados clases (cuando el número de clases es dos, tenemos lo que se llama clasificación binaria). La función aprendida será capaz de determinar la clase para cada nuevo ejemplo sin etiquetar, es decir, dará un valor de S para cada valor de E. Ejemplos: clasificar un mensaje de correo electrónico como spam o no, clasificar entre varios medicamentos cuál es el mejor para una determinada patología..."*

### 2.3.2 Técnicas de minería de datos

Los métodos de minería de datos están basados en distintas técnicas. Para cada técnica puede haber varios algoritmos que la implementen y estos serán capaces de abordar una o varias de las tareas descritas en el apartado anterior. A continuación se describen algunas de estas técnicas:

- Árboles de decisión. Se utilizan para tareas de clasificación. Los algoritmos de minería de datos que se basan en ellos generan un árbol como modelo. Cada hoja del árbol representa una clase del problema que se trata. Cada nodo es una decisión simple que llevará finalmente a la hoja que representa la clase de la instancia que se está tratando.

- Inducción de reglas. Son técnicas que generan reglas lógicas del tipo: antecedente -> consecuente. El antecedente está formado por un predicado que se evaluará como verdadero o falso y el consecuente contiene el valor de clase del ejemplo.

- Algoritmos evolutivos. *Son métodos de optimización y búsqueda de soluciones basados en los postulados de la evolución biológica*.[wikipedia, 2]

- Redes neuronales. *Se trata de un paradigma de aprendizaje y procesamiento automático inspirado en el sistema nervioso de los animales*.[Pérez Godoy, 2010]

De estas cuatro técnicas serán las dos últimas las que se aplicarán al desarrollo de los métodos a implementar en este proyecto.

### 2.3.3 Evaluación y validación del modelo

Las medidas que se usan para evaluar un modelo dependen de la tarea a la que esté destinado. Así, en el caso de la extracción de reglas se usa la cobertura y la confianza de las reglas generadas. En el caso de la regresión se mide el error cuadrático medio. Y, en el caso más interesante desde el punto de vista de este proyecto, el de la clasificación, se utiliza la precisión del modelo.[Pérez Godoy, 2010]

Se entiende por validación del modelo al proceso seguido para establecer su fiabilidad. Normalmente, cuando se va a generar un modelo se parte de un conjunto de datos para hacerlo. Este conjunto de datos será el que se utilizará tanto para entrenar (generar) el modelo como para evaluar su fiabilidad(en este proyecto será medir la precisión del mismo). Es común dividir este conjunto de datos en dos subconjuntos: conjunto de entrenamiento y conjunto de test. De este modo, se utiliza el conjunto de entrenamiento para generar el modelo y el conjunto de test para validar la fiabilidad del mismo.

La intención de dividir el conjunto total de datos de esta manera es la de obtener unos resultados más realistas. Cuando se entrena un modelo con un determinado conjunto de datos, los patrones y relaciones existentes en el mismo se refieren a esos datos. Pero puede ser (y es lo más probable) que esos patrones y relaciones no se correspondan tan fielmente con datos nuevos.

Cuando la diferencia de representación del modelo de los datos de entrenamiento con respecto a los de test es acusada, se dice que el modelo sufre de

sobre-entrenamiento. Esto es, obtenemos mucho mejores resultados cuando probamos el modelo con los datos de entrenamiento a cuando lo probamos con los datos de test.

Existen distintas formas de dividir el conjunto de datos para realizar el entrenamiento y la validación. Algunos de los métodos son los siguientes.

- Resustitución: Los datos de entrenamiento y test son los mismos. Es normal que se obtengan mejores resultados en la evaluación que los que luego se obtendrán con datos nuevos.
- Partición (Holdout): Se hace una partición del conjunto de datos. Un subconjunto de la partición se utiliza para realizar el entrenamiento y el otro para testearlo.
- Bootstrapping: Se realiza entrenamiento y test usando subconjuntos del conjunto de datos. Estos subconjuntos son generados cogiendo elementos al azar del conjunto de datos y con reemplazamiento.
- *Leaving-one-out*: Se divide el conjunto de datos en n particiones iguales. n-1 de ellas son destinadas al entrenamiento del modelo y la última para testearlo.
- Validación cruzada (*k-fold*): El usuario elige un número entero *k*, este será el número de particiones en las que se dividirá el conjunto de datos. Se crean por tanto k conjuntos de igual tamaño a partir del conjunto de datos original. Los conjuntos son creados de forma aleatoria y sin reemplazamiento. Una vez creados los subconjuntos se realizará el entrenamiento con k-1 conjuntos y con el último no utilizado se evaluará el modelo. El proceso se repite k veces usando un subconjunto distinto siempre para realizar la evaluación y los restantes para hacer el entrenamiento del modelo. La evaluación final será la media de las k evaluaciones anteriores.

En la fase de experimentación de este proyecto se realizará la evaluación y validación del modelo usando el método de validación cruzada, ya que, es un método exhaustivo y fiable.

## 2.4 Computación Flexible

Como se ha mencionado anteriormente, uno de los propósitos de este proyecto es estudiar la forma de tratar grandes cantidades de datos. En el proceso de extracción de conocimiento y, más concretamente, de minería de datos hay varias tareas que pueden formularse como problemas de optimización y búsqueda. Hacerlo mediante un enfoque tradicional, es decir, mediante una búsqueda exhaustiva dentro del espacio de soluciones no es posible en el caso que se trata debido a la magnitud de dicho espacio.

La computación flexible (soft-computing), hace referencia a una serie de metodologías con una característica común, la tolerancia a la incertidumbre. Son metaheurísticas que no realizan una búsqueda exhaustiva en el espacio de soluciones. Este tipo de metodologías son aptas para tratar esos espacios de búsqueda grandes ya que permiten encontrar soluciones aceptables con un coste computacional bajo. Dentro de este tipo de metodologías se encuentran la computación evolutiva y las redes neuronales, entre otras. Se utilizarán estas dos técnicas en el desarrollo de este trabajo para realizar la implementación de los algoritmos.

Englobados dentro de la computación evolutiva se pueden encontrar, a su vez, distintos métodos de optimización. Los que interesan para este trabajo son los dos siguientes:

- Algoritmos genéticos
- Evolución diferencial

El segundo se utiliza en uno de los métodos a implementar en este trabajo. Existe un tercer método de optimización que no está contenido dentro de la

computación evolutiva, la **optimización por medio de enjambres de partículas**. Este es el método utilizado en la otra técnica a implementar.

Los tres métodos citados en el anterior párrafo se encargan de la generación de las redes neuronales que son el método en que se basa el procedimiento de minería de datos que se trata en este proyecto.

## 2.5 Computación evolutiva

La mayoría de los algoritmos de optimización tradicionales utilizan un enfoque determinístico para la búsqueda de soluciones. La solución encontrada pues depende y es siempre la misma en base al punto de partida en que se inicie la ejecución del algoritmo.

Los algoritmos de este tipo suelen mostrar, entre otros, dos problemas:

- El encontrar la solución óptima al problema depende del punto de inicio de la búsqueda.
- Suelen estancarse en óptimos locales.

Estos problemas pueden resolverse incluyendo un componente de aleatoriedad en el proceso de búsqueda de soluciones. Esta es una de las características de la computación evolutiva. Esta se define como un conjunto de algoritmos, todos ellos con una componente estocástica, basados en la teoría de la evolución de Darwin. [Ferry y otros,2007]

Estos algoritmos tienen la ventaja de poder encontrar soluciones en todo el espacio de búsqueda. Esto es debido a que guardan un equilibrio entre exploración y explotación. La exploración del espacio de búsqueda es inducida por los mecanismos de mutación y cruce que se explicarán a continuación. La intensificación(explotación) por otra parte es facilitada por el mecanismo de selección del algoritmo. Al incluir aleatoriedad en el proceso de búsqueda, los algoritmos evolutivos, no tienen porque caer siempre en el mismo óptimo local aún partiendo desde el mismo punto en diferentes ejecuciones. Del mismo modo,

encontrar la solución óptima globalmente no depende del punto de partida del algoritmo.

Como se ha dicho, los algoritmos evolutivos basan su funcionamiento en los mecanismos de evolución y adaptación de los seres vivos en la naturaleza. Están compuestos por una población de individuos, donde cada individuo representa una posible solución al problema que se esté tratando. Los individuos se combinan entre ellos y sufren mutaciones a medida que el algoritmo va avanzando. Asociado a los individuos existe una función *fitness* que mide lo buena que es cada solución. Generación tras generación los mejores individuos, aquellos con mejor valor de función *fitness*, van sustituyendo a aquellos que son menos aptos. No obstante, también existen mecanismos que permiten a algunos individuos no tan buenos seguir viviendo, de este modo se evita la convergencia prematura hacia óptimos locales.

Los individuos se combinan entre ellos y sufren mutaciones. Estas acciones se realizan a través de operadores de cruce y mutación respectivamente. Estos operadores incluyen una componente aleatoria y, en general, se aplican con una probabilidad preestablecida. Con el operador de cruce, dos o más individuos son mezclados entre ellos para dar lugar a una nueva solución candidata. El operador de mutación se aplica sobre los individuos induciendo cambios aleatorios en los mismos, de este modo se favorece la diversificación de soluciones y se escapa de óptimos locales. Estos operadores se van aplicando generación tras generación a la población de individuos. Tras cada generación se eligen los individuos que pasan a la siguiente iteración. Se prima a aquellos que tienen una mejor puntuación de la función *fitness* de modo que se va convergiendo a mejores soluciones; pero no se eliminan por completo a las peores soluciones, si no que se les da una oportunidad más baja de sobrevivir, de este modo se favorece la diversificación.

Existen dos elementos importantes asociados a la computación evolutiva: genotipo y fenotipo. Se habla de genotipo para referirse a la codificación que se utiliza en el sistema para representar a un individuo. El fenotipo hace referencia a los valores de la codificación para un individuo en concreto.

Dentro de la clasificación de computación evolutiva se pueden encontrar varios algoritmos, para el caso que se trata en este proyecto los que realmente interesan son los algoritmos genéticos y la evolución diferencial; este último, implementado en este trabajo. El otro método de optimización usado en el trabajo, la optimización por medio de enjambres de partículas, no está incluido dentro de la computación evolutiva. Aún así, al ser un método de optimización y búsqueda como los otros, guarda bastantes similitudes con ellos. Se dedicará un apartado distinto para él más adelante.

### 2.5.1 Elementos de un algoritmo evolutivo

Los elementos básicos que componen cualquier tipo de algoritmo evolutivo son los siguientes:

- Esquema de representación de los individuos
- Función objetivo (de adaptación o fitness)
- Población
- Mecanismo de selección de los padres
- Operadores de mutación y cruce
- Forma de reemplazamiento (de que forma la antigua generación es sustituida por la nueva)

A continuación se describen cada uno de estos componentes.

#### 2.5.1.1 Esquema de representación

Es la forma en que se codificarán los individuos de la población, el genotipo citado anteriormente. Esta forma irá en función al problema que se esté tratando con el algoritmo.

#### 2.5.1.2 Función de adaptación

Es la función que se aplica a un individuo para conocer lo buena que es la representación que representa el mismo, es decir, lo buena que es la solución. Su valor puede interesar maximizarlo o minimizarlo según el problema que se trate. La elección de los individuos que sobreviven de una generación a otra debe estar en gran medida dirigida por su valor de adaptación (intensificación).

#### 2.5.1.3 Población

La labor de la población en el algoritmo es la de mantener un conjunto de soluciones en cada momento durante la ejecución del algoritmo. De este modo se favorece que existan soluciones repartidas por todo el espacio de búsqueda y así escapar de los óptimos locales.

Esto es especialmente importante al inicio de la ejecución del algoritmo, donde ha de favorecerse la diversificación de soluciones para ocupar todo el espacio. Suele inicializarse por ello la población de forma aleatoria. A medida que las generaciones van pasando, son los individuos más aptos los que normalmente van sobreviviendo y las soluciones, en consecuencia, van agrupándose en núcleos(intensificación) hasta el final del algoritmo.

#### 2.5.1.4 Mecanismo de selección de los padres

El mecanismo de selección favorece a las mejores soluciones dentro de la población. Se trata de elegir a los padres (individuos) que se cruzarán entre ellos para dar lugar a una nueva remesa de individuos. Una vez se ha creado esta nueva remesa, se aplica de nuevo una selección sobre todos los individuos para decidir cuales pasarán a la siguiente generación.

La selección de los individuos debe de hacerse primando a los mejores pero también hay que dar una oportunidad de supervivencia a los individuos menos buenos con el objetivo de no caer en óptimos locales. Existen distintas estrategias de selección de los padres, entre ellas se pueden encontrar las siguientes:

- Selección proporcional: La selección de los padres se hace de forma aleatoria. A cada individuo se le asigna una probabilidad de ser elegido proporcional al valor de su función de adaptación.

- Selección aleatoria: Se eligen los padres de forma aleatoria; este método tiene el inconveniente de que no favorece la convergencia hacia las mejores soluciones.

- Selección por torneo: Se eligen los padres en tandas de k individuos. En cada tanda se escogen k individuos al azar de la población, con reemplazamiento. De los k individuos, se escoge a aquel con mejor valor fitness para que se cruce. A mayor valor de k, por tanto, mayor intensificación de las mejores soluciones; pues hay mayor probabilidad de que en la tanda aparezca una buena solución.

#### 2.5.1.5 Operadores de cruce y mutación

Una vez seleccionados los individuos que van a cruzarse en la actual iteración se aplicarán estos dos operadores.

En primer lugar se aplica el operador de cruce. Al contrario que en los procesos biológicos de la naturaleza, el cruce en los algoritmos evolutivos puede darse entre más de dos individuos. Cada cruce da lugar a uno o más descendientes. No hay una regla fija para este operador con respecto a la forma en que se combinan los padres y es común que se utilicen componentes aleatorias en el mismo.

Una vez se ha aplicado el operador de cruce para generar un conjunto nuevo de individuos, se aplica el operador de mutación sobre los mismos. Este operador se aplica con una probabilidad preestablecida sobre cada individuo. Esta probabilidad suele ser baja y se basa en aplicar cambios aleatorios sobre las soluciones creadas en el anterior paso. De este modo se diversifica la población.

#### 2.5.1.6 Estrategia de reemplazamiento

Mediante esta estrategia se eligen los individuos que pasan a la siguiente iteración. Hay dos modelos fundamentales en los que se puede clasificar un algoritmo evolutivo en base a este criterio: generacionales o estacionarios.

En los primeros, la población generada a partir de los cruces y mutaciones sustituye directamente a la antigua generación para pasar a la siguiente iteración. En el caso de los estacionarios, se hace una selección de los individuos sobre las dos poblaciones: la antigua y la generada en los pasos anteriores. En este último caso es necesario mantener un equilibrio entre exploración e intensificación.

Favorecer la exploración es favorecer que sobrevivan a la siguiente generación soluciones que no sean muy buenas. De este modo, se favorece que la población quede más dispersa sobre el espacio de búsqueda y, por tanto, pueda explorarse el mismo. Si se prima en exceso este fenómeno se corre el riesgo de no converger a una buena solución.

En el otro extremo está la intensificación. Favorecerla implica dar mayor probabilidad de supervivencia a las buenas soluciones. De este modo, la población se va congregando en núcleos en torno a soluciones prometedoras. Si se prima en exceso este fenómeno se convergerá rápidamente a una solución que con mucha probabilidad será un óptimo local.

Un equilibrio adecuado entre ambas, por tanto, es la mejor forma de abordar el problema.

### 2.5.2 Esquema básico de un algoritmo evolutivo

Un esquema básico de ejecución de un algoritmo evolutivo en general podría ser el siguiente[Pérez Godoy, 2010]:

```
Inicialización de la población
Calculo función de adaptación
Repetir
   Aplicación de cruce y mutación
   Cálculo de función de adaptación de descendientes
```

```
    Selección de individuos
    Sustitución de la población actual
Hasta (condición de parada)
```

## 2.6 Algoritmos Genéticos

Los algoritmos genéticos son una de las clases de algoritmos evolutivos. Han sido aplicados con éxito en multitud de problemas de optimización, tales como: el cableado de rutas, control adaptativo, modelado cognitivo, problemas de transporte, el problema del viajante de comercio, optimización de consultas a bases de datos... [Michalewicz,1999]

### 2.6.1 Inicialización

Tradicionalmente los Algoritmos Genéticos utilizaban un sistema de representación binaria para las soluciones. Posteriormente, es el caso que trata este trabajo, se empezaron a utilizar otro tipo de codificaciones. La población inicial se inicializa de forma aleatoria.

### 2.6.2 Cruce

Este operador es el método que se utiliza principalmente para generar cambios en la población en el caso de los algoritmos genéticos; esta es la característica que los diferencia principalmente del resto de algoritmos evolutivos. Normalmente el cruce se realiza entre dos individuos que dan lugar a uno nuevo mediante la recombinación de ambos.

### 2.6.3 Mutación

Este operador se utiliza para producir cambios en la población al igual que el anterior, pero en el caso de los algoritmos genéticos genera menor diversidad que el operador de cruce. Se suele utilizar con una muy baja probabilidad y su funcionamiento se basa en provocar variaciones aleatorias en las componentes que definen a un individuo de la población.

### 2.6.4 Selección

El esquema de selección de los algoritmos genéticos proporcional al valor de la función de adaptación. Ejemplos de este tipo de selección son los anteriormente señalados: selección por torneo y selección proporcional.

### 2.6.5 Pseudocódigo

El pseudocódigo general de un algoritmo genético coincide con el especificado en el apartado anterior para los algoritmos evolutivos.

## 2.7 Evolución diferencial (ED)

Como anteriormente se ha citado, la evolución diferencial es otro de los algoritmos incluidos en la clasificación de programación evolutiva. Este se ha utilizado con gran éxito para la resolución de problemas de optimización continua. La idea principal de este enfoque es utilizar la diferencia entre vectores como mecanismo para producir cambios en la población.[Talbi, 2009]

### 2.7.1 Inicialización

En este método cada solución es codificada como un vector D-dimensional de números reales (siendo D el número de variables predictivas en el caso de minería de datos que se trata en este proyecto) . La población inicial se inicializa de forma aleatoria. Mas específicamente, para cada componente de este vector, se especifican dos valores suelo y techo. Posteriormente, a la hora de inicializar la población, se le da un valor a cada individuo de forma aleatoria; estableciendo el valor de los mismos desde una distribución uniforme comprendida entre los valores suelo y techo preestablecidos para cada componente del vector. Una definición más formal a esta inicialización la encontramos también en [Talbi, 2009] y se describe a continuación.

Si se denota los parámetros suelo y techo respectivamente como $[x_j^{lo}, x_j^{hi}]$, entonces, el mecanismo de inicialización de la población podría definirse con la expresión 2.1.

$$x_{ij} = x_j^{lo} + rand_j[0,1] \cdot (x_j^{hi} - x_j^{lo}),\ i \in [1,k],\ j \in [1,D] \qquad \textit{(Fórmula 2.1)}$$

Donde:

- k es el tamaño de la población.
- $rand_j[0,1]$ es una distribución uniforme entre 0 y 1.
- El índice i hace referencia al individuo que se está inicializando.
- El índice j hace referencia a la componente del vector que representa al individuo.

### 2.7.2 Mutación

El proceso de mutación dará lugar a una nueva población de individuos mutados a partir de la original. Se denominará, en este proceso, a cada uno de los individuos de la población actual *individuo objetivo* ( $v^i_{obj}$ ) y a los individuos de la nueva población: *individuos mutantes* ( $v^i_{mut}$ ); en ambos casos i ∈ [1,n] y n es el tamaño de la población actual.

Para cada $v^i_{obj}$ se genera un vector mutante $v^i_{mut}$. Este $v^i_{mut}$ se obtiene mediante combinaciones lineales entre los vectores de la población, donde el concepto de distancia entre vectores tiene importancia. Un tipo de mutación (que no el único) es el descrito en la siguiente expresión como [Sung-Kwun y otros, 2012], DE/Rand/1/B:

$$v^i_{mut} = x_c + B(x_a - x_b) \qquad \textit{(Fórmula 2.2)}$$

donde:

- $x_a$, $x_b$ y $x_c$ son vectores escogidos al azar de la población.
- B, se denomina factor de escala. Es un parámetro que se especifica antes de la ejecución del algoritmo.

### 2.7.3 Cruce

Una vez ya se tiene la población $v_{obj}$ y la población $v_{mut}$, se realiza una recombinación entre ambos para obtener un descendiente $v^{i}_{hijo}$. La expresión que define esta combinación es la siguiente[Sung-Kwun y otros, 2012]:

$$v_{hijo}^{ij} = \begin{cases} v_{mut}^{ij} & \text{si rand} < \text{CR or } j = j^{randindex} \\ x_{obj}^{ij} & \text{de otro modo} \end{cases} \qquad \textit{(Fórmula 2.3)}$$

donde:

- j es el índice de la componente del vector que se está tratando
- CR ∈ [0,1] se denomina *crossing rate* y es un parámetro especificado antes de la ejecución.
- $j^{randindex}$ es un índice cogido al azar.
- i es el indice del individuo en la población donde i ∈ [1,n]

### 2.7.4 Selección

$v^{i}_{hijo}$ sustituye a $v^{i}_{obj}$ si mejora en el valor de la función de adaptación a este último.

### 2.7.5 Pseudocódigo

Un ejemplo de pseudocódigo que muestra los pasos que se han descrito se encuentra en [Talbi, 2009] (ilustración 2.1):

Ilustración 2.1 Pseudocódigo ED

**Input:** Parameters: $F$ (scaling factor), $CR$ (crossover constant).
Initialize the population (uniform random distribution) ;
**Repeat**
  **For** $(i = 1, i \leq k, i++)$ **Do** /* Each individual */
    **Mutate and Recombine:**
      $j_{\text{rand}} = int(\text{rand}_i[0, 1] \cdot D) + 1$ ;
      **For** $(j = 1, j \leq D, j++)$ **Do**
        **If** $(\text{rand}_j[0, 1]) < CR)$ or $(j = j_{\text{rand}})$ **Then**
          $u_{ij} = v_{ij} = x_{r3j} + F \cdot (x_{r1j} - x_{r2j})$
        **Else**
          $u_{ij} = x_{ij}$
      **Replace:**

$$x_i(t+1) = \begin{cases} u_i(t+1) & \textbf{If } f(u_i(t+1)) \leq f(x_i(t)) \\ x_i(t) & \textbf{Otherwise} \end{cases}$$

  **End For**
**Until** Stopping criteria /* ex: a given number of generations */
**Output:** Best population or solution found.

## 2.8 Optimización mediante enjambres de partículas (PSO)

Como su propio nombre indica, la optimización por enjambres de partículas (en inglés *Particle Swarm Optimization - PSO*) es otra técnica de optimización y búsqueda. No se engloba dentro de la programación evolutiva, ya que, no basa su funcionamiento en el mecanismo biológico de evolución de las especies. Aún así, si que lo está en la de las metaheurísticas basadas en poblaciones, al igual que la programación evolutiva. Esto hace que comparta ciertas similitudes con los algoritmos géneticos y la evolución diferencial.

Es ésta una técnica estocástica, al igual que las comprendidas en la computación evolutiva, inspirada en el comportamiento social de los animales; como las bandadas de aves y los bancos de pescado. El algoritmo fue introducido por primera vez por James Kennedy y Russel Eberhart en 1995. En PSO, cada individuo, llamado en este caso partícula, se mueve a través del espacio de búsqueda ajustando su posición a partir de su propia experiencia y también la de sus vecinos.[Rahmani,2008]

### 2.8.1 Inicialización

Al igual que ocurre en el método de Evolución Diferencial, cada solución es codificada como un vector D-dimensional de números reales (siendo D el número de variables predictivas en el caso de minería de datos que se trata en este proyecto) . La población inicial se inicializa de forma aleatoria. Mas específicamente, para cada componente de este vector, se especifican dos valores suelo y techo. Posteriormente, a la hora de inicializar la población, se le da un valor a cada individuo de forma aleatoria; estableciendo el valor de los mismos desde una distribución uniforme comprendida entre los valores suelo y techo preestablecidos para cada componente del vector.

### 2.8.2 Movimiento de las partículas

A la hora de mover una partícula en el espacio se tienen en cuenta tres factores:

- La posición y velocidad actual de la partícula.
- La mejor posición que ha ocupado la partícula durante toda su historia.
- La mejor posición global que ha sido ocupada por cualquier partícula del enjambre durante toda la ejecución. La propuesta original del método no hace referencia al enjambre completo en este caso; si no, a un vecindario definido. En este trabajo, no obstante, el vecindario será todo el enjambre.

La siguiente expresión define el citado movimiento:

$$x_i(t+1) = x_i + v_i(t+1) \qquad \textit{(Fórmula2.4)}$$

Donde:

- $x_i(t+1)$ será la posición de la partícula i en la siguiente iteración.
- $x_i$ es la posición actual de la partícula i.

- $v_i(t+1)$ es el vector de velocidad que se aplicará a la partícula para moverla a su posición en la siguiente iteración.

Y para el cálculo del vector de velocidad , $v_i(t+1)$, se aplica la siguiente expresión:

$$v_{ij}(t+1)=v_{ij}(t)+c_1 r_{1j}[y_{ij}(t)-x_{ij}(t)]+c_2 r_{2j}[y'_j(t)-x_{ij}(t)] \quad \textit{(Fórmula 2.5)}$$

Donde:

- $v_{ij}(t)$ es la velocidad actual de la partícula.
- $c_1$ y $c_2$ son dos constantes de aceleración
- $r_{1j}$ y $r_{2j}$ son dos números aleatorios calculados según una distribución aleatoria en el intervalo [0,1]
- $x_{ij}(t)$ es la posición de la partícula en el momento actual, para la componente j del vector; con j comprendido en [1,D]. Siendo D la dimensión del vector.
- $y_{ij}(t)$: la mejor posición que ha ocupado la partícula $x_i$ hasta el momento para la componente j del vector.
- $y'_j(t)$: la mejor posición que ha ocupado cualquier partícula del enjambre en toda la historia de la ejecución hasta el momento actual para la componente j del vector.

En la fórmula 2.5 se pueden distinguir tres componentes, cada una de ellas es uno de los sumandos. La primera componente es la velocidad actual de la partícula; es conocida como el *término memoria* de la partícula.

La segunda componente incluye la mejor posición ocupada por la partícula en su historia, es conocida como el *término cognitivo*. Expresa el deseo de la partícula de volver a su mejor posición pasada.

La tercera componente incluye la mejor posición global ocupada durante toda la ejecución del algoritmo por cualquier partícula de la población, es conocida como

el *término social*. Expresa el conocimiento adquirido por todo el enjambre. [Alexandridis y otros, 2013]

Como en cualquier otro método de optimización y búsqueda, en PSO es necesario mantener un balance entre diversificación e intensificación. Por este motivo se establece un límite superior e inferior al valor que la velocidad puede adoptar, así se limita una exploración excesiva del espacio de búsqueda. En caso de que el resultado obtenido en la fórmula 2.5 se sobrepase estos límites, la velocidad de la partícula se ajustará al límite correspondiente.

### 2.8.3 Selección

En PSO, no existe un operador de selección tal y como se entiende en los algoritmos evolutivos. Todo el enjambre se mueve y actualiza su posición; el funcionamiento es similar al de un enfoque generacional en un algoritmo genético donde la nueva generación sustituye a la antigua.

### 2.8.4 Pseudocódigo

Un ejemplo de pseudocódigo que muestra los pasos que se han descrito se encuentra en [Talbi, 2009] :

Random initialization of the whole swarm ;
**Repeat**
  Evaluate $f(x_i)$ ;
  **For all** particles $i$
    Update velocities:
      $v_i(t) = v_i(t-1) + \rho_1 \times (p_i - x_i(t-1)) + \rho_2 \times (p_g - x_i(t-1))$ ;
    Move to the new position: $x_i(t) = x_i(t-1) + v_i(t)$ ;
    **If** $f(x_i) < f(pbest_i)$ **Then** $pbest_i = x_i$ ;
    **If** $f(x_i) < f(gbest)$ **Then** $gbest = x_i$ ;
    Update$(x_i, v_i)$ ;
  **EndFor**
**Until** Stopping criteria

**Ilustración 2.2: Pseudocódigo PSO**

## 2.9 Redes Neuronales Artificiales

Las Redes Neuronales Artificiales (RNAs) son un tipo de algoritmos de aprendizaje automático inspiradas en el funcionamiento del sistema nervioso de los animales. Pueden aplicarse a distintas tareas, tales como, el reconocimiento de patrones, la compresión de información, agrupamiento y clasificación entre otras. Algunas de esas tareas pertenecen al campo de la minería de datos, en este proyecto se utilizarán las RNAs para abordar el problema de clasificación.

El sistema nervioso animal se basa en un conjunto de neuronas enlazadas entre sí mediante conexiones sinápticas capaces de conducir estímulos eléctricos. Del mismo modo, las redes neuronales artificiales están formadas por un conjunto de nodos (llamados neuronas) interconectados entre sí por aristas que conducen valores; el sistema puede verse como un grafo dirigido. Las conexiones entre los nodos están reguladas por pesos, también conocidos como *valores sinápticos*.

### 2.9.1 Arquitectura

Una neurona artificial (Ilustración 2.3) es un elemento de cómputo que realiza una operación simple. Recibe una serie de entradas(x's) a través de las conexiones de entrada y origina una salida. Cada una de las conexiones de entrada tiene asociado un peso(w); si este es positivo, la conexión se entiende como *excitadora,* en caso de ser negativo, se dice que la conexión es *inhibidora*.[Pérez Godoy, 2010]

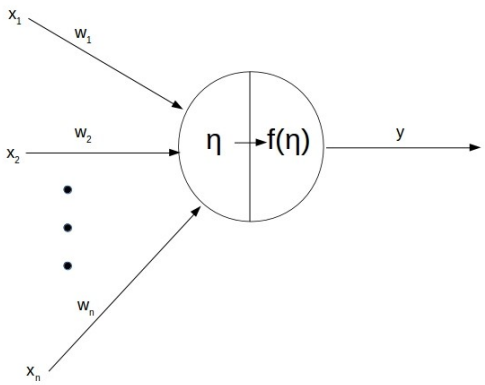



**Ilustración 2.3: Neurona Artificial**

La componente de procesamiento de la neurona está formada comúnmente por tres funciones encadenadas: base(ή), activación y salida. Las funciones de activación y salida no existen en las implementaciones que trata este trabajo. Por tanto, la salida de final (y) de las neuronas se corresponde con la salida de la función base.

Hay distintos tipos de funciones base, pero las que se van a utilizar en este proyecto son conocidas como *funciones de base radial*. La salida de estas funciones representa una distancia a un determinado patrón.

### 2.9.2 Topología

Una red neuronal está compuesta por un conjunto de neuronas artificiales interconectadas entre ellas. A la forma en que están conectadas unas con otras se le llama topología de la red.

Normalmente, la topología de la red se divide en capas de neuronas dispuestas de manera secuencial. De este modo, las salidas de cada capa anterior son las entradas a la siguiente capa. Basándose en este funcionamiento se definen tres tipos de capas:

- Capa de entrada: Es la primera capa de la red. Recoge las entradas de la red y las pasa a la siguiente capa, generalmente sin aplicar procesamiento alguno sobre los datos.
- Capa(s) ocultas: Son las capa(s) que siguen a la capa de entrada. Puede haber varias encadenadas y son las encargadas de procesar los datos.
- Capa de salida: Recibe los datos procesados por las capas ocultas y los envía directamente como la salida de la red.

En base a la topología de la red hay que tener en cuenta como parámetros: el número de neuronas que compone cada capa, el número y tipo de capas y el grado de interconexión entre las mismas.

### 2.9.3 Entrenamiento

El entrenamiento de la red neuronal consiste en ajustar los pesos de las conexiones entre neuronas de modo que la salida de la red se aproxime a los ejemplos con los que se va a trabajar.

Se distinguen dos enfoques principales para realizar el entrenamiento de la red. El aprendizaje supervisado donde se conoce la salida que la red debe dar para cada ejemplo que se utiliza en el entrenamiento. Y el aprendizaje no supervisado, donde la salida para los ejemplos usados en el entrenamiento es desconocida.

El enfoque utilizado en este proyecto es el de aprendizaje supervisado. Cuando se utiliza este método para el entrenamiento de redes, se compara el resultado esperado con el obtenido realmente por la red y, dependiendo de si ambos coinciden, se van ajustando los pesos de las conexiones con el objetivo de obtener la salida deseada.

## 2.10 Redes Neuronales de Base Radial

Las redes neuronales de base radial(*Radial Basis Function Network* - RBFN) son un tipo específico de redes neuronales artificiales. Los métodos implementados en este proyecto se basan en este tipo de redes. Éstas se caracterizan por su topología y porque el tipo de función que implementan sus neuronas es una *función de base radial*(RBF, también notadas como Φ).

### 2.10.1 Arquitectura

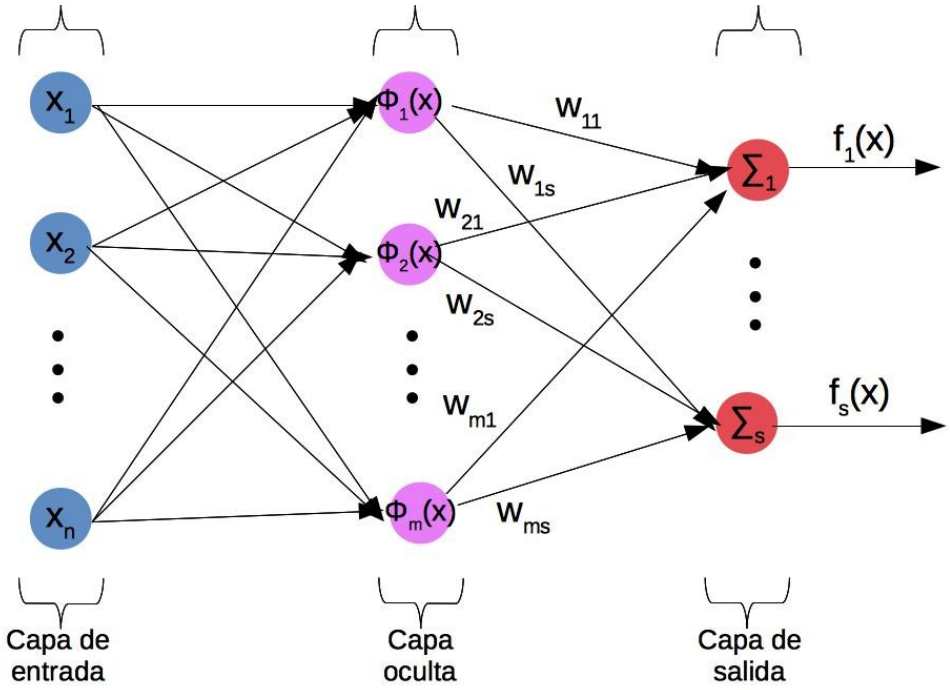
Una red de funciones de base radial está compuesta por tres capas:

- Capa de entrada. Esta capa contiene un nodo por cada característica del problema, considerado como *n*. No realiza procesamiento alguno sobre los datos, su función es la de pasar las entradas a la siguiente capa, la capa oculta.

- Capa oculta. Contiene *m* nodos. Cada uno de ellos representa una RBF que procesa los datos de entrada pasados por la anterior capa. La salida de esta

capa es enviada a la siguiente capa, la capa de salida. Los resultados de las RBFs son modificados en base a pesos asignados a las aristas de salida de sus nodos correspondientes.

- Capa de salida. Puede tener uno o varios nodos, considerado como *s*. Cada uno de ellos devuelve como resultado la sumatoria de los valores de sus aristas de entrada.




**Illustración 2.4: Arquitectura de una red neuronal de base radial**

## 2.10.2 Funciones de base radial(Φ)

Como se ha mencionado anteriormente, el valor de retorno de una función de base radial es la distancia a un patrón de referencia. Son de la forma: $\Phi: R^n \rightarrow R$ y se definen por tres parámetros[Pérez Godoy, 2010]:

- Centro($c_i$): Es un vector con la misma dimensión que los que se aceptan como entradas.

- Radio($r_i$): Permite escalar la distancia de los puntos de entrada con respecto al centro.

- Tipo de función de activación.

#### 2.10.2.1 Diseño de Redes de Funciones de Base Radial

Una RBFN viene determinada por los siguientes parámetros:

1. Número de nodos de la capa de entrada.
2. Número de nodos de la capa oculta(m).
3. Tipo de RBF que implementan los nodos de la capa oculta.
4. Posición de los centros de las RBFs.
5. Radios de las RBFs
6. Pesos de las conexiones entre la capa oculta y la capa de salida
7. Número de nodos de la capa de salida (s)

El número de nodos de la capa de entrada y salida vienen determinados por el problema que se aborde. El tipo de RBFs que implementan los nodos de la capa oculta vienen, generalmente, dados de antemano.

Por tanto, el diseño de una RBFN, consistirá en establecer el **número de nodos** de la capa oculta, sus **centros** y sus **radios,** así como, los **pesos** entre la capa oculta y la capa de salida.

Existen diversas técnicas y métodos para calcular estos parámetros. En este proyecto se hace uso de dos de ellas: el algoritmo k-medias para la inicialización de los centros y radios de las RBF's; y la descomposición en valores singulares para el cálculo de los pesos entre la capa oculta y la de salida.

#### 2.10.2.2 Inicialización: Algoritmo de las k-medias

Es un algoritmo de agrupamiento (*clustering*) que divide el conjunto de vectores de entrada en k grupos. Esta división produce una partición de Voronoi[wikipedia, 3] del espacio de búsqueda. Cada uno de los grupos está representado por un vector llamado prototipo. El algoritmo necesita de algún tipo de medida de distancia entre los vectores de entrada para relacionarlos con su prototipo.

El algoritmo puede dividirse en los siguientes pasos:

1. Inicialización. Se asignan valores aleatorios a los prototipos.
2. Asignación. Se asigna, a cada vector del conjunto, el prototipo que tiene más cerca.
3. Actualización. Se calcula el vector media de cada uno de los grupos generados. Ese vector media sustituye al vector prototipo correspondiente en cada caso.
4. Repetir 2 y 3 hasta que las asignaciones no cambien.

#### 2.10.2.3 Entrenamiento: Descomposición en valores singulares

Es un algoritmo para resolver sistemas de ecuaciones mediante matrices[Cliff, 1983][Golub y Van Loan, 1996]. En el diseño de redes neuronales de base radial puede usarse para el cálculo de los pesos entre la capa oculta y la capa de salida.

## 2.11 Map Reduce

*Map Reduce es un modelo de programación y su implementación asociada que tiene como propósito el procesar grandes conjuntos de datos.*[Dean y Ghemawat, 2004]

Una de sus ventajas es la simplicidad, ya que oculta los detalles de la distribución del procesamiento. De este modo, programadores sin conocimientos en computación paralela pueden hacer uso del mismo.

El modelo toma un conjunto de entrada formado por pares clave-valor para dar lugar a otro conjunto de salida de pares clave-valor. El proceso se puede dividir en dos funciones que serán escritas por el usuario: map y reduce.

La primera función, map, recibe uno de los pares clave-valor de entrada para dar lugar a un nuevo conjunto intermedio de pares clave-valor. La librería interna de map-reduce agrupará luego todos estos pares clave-valor intermedios según su clave y los pasará a la función reduce.

La función reduce, acepta un conjunto de valores y una clave. Su propósito es combinar todos estos valores para obtener un conjunto reducido de datos (típicamente 1 o ninguno).

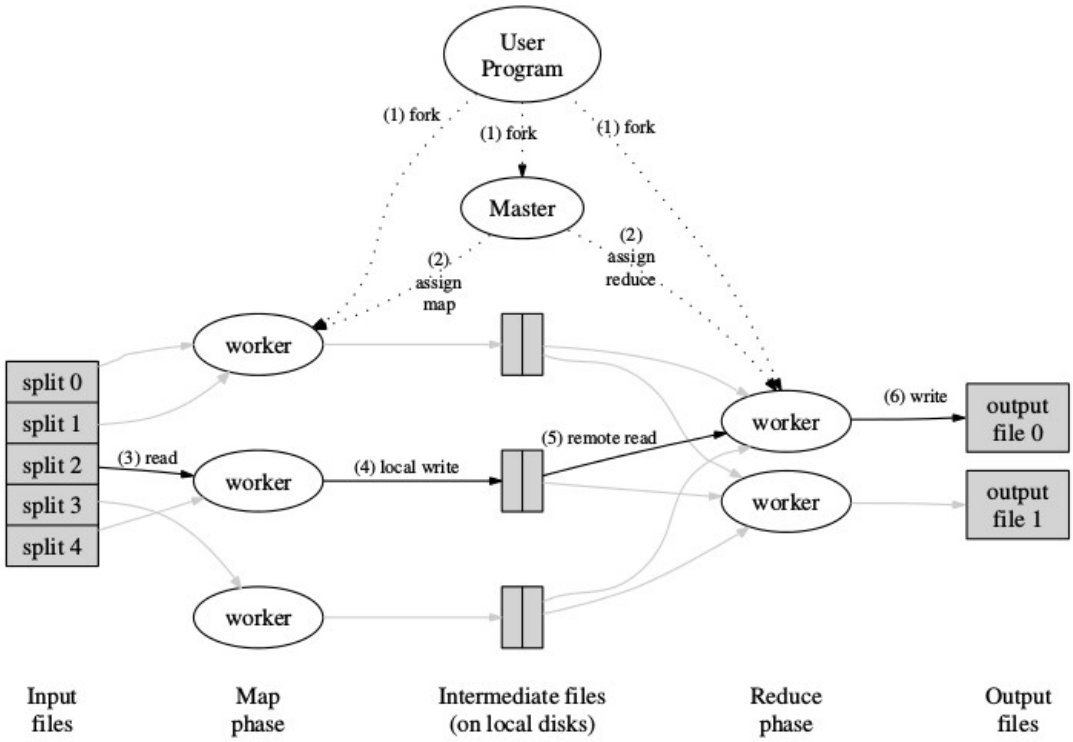



**Ilustración 2.5: Esquema de ejecución de Map Reduce**

# 3. MATERIAL Y MÉTODOS

En este capítulo se pretende describir los materiales y las técnicas utilizadas para llevar a cabo la implementación del proyecto.

## 3.1 Evolución Diferencial

Como se ha mencionado en el primer capítulo, el primer método de aprendizaje automático que se va a implementar estará basado en la técnica de evolución diferencial. El esquema de ejecución del método es el siguiente:

1. Carga del conjunto de datos.
2. Inicialización de la población.
3. Optimización de la población.
4. Devolver los resultados de clasificación obtenidos por el mejor individuo de la población.

A continuación se detallan cada una de estas fases.

### 3.1.1 Carga del conjunto de datos

Los conjuntos de datos que el sistema será capaz de tratar estarán codificados en el formato propio de KEEL. Estos pueden descargarse de su página web[keel, 1].

### 3.1.2 Inicialización de la población

La inicialización de los individuos se realiza aplicando el algoritmo de las k-medias introducido en el capítulo anterior. Como se mencionó entonces, este algoritmo necesita de una medida de distancia entre los vectores que se van a tratar. En el problema que se aborda, estos vectores podrán contener valores enteros, reales y nominales. Sobre los valores enteros y reales puede calcularse la distancia euclídea, pero no sobre los nominales. Por este motivo se ha utilizado la distancia HVDM[Pérez Godoy, 2010] con la que se pueden tener en cuenta los tres tipos de valores.

### 3.1.3 Optimización de la población

En este paso se aplica el método de optimización y búsqueda de evolución diferencial. El objetivo del mismo es el de encontrar la mejor solución posible dentro del espacio de búsqueda. A continuación se describen sus componentes.

#### 3.1.3.1 Esquema de Representación

Dentro de la población, cada individuo representa una red neuronal de base radial. Los parámetros que definen a un individuo son los siguientes:

- Número de RBFs de su capa oculta
- Posición de los centros y tamaño de los radios de las RBFs
- Pesos de las conexiones entre la capa oculta y la capa de salida

El número de neuronas de la capa de entrada viene determinado por el número de variables de entrada del problema y el número de neuronas de la capa de salida se corresponde con el número de clases que tiene el mismo.

La función que implementa los nodos de la capa oculta es fija. Se trata de una función gaussiana definida por la siguiente expresión:

$$e^{-\left(\frac{(\vec{c}_i - \vec{x})^2}{r_i^2}\right)} \qquad \textit{(Fórmula 3.1)}$$

Donde:

- $\vec{c}_i$ es el centro de la RBF.
- $\vec{x}_i$ es el vector de entrada de la instancia que se está tratando.
- $r_i$ es el radio de la función.

### 3.1.3.2 Función de evaluación

Para evaluar la calidad de los individuos de la población es necesario utilizar alguna función. En este caso, la función, es la precisión de la red neuronal. Viene dada por la siguiente expresión:

$$1-\frac{e}{n} \qquad \text{(}\textit{Fórmula}\text{ 3.2)}$$

Donde:

- e: es el número de errores en la clasificación por parte de la RBFN
- n: es el número total de instancias clasificadas

### 3.1.3.3 Operadores de mutación

Se utilizan cuatro operadores de mutación definidos por las siguientes expresiones [Sung-Kwun y otros, 2012]:

Operador 1 (DE/Rand/1/β)

$$D_{mutant}(t+1)=D_c(t)+\beta(D_a(t)-D_b(t)) \qquad \text{(}\textit{Fórmula}\text{ 3.3)}$$

Operador 2 (DE/Best/1/β)

$$D_{mutant}(t+1)=D_{best}(t)+\beta(D_a(t)-D_b(t)) \qquad \text{(}\textit{Fórmula}\text{ 3.4)}$$

Operador 3 (DE/Rand/2/β)

$$D_{mutant}(t+1)=D_e(t)+\beta(D_a(t)-D_b(t)-D_c(t)-D_d(t)) \qquad \textit{(Fórmula 3.5)}$$

Operador 4 (DE/Best/2/β)

$$D_{mutant}(t+1) = D_{best}(t) + \beta(D_a(t) - D_b(t) - D_c(t) - D_d(t)) \quad \textit{(Fórmula 3.6)}$$

Donde:

- $D_x$: x ∈ {a, b, c, d}. Son individuos cogidos al azar de la población.
- $D_{best}$: Es el individuo con mejor valor de función de evaluación de la población.
- β ∈ [0,1]. Es un parámetro establecido antes de la ejecución llamado factor de escala.

En cada generación, se obtendrá una nueva población mediante la aplicación de estos operadores a la población existente. Se llamará a esta nueva población: *población mutante*. La implementación que se va a realizar difiere un poco de la definida en el apartado Antecedentes. Mientras que en la implementación definida anteriormente el tamaño de la población mutante era siempre el mismo que el de la población actual, en este caso el tamaño de la población mutante vendrá proporcionada de antemano, mediante un parámetro facilitado antes de la ejecución, siendo como máximo del tamaño de la población. Los cuatro operadores de mutación definidos, se van eligiendo al azar, todos con la misma probabilidad de ser elegidos, en el proceso de creación de la *población mutante*.

#### 3.1.3.4 Operador de cruce

El operador de cruce se define en la (Fórmula 2.3). Se aplica a toda la *población mutante* y da lugar a una nueva población llamada *población trial*.

Para la aplicación de este operador, a cada *mutante* se le asigna un individuo de la población escogido al azar. Este último se denomina *individuo objetivo*. El cruce entre el *mutante* y el *objetivo* da lugar al *trial*.

#### 3.1.3.5 Operador de selección

Los individuos de la población *trial* que superen en valor fitness a sus correspondientes *individuos objetivo*, sustituirán a estos últimos en la población; y pasarán, por tanto, a la siguiente generación. Aquellos que no lo hagan serán descartados.

## 3.2 Optimización mediante Enjambres de Partículas

Este método no entra dentro de la clasificación de los algoritmos evolutivos, por lo que no puede hablarse de operadores de mutación, cruce o selección. Como se ha indicado antes, sí que comparte con los dos métodos anteriores la función de evaluación y el esquema de representación de la población. En este caso, a la población se le llama enjambre y a los individuos se les llama partículas.

No hay un proceso de mutación, cruce y selección, como se ha dicho, si no que lo que se hace es ir actualizando las posiciones de las partículas en base a su posición actual, la mejor posición que han ocupado y la mejor posición que ha ocupado cualquier partícula a lo largo de la ejecución. En el capítulo anterior se hace una descripción más detallada de esta actualización de posiciones. El esquema de ejecución del método es el que sigue:

1. Carga del conjunto de datos.
2. Inicialización de la población.
3. Optimización de la población.
4. Devolver los resultados de clasificación obtenidos por el mejor individuo de la población.

### 3.2.1 Carga del conjunto de datos

Los conjuntos de datos que serán utilizados por el sistema serán los que pueden descargase de la página web de KEEL.[keel, 1]

### 3.2.2 Inicialización de la población

La inicialización de los individuos se realiza aplicando el algoritmo de las k-medias, este fue descrito en el capítulo 2. El algoritmo de las k-medias necesita de una medida de distancia entre los vectores que se van a tratar. En el caso de este proyecto, los valores que podrán contener estos vectores serán de tipo entero, real y nominal. La distancia eucídea puede aplicarse sobre valores enteros y reales, no así

sobre los nominales. Debido a este hecho, para calcular la distancia entre valores de tipo nominal se utilizará la distancia HVDM.[Pérez Godoy, 2010]

### 3.2.3 Optimización de la población

En este paso se aplica el método de optimización y búsqueda mediante enjambres de partículas. El objetivo de esta es el de encontrar la mejor solución posible dentro del espacio de búsqueda. Sus componentes serán descritas a continuación.

#### 3.2.3.1 Esquema de Representación

Cada individuo contenido en la población representa una red neuronal de base radial. Esta red neuronal viene definida por los siguientes parámetros:

- Número de RBFs de su capa oculta
- Posición de los centros y tamaño de los radios de las RBFs
- Pesos de las conexiones entre la capa oculta y la capa de salida

El número de variables de entrada del problema determina el número de neuronas de la capa de entrada de la red. Así mismo, el número de neuronas de la capa de salida vendrá determinado por el número de clases del problema.

Los nodos de la capa oculta implementarán siempre la misma función. Esta es de tipo gaussiana, se define según la siguiente expresión:

$$e^{-\left(\frac{(\vec{c}_i-\vec{x})^2}{r_i^2}\right)} \qquad \textit{(Fórmula 3.7)}$$

Donde:

- $\vec{c}_i$ es el centro de la RBF.
- $\vec{x}_i$ es el vector de entrada de la instancia que se está tratando.

- $r_i$ es el radio de la función.

#### 3.2.3.2 Función de evaluación

Es necesario disponer de una función de evaluación para evaluar la calidad de los individuos que conforman la población. En el caso que se trata será la precisión de la red neuronal. Esta viene definida según la siguiente expresión:

$$1-\frac{e}{n} \qquad \text{(Fórmula 3.8)}$$

Donde:

- e: es el número de errores en la clasificación por parte de la RBFN
- n: es el número total de instancias clasificadas

#### 3.2.3.3 Operador de movimiento de partículas

En cada iteración las partículas que componen el enjambre actualizan su posición. La forma en que la implementación realizará esta actualización de posiciones es la definida en el capítulo de antecedentes(2.8.2) de este trabajo.

#### 3.2.3.4 Operador de selección

Como se ha explicado anteriormente, este no es un método englobado dentro de los algoritmos evolutivos. Por tanto, no tiene un operador de selección como tal, pero se comporta igual que un algoritmo genético con un enfoque generacional donde la nueva población sustituye a la antigua. Esto es porque, en la optimización mediante enjambres de partículas, todas las partículas que componen el enjambre modifican su posición a cada iteración; es decir, sustituyen a la "población antigua".

## 3.3 Análisis, Diseño e Implementación de los métodos

En esta sección se desarrollan labores comprendidas en la ingeniería del software: análisis de requerimientos, diseño e implementación de los métodos de aprendizaje automático.

### 3.3.1 Requisitos

Este apartado está destinado a especificar los requisitos que el sistema ha de cumplir. Estos requisitos describen lo que el sistema "debe de hacer" y también restricciones sobre el mismo.

#### 3.3.1.1 Requisitos funcionales

Los requisitos funcionales definen las funciones que el sistema tiene que realizar. Estos requisitos describen lo que el sistema tiene que hacer y no el *cómo* debe hacerlo. Los requisitos de este trabajo son los que siguen:

- El sistema debe permitir el paso de parámetros al mismo a través de un fichero de texto.
- El sistema ha de leer el listado de datasets a tratar desde otro fichero de texto.
- El sistema debe ser capaz de interpretar la codificación de los ficheros de datos de la plataforma KEEL.
- El sistema ha de mostrar por pantalla y también reflejarlo en ficheros de salida tanto el estado de la ejecución como los resultados obtenidos.
- El sistema ha de optimizar los parámetros que definen la red neuronal usando en cada caso el algoritmo elegido.

#### 3.3.1.2 Requisitos no funcionales

Un requisito no funcional es una restricción situada sobre el sistema[Arlow J y otros, 2006]. En el caso del sistema que se trata en este trabajo son los siguientes:

- El sistema a implementar debe de escribirse en el lenguaje de programación *Scala*.
- El sistema a implementar debe hacer uso del *motor Spark*.

- El sistema a implementar debe poder ejecutarse de forma concurrente en un conjunto de unidades de procesamiento.

### 3.3.1.3 Casos de uso

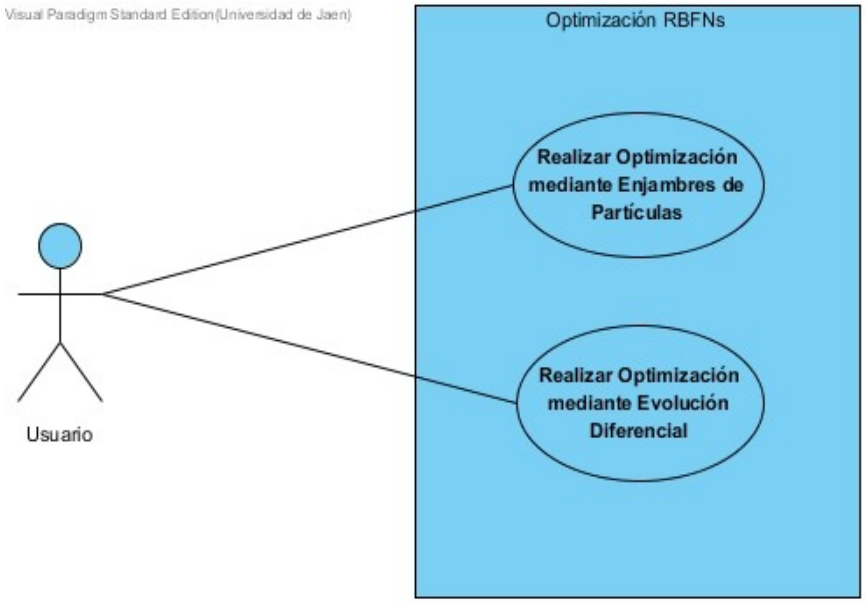



Ilustración 3.1: Caso de Uso

### 3.3.1.4 Narrativa de casos de uso

En este punto se pretende describir el funcionamiento de los casos de uso definidos en el punto anterior.

#### *3.3.1.4.1 Caso de uso "Optimización Enjambres de Partículas"*

| Caso de uso | Optimización Enjambres de Partículas |
|---|---|
| **Actor primario** | Usuario |
| **Condición de entrada** | El usuario indica la ruta con el fichero de parámetros y el fichero con las rutas a los conjuntos de datos. |
| **Condición de salida** | El sistema optimiza los parámetros que definen a la red neuronal usando un enfoque basado en enjambres de partículas. |
| **Flujo de eventos** | 1. El usuario ejecuta el algoritmo.<br>2. El sistema devuelve los resultados del entrenamiento de la red neuronal. |

Tabla 3.1: Caso de uso: Optimización mediante enjambres de partículas

#### *3.3.1.4.2 Caso de uso "Optimización Evolución Diferencial"*

| Caso de uso | Optimización Evolución Diferencial |
|---|---|
| **Actor primario** | Usuario |
| **Condición de entrada** | El usuario indica la ruta con el fichero de parámetros y el fichero con las rutas a los conjuntos de datos. |
| **Condición de salida** | El sistema optimiza los parámetros que definen a la red neuronal usando un enfoque basado en evolución diferencial. |
| **Flujo de eventos** | 1. El usuario ejecuta el algoritmo.<br>2. El sistema devuelve los resultados del entrenamiento de la red neuronal. |

**Tabla 3.2: Caso de uso: Optimización mediante evolución diferencial**

## 3.3.2 Diseño

Una vez se han definido las funciones que el sistema debe cumplir (requisitos) se debe especificar completamente cómo se implementará esta funcionalidad[Arlow J y otros, 2006], esta es la tarea a realizar en la fase de diseño del sistema.

Es una fase muy importante en el desarrollo del proyecto ya que de ella depende directamente la calidad de la solución que se genere y también su extensibilidad y posible reutilización para otros trabajos.

El proyecto va a escribirse en el lenguaje de programación *Scala* como se detallará posteriormente. Es un lenguaje que soporta el paradigma de orientación a objetos. Este paradigma, entre otras ventajas, facilita el desarrollo de sistemas más comprensibles, por este motivo se hará uso del mismo para el desarrollo de este trabajo.

Se van a utilizar dos de las herramientas definidas en UML[uml, 1] para realizar el diseño del sistema: el diagrama de clases y el diagrama de secuencia. En los puntos siguientes se detalla su funcionamiento y el resultado obtenido.

### 3.3.2.1 Diseño de clases

Mediante el diseño de clases se pretende descomponer el sistema en bloques(clases) y especificar las relaciones existentes entre los mismos. Es

deseable que las clases del sistema posean las siguientes características[Arlow J y otros, 2006]:

- Totalidad y suficiencia: "*Una clase debe hacer lo que los usuarios de la misma esperan; ni más ni menos"*[Arlow J y otros, 2006].
- Sencillez.
- Alta cohesión: "*Toda clase debería modelar un solo concepto abstracto y debería tener un conjunto de operaciones que soporten el propósito de la clase"*[Arlow J y otros, 2006].
- Bajo acoplamiento: Toda clase ha de ser lo más independiente posible, esto es, estar asociada con las clases estrictamente necesarias para llevar a cabo su labor.

A continuación se muestran los diagramas de clase diseñados para este trabajo.

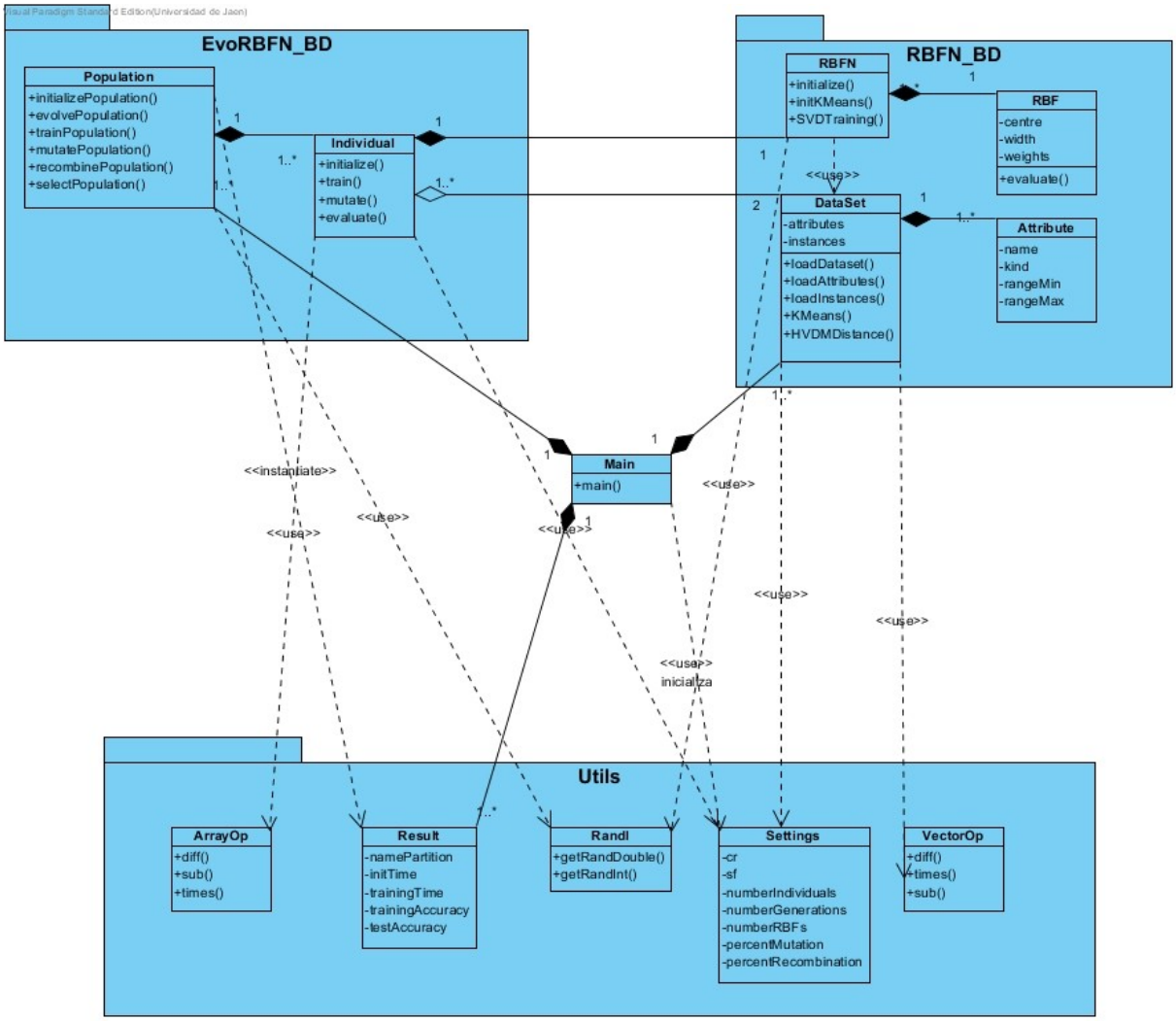



**Ilustración 3.2: Diagrama de clases ( Evolución Diferencial )**

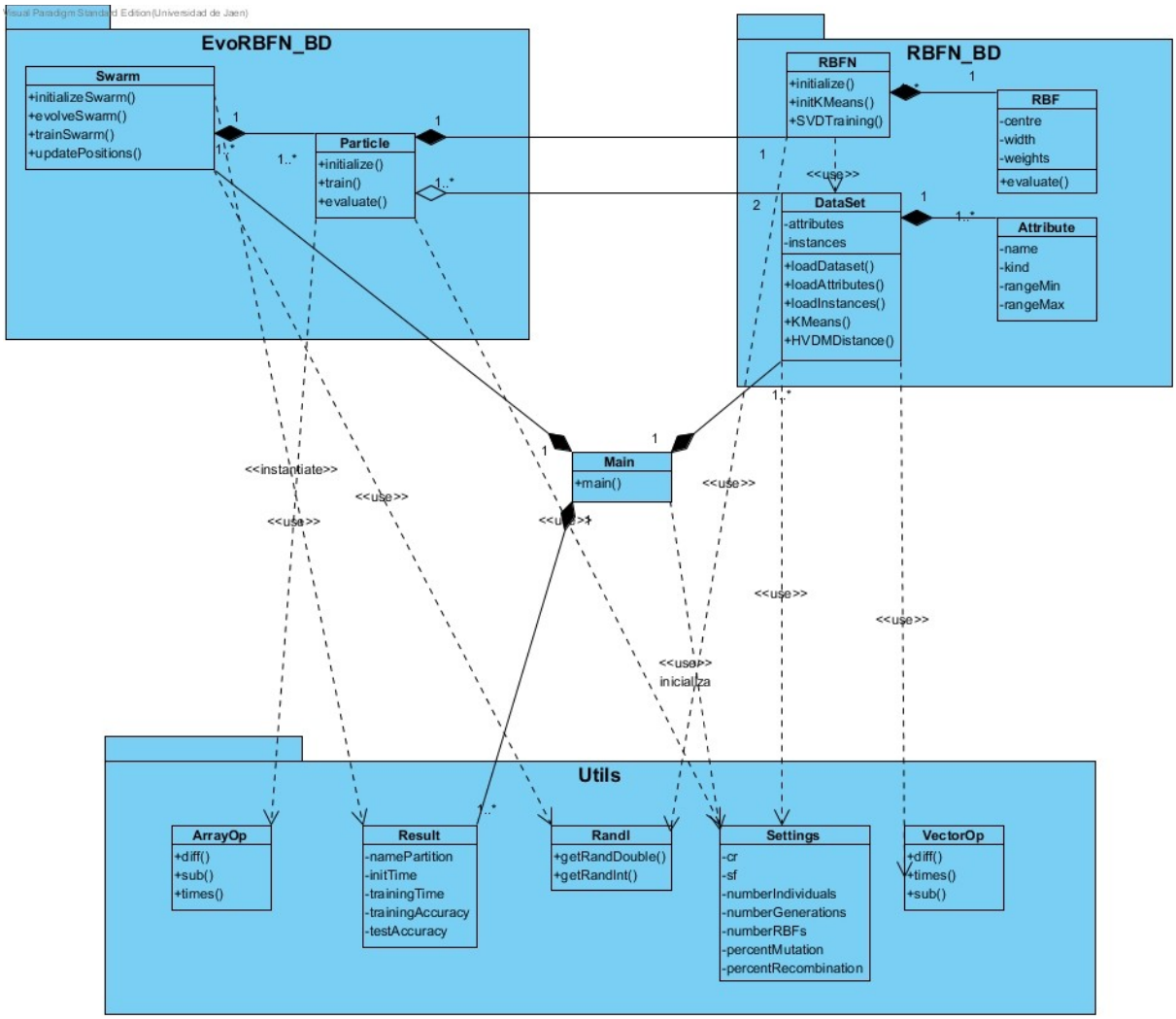



**Ilustración 3.3: Diagrama de clases ( Optimización mediante enjambres de partículas)**

### 3.3.2.2 Diseño de la secuencia de actividades del sistema

Una vez definidas las clases que componen el sistema y sus relaciones, se van a mostrar las interacciones entre las mismas a lo largo de la ejecución. Se utilizarán los diagramas de secuencia definidos en UML a tal efecto.

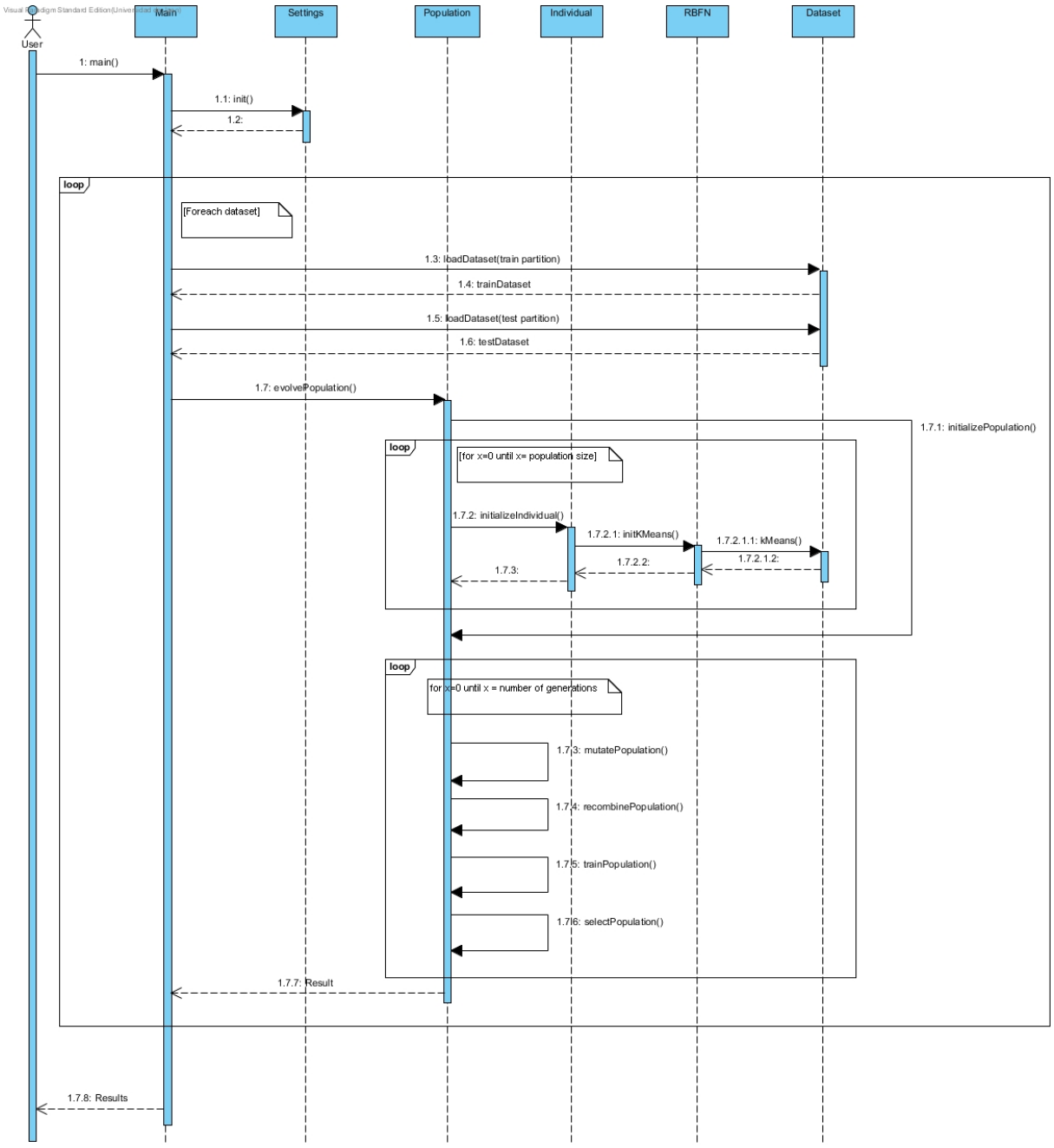



**Ilustración 3.4: Diagrama de secuencia ( Evolución Diferencial )**

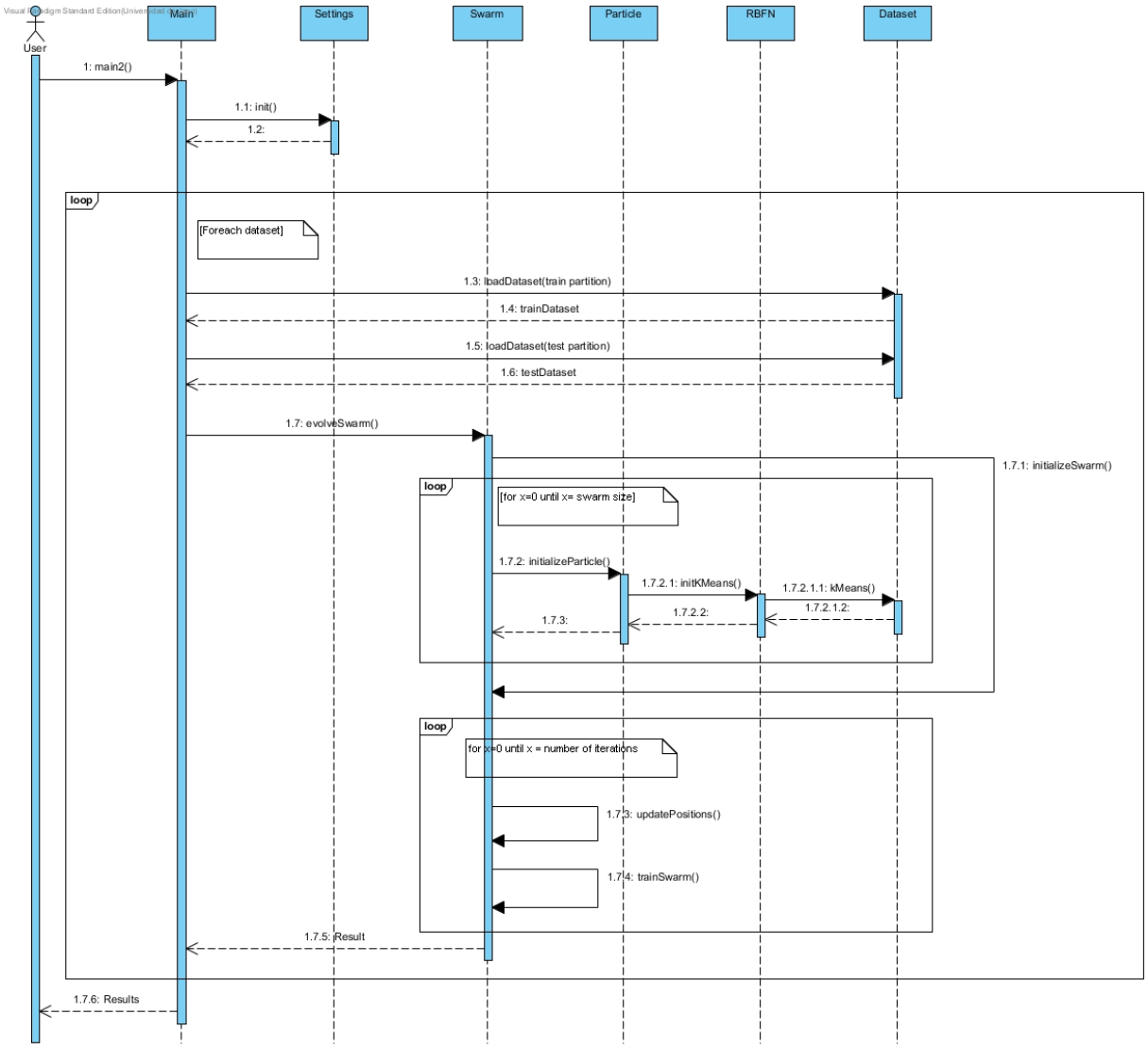



**Ilustración 3.5: Diagrama de secuencia ( optimización mediante enjambres de partículas )**

### 3.3.3 Implementación

En este apartado se pretende hacer una reseña de los componentes más importantes para llevar a cabo la implementación de los modelos descritos.

### 3.3.3.1 El lenguaje de programación *Scala*

El lenguaje de programación en el que se va a realizar la implementación es *Scala* [Ordersky y otros, 2010]. Este lenguaje suele utilizarse sobre una máquina virtual de Java. Esto le da un valor añadido, ya que, puede hacer uso de la biblioteca de funciones base de la misma o de cualquier otro código escrito en Java.


**Ilustración 3.6:**
**Logotipo de Scala**

Es un lenguaje de programación funcional y orientado a objetos. En *Scala* no existen tipos primitivos, cualquier variable es un objeto. Uno de los objetivos perseguidos al crear el lenguaje fue la escalabilidad(de ahí su nombre), éste pretende ser una herramienta que crezca con las demandas del usuario. Permite sobreescribir cualquier operador, de hecho, en el lenguaje no hay operadores propiamente dichos; si no que son métodos de objetos.

El lenguaje pretende ser conciso y legible. Las líneas de código tienden a ser significativamente más cortas que en Java y con un menor número de las mismas, consiguiendo una mayor funcionalidad. El lenguaje omite ciertas partes que eran reduntantes en su "predecesor" sobre todo en la declaración de variables.

### 3.3.3.2 El framework Spark

Una de las metas de este trabajo es el de adaptar modelos de aprendizaje automático para que puedan ejecutarse de forma paralela en un conjunto de máquinas: *cluster*. Apache Spark es un framework para computación distribuida diseñado para ser rápido y de propósito general.[Karau H, 2015]

Spark utiliza el paradigma *MapReduce[Dean y Ghemawat, 2004]* para distribuir los cálculos sobre el conjunto de máquinas. Una de sus características más

interesantes, ya que esto hace que sea un framework que se ejecute deprisa, es que permite la carga y procesamiento en RAM de los conjuntos de datos.

El framework ofrece una API para conectarlo con Python, Java y Scala. Así mismo, puede ser integrado con otras herramientas como Hadoop.

#### 3.3.3.3 El kit de desarrollo integrado IntelliJ IDEA

IntelliJ es un entorno de desarrollo integrado escrito en Java por la compañía JetBrains. Es el que se ha utilizado para el desarrollo y testeo de los algoritmos de minería de datos en este trabajo. Posee dos versiones: una es gratuita y la otra de pago. Ambas pueden ser utilizadas con fines comerciales pero para el desarrollo de este trabajo se ha utilizado la versión gratuita.

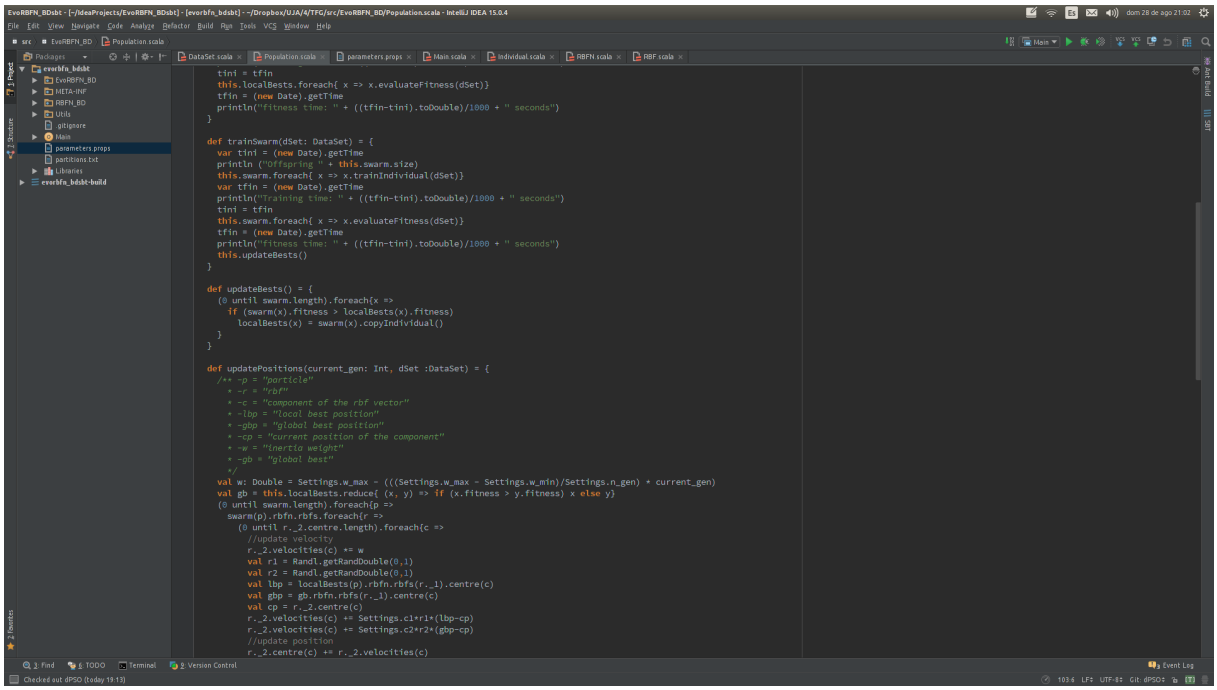


**Ilustración 3.7: Entorno de desarrollo integrado IntelliJ**

# 4. RESULTADOS Y DISCUSIÓN

El objetivo de este capítulo es el de describir la configuración de la experimentación, mostrar los resultados obtenidos a partir de la misma y hacer un breve análisis de los mismos.

## 4.1 Experimentación

Una vez finalizada la implementación de los modelos, se va a realizar una experimentación con los mismos para evaluar sus prestaciones. Para llevar a cabo la tarea, se ejecutarán los algoritmos sobre distintos conjuntos de datos.

### 4.1.1 Conjuntos de datos

Se van a utilizar cinco conjuntos de datos de gran tamaño, todos ellos disponibles en la plataforma Keel: shuttle, letters, connect-4, penbased y magic. Las propiedades de los mismos están recogidas en la siguiente tabla.

| Dataset | nº de Instancias | Atributos R/I/N | nº clases |
|---|---|---|---|
| shuttle | 58000 | (0 / 9 / 0) | 7 |
| letter | 20000 | (0 / 16 / 0) | 26 |
| connect-4 | 67557 | (0 / 0 / 42) | 3 |
| penbased | 10992 | (0 / 16 / 0) | 10 |
| magic | 19020 | (10 / 0 / 0) | 2 |

**Tabla 4.1: Características de los conjuntos de datos**

### 4.1.2 Características de la experimentación

Para realizar la experimentación se utilizará un cluster propiedad de la Universidad de Jaén. El cluster está compuesto por 14 nodos; cada uno de ellos equipado con 2 procesadores Intel Xeon E5-2670 v2 y 64 GB de RAM.

La experimentación se realizará utilizando la técnica de validación cruzada con 5 particiones. Los conjuntos de datos se descargan ya preparados para su ejecución desde la página web de Keel.

Para cada uno de los conjuntos de datos se llevarán a cabo 7 ejecuciones; dividiendo el conjunto de datos en mayor número de particiones a medida que se avanza en las ejecuciones. Más concretamente, los conjuntos de datos se dividirán en 1, 2, 4, 8, 16, 32 y 64 particiones internas. Si se ejecuta el experimento utilizando una sola partición interna se obtiene el mismo resultado que si se ejecutara de forma

local en una máquina con un solo núcleo de procesamiento. A medida que se van utilizando mayor número de particiones, el procesamiento de los datos se va dividiendo en mas unidades de procesamiento; tantas unidades como particiones. El objetivo de ir aumentando paulatinamente el número de particiones internas es el de ver la evolución de los tiempos de ejecución del algoritmo.

A continuación, se presentan los parámetros que se han utilizado para los experimentos realizados.

| Parámetro | Valor |
|---|---|
| Número de generaciones | 50 |
| Tamaño de la población | 30 |
| Número máximo de RBFs | dos veces el número de clases |
| Número mínimo de RBFs | número de clases |
| Porcentaje de mutación | 10% |
| Porcentaje de recombinación | 50% |
| Porcentaje de actualización en centros y radios | 20% |

**Tabla 4.2: Parámetros del Algoritmo Genético**

| Parámetro | Valor |
|---|---|
| Número de generaciones | 50 |
| Tamaño de la población | 30 |
| Número de RBFs | dos veces el número de clases |
| Factor de Escala - β | 0,5 |
| CR | 1 |
| Porcentaje de recombinación | 50% |

**Tabla 4.3: Parámetros de Evolución Diferencial**

| Parámetro | Valor |
|---|---|
| Número de generaciones | 50 |
| Tamaño de la población | 8 |
| Número de RBFs | dos veces el número de clases |
| Peso inicial de la inercia | 0,7 |
| Peso final de la inercia | 0,1 |
| c1 | 0,1 |
| c2 | 0,15 |

**Tabla 4.4: Parámetros de PSO**

Puede resultar llamativo el tamaño de la población del último método; es 8 frente a 30 de los otros dos casos. Es un cuarto de los otros dos métodos aproximadamente. En el método de optimización mediante enjambres de partículas,

toda la población actualiza su posición y ha de ser entrenada en cada iteración. En los otros dos casos, en cada generación, se genera un número aleatorio de hijos; este número se escoge de una distribución uniforme entre el 0% y el 50% del tamaño de la población; lo que da una media del 25% de la población. Será este el número de individuos a los que, en cada iteración, se les aplicarán los operadores genéticos y serán entrenados. La razón de ese valor es pues, que los tres métodos se ejecuten en similares condiciones y que los tiempos obtenidos por los tres sean también parecidos.

Puede sorprender también el valor del parámetro *CR* en el método de evolución diferencial. Ese valor implica que el proceso de recombinación no se ejecutará; o lo que es lo mismo, siguiendo la notación anterior: que $v^i_{hijo}$ será siempre igual a $v^i_{mut}$. Tras algunas pruebas se han obtenido buenos resultados con ese valor, este se recomienda en el trabajo de Truong[Truong,2012].

### 4.1.3 Métodos con los que se compara

En la actualidad existe un único trabajo conocido que aborda el problema de la optimización de redes neuronales de base radial utilizando técnicas de procesamiento distribuido: *GenRBFNSpark.*[Rivera, Pérez Godoy y otros; 2015]

Este trabajo utiliza un algoritmo genético, descrito en el capítulo 2 de este proyecto, para optimizar los parámetros que conforman la red neuronal.

## 4.2 Resultados

A continuación se muestran los resultados obtenidos tras realizar las ejecuciones de los algoritmos en el cluster. La propiedad principal que se pretende analizar son los tiempos de ejecución. Se desea comprobar que éstos se acortan a medida que el procesamiento de los datos se va distribuyendo en más núcleos. No obstante, también se prestará atención a la precisión de los modelos y la estabilidad de los resultados obtenidos.

Esto último es importante en los métodos que se van a analizar, ya que estos son estocásticos. Al incluir una componente aleatoria, la precisión obtenida en una

ejecución puede no ser la misma en la siguiente. Interesa que los resultados sean parecidos en distintas ejecuciones sobre los mismos datos para comprobar que el método es fiable. También interesa que sean parecidos entre las distintas particiones que se realizarán mediante la técnica de validación cruzada.

Esta sección va a dividirse en cuatro apartados. Los tres primeros se van a dedicar a detallar los resultados de cada uno de los algoritmos por separado. En el último se realizará un análisis global.

### 4.2.1 Evolución Diferencial

A continuación se muestran los resultados obtenidos por el modelo basado en Evolución Diferencial en el cluster. Se dedica una tabla a cada uno de los conjuntos de datos. Los datos recogidos en cada tabla son los siguientes:

1. Particiones: Número de particiones internas del conjunto de datos. Para distribuir el trabajo sobre las distintas unidades de procesamiento del cluster.

2. Precisión(Training): Precisión media obtenida por las 5 ejecuciones (recordar que la experimentación se realiza con validación cruzada de 5 particiones) utilizando el conjunto de datos de entrenamiento.

3. σ(Training): Desviación estándar de los resultados definidos en el punto anterior.

4. Precisión(Test): Precisión media obtenida por las 5 ejecuciones utilizando el conjunto de test.

5. σ(Test): Desviación estándar de los resultados definidos en el punto anterior.

6. Tiempo(segundos): Tiempo en segundos que ha tardado en ejecutarse el experimento.

| Particiones | Precisión (Training) | σ (Training) | Precisión (Test) | σ (Test) | Tiempo (segundos) |
|---|---|---|---|---|---|
| 1 | 88,47% | 0,12% | 88,45% | 0,29% | 13754 |
| 2 | 88,53% | 0,07% | 88,53% | 0,33% | 7525 |
| 4 | 88,55% | 0,10% | 88,55% | 0,32% | 4537 |
| 8 | 88,46% | 0,04% | 88,45% | 0,36% | 2982 |
| 16 | 88,54% | 0,11% | 88,55% | 0,32% | 2556 |
| 32 | 88,48% | 0,11% | 88,45% | 0,29% | 2928 |
| 64 | 88,48% | 0,12% | 88,47% | 0,33% | 3179 |

**Tabla 4.5: Resultados para el dataset shuttle.**

| Particiones | Precisión (Training) | σ (Training) | Precisión (Test) | σ (Test) | Tiempo (segundos) |
|---|---|---|---|---|---|
| 1 | 72,22% | 0,30% | 70,10% | 1,71% | 26287 |
| 2 | 71,83% | 0,33% | 70,40% | 2,30% | 14570 |
| 4 | 71,87% | 0,28% | 67,47% | 4,40% | 8074 |
| 8 | 71,84% | 0,37% | 70,04% | 2,62% | 5227 |
| 16 | 71,77% | 0,45% | 70,69% | 1,39% | 3504 |
| 32 | 72,12% | 0,26% | 69,59% | 2,48% | 2832 |
| 64 | 72,11% | 0,21% | 68,35% | 2,90% | 2528 |

**Tabla 4.6: Resultados para el dataset connect4.**

| Particiones | Precisión (Training) | σ (Training) | Precisión (Test) | σ (Test) | Tiempo (segundos) |
|---|---|---|---|---|---|
| 1 | 70,87% | 0,65% | 70,05% | 0,81% | 34077 |
| 2 | 70,83% | 0,25% | 70,13% | 0,66% | 17413 |
| 4 | 71,00% | 0,44% | 70,36% | 0,41% | 10189 |
| 8 | 71,13% | 0,47% | 70,40% | 0,31% | 6496 |
| 16 | 71,26% | 0,36% | 70,40% | 0,53% | 4931 |
| 32 | 70,81% | 0,48% | 69,95% | 0,51% | 4069 |
| 64 | 70,89% | 0,38% | 69,94% | 0,31% | 7971 |

**Tabla 4.7: Resultados para el dataset letter.**

| Particiones | Precisión (Training) | σ (Training) | Precisión (Test) | σ (Test) | Tiempo (segundos) |
|---|---|---|---|---|---|
| 1 | 90,35% | 0,07% | 90,24% | 0,27% | 5579 |
| 2 | 90,24% | 0,30% | 89,94% | 0,44% | 3405 |
| 4 | 90,30% | 0,43% | 89,82% | 0,51% | 2113 |
| 8 | 90,39% | 0,62% | 89,99% | 0,53% | 1579 |
| 16 | 90,20% | 0,17% | 90,08% | 0,39% | 1458 |
| 32 | 90,48% | 0,39% | 90,37% | 0,33% | 1370 |
| 64 | 90,45% | 0,36% | 90,23% | 0,90% | 2172 |

**Tabla 4.8: Resultados para el dataset penbased.**

| Particiones | Precisión (Training) | σ (Training) | Precisión (Test) | σ (Test) | Tiempo (segundos) |
|---|---|---|---|---|---|
| 1 | 74,83% | 0,68% | 74,47% | 0,82% | 3732 |
| 2 | 74,81% | 0,70% | 74,64% | 0,54% | 2136 |
| 4 | 75,29% | 0,35% | 75,07% | 0,65% | 1476 |
| 8 | 74,28% | 0,63% | 74,21% | 1,07% | 1188 |
| 16 | 74,96% | 1,01% | 75,01% | 1,15% | 1034 |
| 32 | 74,78% | 0,60% | 74,62% | 0,87% | 1042 |
| 64 | 74,76% | 0,59% | 74,62% | 0,59% | 1751 |

**Tabla 4.9: Resultados para el dataset magic.**

Al igual que en el caso anterior, los resultados muestran que el algoritmo es bastante robusto. Hay que recordar que los métodos estudiados en este trabajo son estocásticos y, por tanto, cada ejecución es distinta. Aun así se observa que, independientemente del número de particiones utilizadas, la precisión obtenida es siempre muy similar. Las desviaciones típicas obtenidas en todos los casos son muy bajas, esto es, se obtienen resultados muy parecidos en distintas ejecuciones del método; esto nos da la idea de que el método es fiable. Por último, comentar que los resultados obtenidos con el conjunto de datos de entrenamiento y test son muy parecidos. En general, siempre han de obtenerse mejores precisiones con el conjunto de entrenamiento; pero en este caso la diferencia entre ambos es muy pequeña. El método, por tanto, no sufre de problemas de sobre-entrenamiento.

### 4.2.2 Optimización mediante Enjambres de Partículas

A continuación se muestran los resultados obtenidos por el modelo basado en Enjambres de Partículas en el cluster. Se dedica una tabla a cada uno de los conjuntos de datos. Los datos recogidos en cada tabla son los siguientes:

1. Particiones: Número de particiones internas del conjunto de datos. Para distribuir el trabajo sobre las distintas unidades de procesamiento del cluster.
2. Precisión(Training): Precisión media obtenida por las 5 ejecuciones (recordar que la experimentación se realiza con validación cruzada de 5 particiones) utilizando el conjunto de datos de entrenamiento.
3. σ(Training): Desviación estándar de los resultados definidos en el punto anterior.
4. Precisión(Test): Precisión media obtenida por las 5 ejecuciones utilizando el conjunto de test.
5. σ(Test): Desviación estándar de los resultados definidos en el punto anterior.
6. Tiempo(segundos): Tiempo en segundos que ha tardado en ejecutarse el experimento.

| Particiones | Precisión (Training) | σ (Training) | Precisión (Test) | σ (Test) | Tiempo (segundos) |
|---|---|---|---|---|---|
| 1 | 88,47% | 0,08% | 88,46% | 0,35% | 11492 |
| 2 | 88,44% | 0,08% | 88,43% | 0,35% | 6415 |
| 4 | 88,44% | 0,14% | 88,42% | 0,27% | 3827 |
| 8 | 88,38% | 0,07% | 88,37% | 0,35% | 2560 |
| 16 | 88,40% | 0,09% | 88,35% | 0,39% | 2257 |
| 32 | 88,51% | 0,07% | 88,49% | 0,33% | 2460 |
| 64 | 88,43% | 0,14% | 88,42% | 0,29% | 3130 |

**Tabla 4.10: Resultados para el dataset shuttle.**

| Particiones | Precisión (Training) | σ (Training) | Precisión (Test) | σ (Test) | Tiempo (segundos) |
|---|---|---|---|---|---|
| 1 | 71,10% | 0,31% | 69,08% | 1,71% | 27584 |
| 2 | 70,61% | 0,39% | 67,54% | 2,84% | 14769 |
| 4 | 71,03% | 0,45% | 69,18% | 1,39% | 8167 |
| 8 | 71,33% | 0,47% | 71,05% | 0,92% | 5079 |
| 16 | 71,03% | 0,63% | 68,88% | 0,77% | 3489 |
| 32 | 70,93% | 0,41% | 68,62% | 1,46% | 2802 |
| 64 | 70,81% | 0,32% | 69,18% | 1,72% | 2339 |

**Tabla 4.11: Resultados para el dataset connect4.**

| Particiones | Precisión (Training) | σ (Training) | Precisión (Test) | σ (Test) | Tiempo (segundos) |
|---|---|---|---|---|---|
| 1 | 69,93% | 0,16% | 69,35% | 0,65% | 29799 |
| 2 | 70,52% | 0,41% | 69,46% | 0,85% | 15483 |
| 4 | 70,38% | 0,47% | 69,64% | 0,60% | 8982 |
| 8 | 70,35% | 0,37% | 69,45% | 0,85% | 5763 |
| 16 | 70,32% | 0,45% | 69,63% | 0,91% | 4305 |
| 32 | 69,87% | 0,42% | 69,56% | 0,33% | 3767 |
| 64 | 70,10% | 0,51% | 69,49% | 0,95% | 4649 |

**Tabla 4.12: Resultados para el dataset letter.**

| Particiones | Precisión (Training) | σ (Training) | Precisión (Test) | σ (Test) | Tiempo (segundos) |
|---|---|---|---|---|---|
| 1 | 90,17% | 0,22% | 90,02% | 0,40% | 5113 |
| 2 | 89,98% | 0,50% | 89,67% | 0,42% | 2872 |
| 4 | 90,05% | 0,48% | 89,96% | 0,67% | 1814 |
| 8 | 90,15% | 0,32% | 89,98% | 0,46% | 1293 |
| 16 | 89,88% | 0,39% | 89,87% | 0,71% | 1277 |
| 32 | 90,40% | 0,57% | 90,01% | 0,85% | 1264 |
| 64 | 89,81% | 0,60% | 89,45% | 0,50% | 2477 |

**Tabla 4.13: Resultados para el dataset penbased.**

| Particiones | Precisión (Training) | σ (Training) | Precisión (Test) | σ (Test) | Tiempo (segundos) |
|---|---|---|---|---|---|
| 1 | 73,89% | 0,79% | 73,76% | 0,38% | 2885 |
| 2 | 72,88% | 0,66% | 72,98% | 0,92% | 1740 |
| 4 | 73,32% | 0,73% | 73,29% | 0,78% | 1135 |
| 8 | 73,17% | 0,79% | 73,10% | 1,22% | 872 |
| 16 | 73,54% | 0,58% | 73,62% | 0,65% | 752 |
| 32 | 73,18% | 0,63% | 73,08% | 0,57% | 737 |
| 64 | 72,89% | 0,53% | 72,73% | 1,03% | 1164 |

**Tabla 4.14: Resultados para el dataset magic.**

Como en el caso anterior, los resultados muestran que el algoritmo es bastante robusto. Los métodos estudiados en este trabajo son estocásticos y, por tanto, no hay dos ejecuciones iguales. Aun así se observa que, con independencia del número de particiones utilizadas, la precisión obtenida es siempre muy parecida. Puede decirse que el método es fiable debido a las bajas desviaciones típicas obtenidas. Por último, comentar que los resultados obtenidos con el conjunto de datos de entrenamiento y test son muy parecidos. Esto induce a pensar que el método no sufre de sobre-entrenamiento.

### 4.2.3 Algoritmo Genético

A continuación se muestran los resultados obtenidos por el Algoritmo Genético en el cluster. Se dedica una tabla a cada uno de los conjuntos de datos. Los datos recogidos en cada tabla son los siguientes:

1. Particiones: Número de particiones internas del conjunto de datos. Para distribuir el trabajo sobre las distintas unidades de procesamiento del cluster.

2. Precisión(Training): Precisión media obtenida por las 5 ejecuciones (recordar que la experimentación se realiza con validación cruzada de 5 particiones) utilizando el conjunto de datos de entrenamiento.

3. σ(Training): Desviación estándar de los resultados definidos en el punto anterior.

4. Precisión(Test): Precisión media obtenida por las 5 ejecuciones utilizando el conjunto de test.

5. σ(Test): Desviación estándar de los resultados definidos en el punto anterior.

6. Tiempo(segundos): Tiempo en segundos que ha tardado en ejecutarse el experimento.

| Particiones | Precisión (Training) | σ (Training) | Precisión (Test) | σ (Test) | Tiempo (segundos) |
|---|---|---|---|---|---|
| 1 | 88,35% | 0,09% | 88,35% | 0,33% | 11233 |
| 2 | 88,35% | 0,10% | 88,32% | 0,32% | 6362 |
| 4 | 88,39% | 0,10% | 88,38% | 0,32% | 3764 |
| 8 | 88,35% | 0,15% | 88,35% | 0,28% | 2544 |
| 16 | 88,31% | 0,08% | 88,30% | 0,34% | 2344 |
| 32 | 88,35% | 0,07% | 88,36% | 0,32% | 2315 |
| 64 | 88,40% | 0,01% | 88,40% | 0,40% | 2866 |

**Tabla 4.15: Resultados para el dataset shuttle.**

| Particiones | Precisión (Training) | σ (Training) | Precisión (Test) | σ (Test) | Tiempo (segundos) |
|---|---|---|---|---|---|
| 1 | 70,19% | 0,33% | 68,17% | 1,06% | 21933 |
| 2 | 70,72% | 0,25% | 67,98% | 3,03% | 11490 |
| 4 | 70,43% | 0,31% | 66,68% | 4,68% | 6869 |
| 8 | 70,28% | 0,51% | 66,31% | 5,53% | 4307 |
| 16 | 70,29% | 0,22% | 68,87% | 1,08% | 3056 |
| 32 | 70,14% | 0,31% | 69,21% | 1,40% | 2558 |
| 64 | 70,65% | 0,37% | 68,36% | 1,63% | 2291 |

**Tabla 4.16: Resultados para el dataset connect4.**

| Particiones | Precisión (Training) | σ (Training) | Precisión (Test) | σ (Test) | Tiempo (segundos) |
|---|---|---|---|---|---|
| 1 | 67,43% | 0,66% | 66,90% | 1,37% | 27190 |
| 2 | 67,20% | 0,71% | 66,35% | 1,18% | 14865 |
| 4 | 67,31% | 0,77% | 66,58% | 0,68% | 8785 |
| 8 | 66,71% | 0,39% | 65,95% | 0,41% | 5660 |
| 16 | 67,16% | 0,62% | 66,72% | 0,77% | 4367 |
| 32 | 67,08% | 0,41% | 66,57% | 0,70% | 3632 |
| 64 | 67,73% | 0,42% | 67,06% | 0,38% | 8339 |

**Tabla 4.17: Resultados para el dataset letter.**

| Particiones | Precisión (Training) | σ (Training) | Precisión (Test) | σ (Test) | Tiempo (segundos) |
|---|---|---|---|---|---|
| 1 | 88,53% | 0,58% | 88,75% | 0,87% | 5021 |
| 2 | 88,98% | 0,43% | 89,04% | 0,60% | 2933 |
| 4 | 89,06% | 0,25% | 89,02% | 0,65% | 1839 |
| 8 | 88,77% | 0,78% | 88,91% | 0,62% | 1327 |
| 16 | 88,44% | 0,62% | 88,39% | 0,68% | 1311 |
| 32 | 88,67% | 0,18% | 88,51% | 0,63% | 1294 |
| 64 | 88,95% | 0,53% | 88,76% | 0,78% | 2508 |

**Tabla 4.18: Resultados para el dataset penbased.**

| Particiones | Precisión (Training) | σ (Training) | Precisión (Test) | σ (Test) | Tiempo (segundos) |
|---|---|---|---|---|---|
| 1 | 73,71% | 1,06% | 73,82% | 0,91% | 3081 |
| 2 | 73,27% | 0,70% | 73,31% | 0,61% | 1893 |
| 4 | 73,18% | 0,81% | 73,29% | 0,83% | 1258 |
| 8 | 73,47% | 1,27% | 73,60% | 1,00% | 962 |
| 16 | 73,00% | 0,32% | 73,12% | 0,65% | 858 |
| 32 | 73,18% | 0,48% | 72,98% | 0,32% | 747 |
| 64 | 73,91% | 1,34% | 73,92% | 1,61% | 1375 |

**Tabla 4.19: Resultados para el dataset magic.**

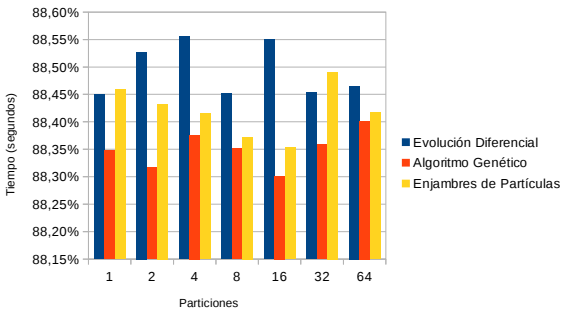
Los resultados muestran, al igual que en los otros dos casos, que se trata de un algoritmo robusto y sin problemas de sobre-entrenamiento.

### 4.2.4 Análisis Global

En este apartado se pretende realizar un análisis global de los resultados obtenidos en la experimentación. Se van a dividir en dos puntos: uno atendiendo a la precisión de los métodos y otro a su tiempo de ejecución.

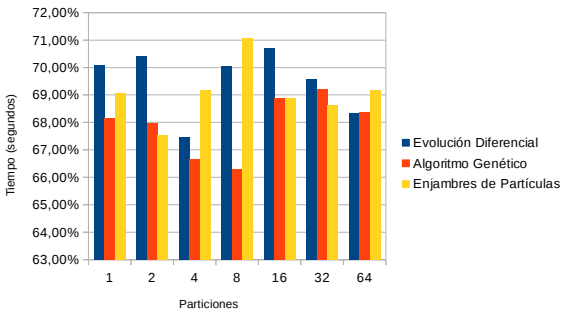
#### 4.2.4.1 Análisis de la precisión

Las gráficas que se presentan a continuación comparan la precisión de cada uno de los métodos en base al número de particiones utilizadas. Se utiliza una gráfica para cada uno de los conjuntos de datos. La medida de precisión mostrada es la media del conjunto de test, recordar que se ha utilizado validación cruzada con 5 particiones. Las tablas al lado de cada gráfica presentan los datos de precisión de forma numérica.



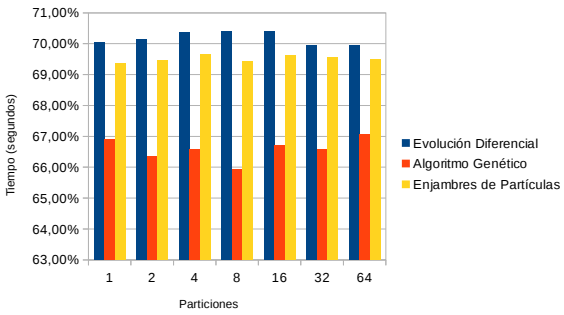

| Particiones | E.D. | P.S.O. | Genético |
|---|---|---|---|
| 1 | 88,45% | 88,46% | 88,35% |
| 2 | 88,53% | 88,43% | 88,32% |
| 4 | 88,55% | 88,42% | 88,38% |
| 8 | 88,45% | 88,37% | 88,35% |
| 16 | 88,55% | 88,35% | 88,30% |
| 32 | 88,45% | 88,49% | 88,36% |
| 64 | 88,47% | 88,42% | 88,40% |

**Tabla 4.20: Precisión obtenida en el dataset "Shuttle"**



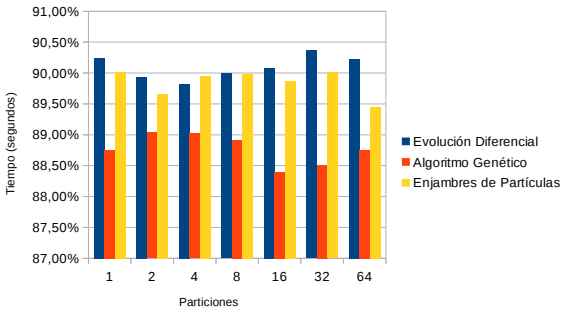

| Particiones | E.D. | P.S.O. | Genético |
|---|---|---|---|
| 1 | 70,10% | 69,08% | 68,17% |
| 2 | 70,40% | 67,54% | 67,98% |
| 4 | 67,47% | 69,18% | 66,68% |
| 8 | 70,04% | 71,05% | 66,31% |
| 16 | 70,69% | 68,88% | 68,87% |
| 32 | 69,59% | 68,62% | 69,21% |
| 64 | 68,35% | 69,18% | 68,36% |

**Tabla 4.21: Precisión obtenida en el dataset "Connect 4"**



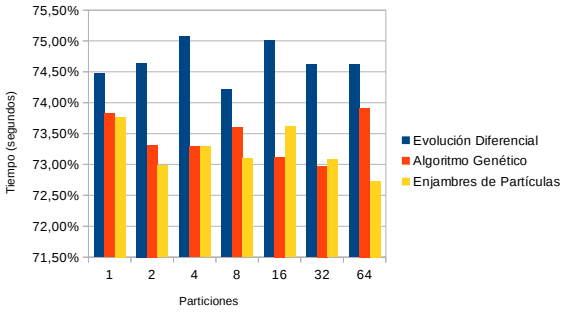

| Particiones | E.D. | P.S.O. | Genético |
|---|---|---|---|
| 1 | 70,05% | 69,35% | 66,90% |
| 2 | 70,13% | 69,46% | 66,35% |
| 4 | 70,36% | 69,64% | 66,58% |
| 8 | 70,40% | 69,45% | 65,95% |
| 16 | 70,40% | 69,63% | 66,72% |
| 32 | 69,95% | 69,56% | 66,57% |
| 64 | 69,94% | 69,49% | 67,06% |

**Tabla 4.22: Precisión obtenida en el dataset "Letters"**




| Particiones | E.D. | P.S.O. | Genético |
|---|---|---|---|
| 1 | 90,24% | 90,02% | 88,75% |
| 2 | 89,94% | 89,67% | 89,04% |
| 4 | 89,82% | 89,96% | 89,02% |
| 8 | 89,99% | 89,98% | 88,91% |
| 16 | 90,08% | 89,87% | 88,39% |
| 32 | 90,37% | 90,01% | 88,51% |
| 64 | 90,23% | 89,45% | 88,76% |

**Tabla 4.23: Precisión obtenida en el dataset "Penbased"**




| Particiones | E.D. | P.S.O. | Genético |
|---|---|---|---|
| 1 | 74,47% | 73,76% | 73,82% |
| 2 | 74,64% | 72,98% | 73,31% |
| 4 | 75,07% | 73,29% | 73,29% |
| 8 | 74,21% | 73,10% | 73,60% |
| 16 | 75,01% | 73,62% | 73,12% |
| 32 | 74,62% | 73,08% | 72,98% |
| 64 | 74,62% | 72,73% | 73,92% |

**Tabla 4.24: Precisión obtenida en el dataset "Magic"**

Según se puede observar, los resultados obtenidos por todos los métodos son bastante fiables; estos varían muy levemente entre distintas ejecuciones. También puede observarse que, de forma general, el enfoque basado en Evolución

Diferencial obtiene unos resultados levemente superiores a los otros dos métodos. Cabe recordar del análisis del apartado anterior, que este método también toma un poco más de tiempo en ejecutarse que los otros dos.

Para hacer una comparación formal de los resultados se van a realizar dos tests estadísticos: el test de Friedman y el test de Wilcoxon. Se trata de pruebas no paramétricas cuya hipótesis nula es la no existencia de diferencias significativas entre los métodos.

#### *4.2.4.1.1 Test de Friedman*

En un primer paso, esta prueba da como resultado un ranking de los métodos (tabla 4.25). En éste, un método es considerado mejor cuanto menor sea su valor de ranking.

| Algoritmo | Ranking |
|---|---|
| Evolución Diferencial | 1 |
| Optimización mediante enjambres de partículas | 2,2 |
| Algoritmo Genético | 2,8 |

**Tabla 4.25: Ranking del test de Friedman para la precisión obtenida en los modelos**

Observando los resultados, se extrae que el método que mejor se comporta es el de evolución diferencial; seguido por optimización mediante enjambres de partículas y finalmente el algoritmo genético.

En el siguiente y último paso, se compara el mejor de los métodos con el resto para determinar si existen diferencias estadísticamente significativas con él. La forma de hacerlo es mediante un contraste de hipótesis donde la hipótesis nula supone que entre los dos métodos que se comparan no existen diferencias. En la tabla 4.26 se muestran los p-valores obtenidos; el primero es el resultado para la comparación entre evolución diferencial y el algoritmo genético, y el segundo para evolución diferencial y PSO. De forma general, se acepta que existen diferencias significativas entre dos métodos si el p-valor es menor o igual que 0.05.

| Algoritmo | p-valor |
|---|---|
| Algoritmo Genético | 0,004427 |
| Optimización mediante enjambres de partículas | 0,05778 |

**Tabla 4.26: p-valores en el test de Friedman para la precisión de los modelos**

Teniendo en cuenta lo anterior podemos extraer dos conclusiones:

- Existen diferencias estadísticamente significativas entre el método de evolución diferencial y el método que utiliza el algoritmo genético.
- No existen diferencias estadísticamente significativas entre el método de evolución diferencial y el método basado en enjambres de partículas con un nivel de significación del 95%. Aunque el resultado está muy cerca del límite.

#### *4.2.4.1.2 Test de Wilcoxon*

En este caso se realiza un contraste de hipótesis de cada método con cada uno de los otros. Por ejemplo, si se tienen tres métodos: A, B y C; se contrastará A con B, A con C, B con A, B con C, C con A y C con B. El resultado de los contrastes no es simétrico, es decir, no es igual comparar A con B que B con A. La hipótesis nula en cada contraste es la misma que en el caso anterior, esto es, que no existen diferencias entre los dos métodos que se comparan. La hipótesis alternativa dice que el método que se compara se comporta mejor que el comparado. Por ejemplo, si se compara A con B, la hipótesis alternativa dice que A se comporta mejor que B.

| Algoritmo | p-valor |
|---|---|
| Optimización mediante enjambres de partículas | 0,030971 |
| Algoritmo Genético | 0,030971 |

**Tabla 4.27: Resultados para Evolución Diferencial**

La tabla 4.27 muestra los p-valores obtenidos para los contrastes de hipótesis de la evolución diferencial. Se extraen de los mismos las siguientes conclusiones:

- El método de evolución diferencial se comporta mejor que la optimización mediante enjambres de partículas al 95% de significación.

- El método de evolución diferencial se comporta mejor que el algoritmo genético al 95% de significación.

| Algoritmo | p-valor |
|---|---|
| Evolución Diferencial | 1 |
| Algoritmo Genético | 0,105645 |

**Tabla 4.28: Resultados para PSO**

La tabla 4.28 muestra los p-valores para optimización con enjambres de partículas.

- No existen diferencias significativas entre la optimización mediante enjambres de partículas y los otros dos métodos.

| Algoritmo | p-valor |
|---|---|
| Evolución Diferencial | 1 |
| Optimización mediante enjambres de partículas | 1 |

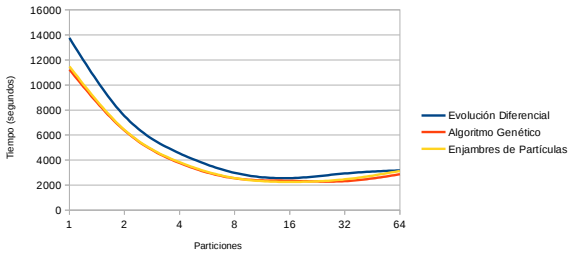
**Tabla 4.29: Resultados para el algoritmo genético**

La tabla 4.29 muestra los p-valores para el algoritmo genético. Por tanto:

- No existen diferencias significativas entre el algoritmo genético y los otros dos métodos.

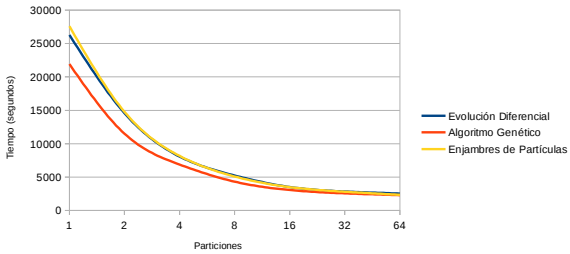
#### 4.2.4.2 Análisis de los tiempos

Las gráficas a que se muestran a continuación presentan una comparativa de los tiempos de ejecución de cada método según el número de particiones utilizadas. Cada una de ellas está asociada a un conjunto de datos. Las tablas al lado de cada gráfica presentan los tiempos (en segundos) de forma numérica.



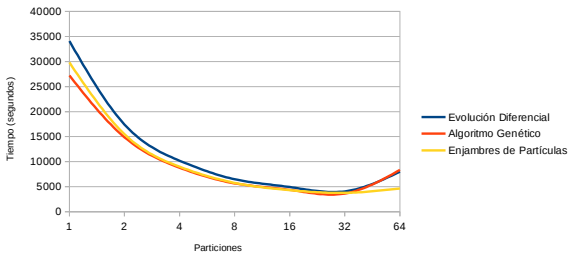

| Particiones | E.D. | P.S.O. | Genético |
|---|---|---|---|
| 1 | 13754 | 11492 | 11233 |
| 2 | 7525 | 6415 | 6362 |
| 4 | 4537 | 3827 | 3764 |
| 8 | 2982 | 2560 | 2544 |
| 16 | 2556 | 2257 | 2344 |
| 32 | 2928 | 2460 | 2315 |
| 64 | 3179 | 3130 | 2866 |

**Tabla 4.30: Tiempos obtenidos en el dataset "Shuttle"**



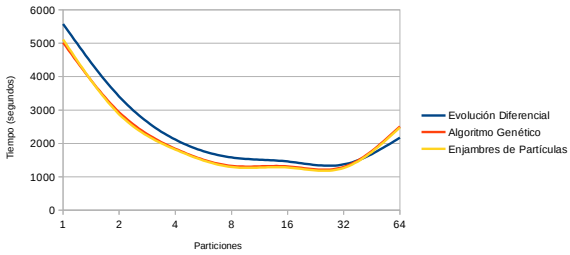

| Particiones | E.D. | P.S.O. | Genético |
|---|---|---|---|
| 1 | 26287 | 27584 | 21933 |
| 2 | 14570 | 14769 | 11490 |
| 4 | 8074 | 8167 | 6869 |
| 8 | 5227 | 5079 | 4307 |
| 16 | 3504 | 3489 | 3056 |
| 32 | 2832 | 2802 | 2558 |
| 64 | 2528 | 2339 | 2291 |

**Tabla 4.31: Tiempos obtenidos en el dataset "Connect 4"**



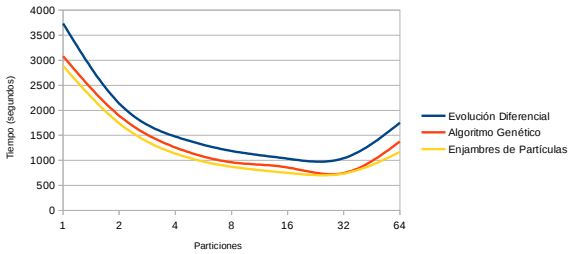

| Particiones | E.D. | P.S.O. | Genético |
|---|---|---|---|
| 1 | 34077 | 29799 | 27190 |
| 2 | 17413 | 15483 | 14865 |
| 4 | 10189 | 8982 | 8785 |
| 8 | 6496 | 5763 | 5660 |
| 16 | 4931 | 4305 | 4367 |
| 32 | 4069 | 3767 | 3632 |
| 64 | 7971 | 4649 | 8339 |

**Tabla 4.32: Tiempos obtenidos en el dataset "Letters"**




| Particiones | E.D. | P.S.O. | Genético |
|---|---|---|---|
| 1 | 5579 | 5113 | 5021 |
| 2 | 3405 | 2872 | 2933 |
| 4 | 2113 | 1814 | 1839 |
| 8 | 1579 | 1293 | 1327 |
| 16 | 1458 | 1277 | 1311 |
| 32 | 1370 | 1264 | 1294 |
| 64 | 2172 | 2477 | 2508 |

**Tabla 4.33: Tiempos obtenidos en el dataset "Penbased"**




| Particiones | E.D. | P.S.O. | Genético |
|---|---|---|---|
| 1 | 3732 | 2885 | 3081 |
| 2 | 2136 | 1740 | 1893 |
| 4 | 1476 | 1135 | 1258 |
| 8 | 1188 | 872 | 962 |
| 16 | 1034 | 752 | 858 |
| 32 | 1042 | 737 | 747 |
| 64 | 1751 | 1164 | 1375 |

**Tabla 4.34: Tiempos obtenidos en el dataset "Magic"**

Los resultados que muestran las tablas apoyan el comportamiento que esperaba obtenerse. En todos los casos, el tiempo de ejecución de un método disminuye a medida que el procesamiento va distribuyéndose en un mayor número de nodos. Puede observarse que llegado a un número de particiones, este tiempo de ejecución vuelve a aumentar. Una hipótesis que explicaría este fenómeno es que llega un momento en el que el trasiego de datos por la red penaliza más los tiempos de lo que los mejora el distribuir la ejecución. Un hecho que apoya esta hipótesis es el caso del dataset "connect4"; en este no se observa un aumento de los tiempos al particionar el conjunto de datos en ningún momento. Este es el conjunto de datos más grande de todos los utilizados, por tanto, es posible que utilizando conjuntos de datos más grandes, este hecho no se produzca.

Con respecto a las diferencias de tiempos de ejecución entre los distintos métodos, puede observarse que de forma general, el Algoritmo Genético y el enfoque basado en enjambres de partículas suelen tener tiempos de ejecución

bastante similares. El enfoque basado en evolución diferencial, por otra parte, suele tomar algo más de tiempo para terminar.

Para hacer una comparación formal de los resultados se van a realizar dos tests estadísticos: el test de Friedman y el test de Wilcoxon. Se trata de pruebas no paramétricas cuya hipótesis nula es la no existencia de diferencias significativas entre los métodos.

#### *4.2.4.2.1 Test de Friedman*

Del mismo modo que se mostró para el test de Friedman para la precisión; el ranking obtenido para los tiempos de ejecución se muestra en la tabla 4.35.

| Algoritmo | Ranking |
|---|---|
| Evolución Diferencial | 2,8 |
| Optimización mediante enjambres de partículas | 1,6 |
| Algoritmo Genético | 1,6 |

**Tabla 4.35: Ranking del test de Friedman para los tiempos de ejecución obtenidos en los modelos**

Por tanto, existe un empate entre el algoritmo genético y la optimización mediante enjambres de partículas en primera posición. En última posición queda el algoritmo de evolución diferencial.

En la tabla 4.36 se indican los p valores del algoritmo de optimización mediante enjambres de partículas con respecto a cada uno de los otros dos(del mismo modo podría haberse escogido el método genético ya que ambos tienen el mismo ranking).

| Algoritmo | p-valor |
|---|---|
| Evolución Diferencial | 0,05778 |
| Algoritmo Genético | 1 |

**Tabla 4.36: p-valores en el test de Friedman para los tiempos de ejecución de los modelos**

En general se acepta el rechazo de la hipótesis nula con un 95% de significación. Partiendo de esa premisa se extraen las siguientes conclusiones:

- No existen diferencias significativas entre el enfoque basado en enjambres de partículas y evolución diferencial. Aunque esta afirmación queda al límite, dado que el p-valor para este contraste es casi igual a 0,05.

- No existen diferencias significativas entre el enfoque basado en enjambres de partículas y el algoritmo genético.

#### *4.2.4.2.2 Test de Wilcoxon*

Tal y como se ha explicado en el test de Wilcoxon para la precisión, en este test se realiza un contraste de hipótesis de cada método con los otros dos. El resultado de los contrastes para evolución diferencial se muestra en la tabla 4.37.

| Algoritmo | p-valor |
|---|---|
| Optimización mediante enjambres de partículas | 1 |
| Algoritmo Genético | 1 |

**Tabla 4.37: Resultados para Evolución Diferencial**

Ambos p-valores son igual a 1. Esto indica que no hay diferencias significativas entre evolución diferencial y cualquiera de los otros dos métodos.

En la tabla 4.38 se muestran los resultados para el enfoque basado en enjambres de partículas.

| Algoritmo | p-valor |
|---|---|
| Evolución Diferencial | 0.059058 |
| Algoritmo Genético | 1 |

**Tabla 4.38: Resultados para PSO**

De los resultados se extraen las siguientes conclusiones:

- No existen diferencias significativas entre el algoritmo basado en enjambres de partículas y el método de evolución diferencial. Aunque esta afirmación queda justo al límite, ya que, el p-valor en este caso está muy cerca de 0,05.

- No existen diferencias significativas entre el método basado en enjambres de partículas y el algoritmo genético.

Los resultados para el algoritmo genético están plasmados en la tabla 4.39.

| Algoritmo | p-valor |
|---|---|
| Evolución Diferencial | 0.030971 |
| Optimización mediante enjambres de partículas | 0.787406 |

**Tabla 4.39: Resultados para el algoritmo genético**

Según los p-valores obtenidos, se observa que:

- El algoritmo genético se comporta mejor que evolución diferencial al 95% de significación.
- No existen diferencias significativas entre el algoritmo genético y el método basado en enjambres de partículas.

# 5. CONCLUSIONES

## 5.1 Conclusiones científicas

Tras la realización de la experimentación y el análisis de los resultados se pueden extraer dos conclusiones principales: la primera referente a los resultados del algoritmo de evolución diferencial y la segunda relacionada con la evolución de los tiempos de ejecución a medida que se distribuye la misma en un mayor número de unidades de procesamiento.

Como se observa en las tablas y gráficos presentados en el capítulo anterior, el algoritmo de evolución diferencial ha supuesto una mejora en prácticamente todos los casos probados con respecto al enfoque basado en enjambres de partículas y el algoritmo genético. Este hecho queda demostrado, además, tras realizar el test de Friedman y el de Wilcoxon sobre los resultados obtenidos. Por otra parte, de forma general, este algoritmo tarda un poco más de tiempo en ejecutarse que los otros dos bajo un conjunto de parámetros similares. Especialmente en el caso del algoritmo genético; el test de Wilcoxon lo declara como un mejor candidato en lo que se refiere a tiempos de ejecución con respecto a la evolución diferencial.

Por otra parte, se confirma el comportamiento principal que deseaba verse mediante la realización de este trabajo. A medida que el procesamiento de un conjunto de datos se va distribuyendo en más unidades de procesamiento, el tiempo que tarda en completarse disminuye. En 4 de los 5 conjuntos de datos ocurre un fenómeno respecto a esto, llega un momento en que un particionamiento excesivo de los datos penaliza los tiempos y estos vuelven a subir. Esto no ocurre en uno de los conjuntos de datos (connect4), el más grande de todos. Por tanto, puede que exista un número de particiones óptimo para cada caso que se trate; mayor cuanto más grande sea el conjunto de datos.

## 5.2 Conclusiones personales

Llegado a este punto, creo que se han cumplido los objetivos marcados para este trabajo de fin de grado.

Quiero aprovechar esta sección para agradecer especialmente su ayuda a Antonio Jesús Rivera Rivas, Francisco Javier Pulgar Rubio y a Loli Pérez Godoy, ya que sin ellos no podría haber realizado este proyecto. Antonio y Francisco, como aparece en la portada de este trabajo, son los tutores de este trabajo y Loli Pérez Godoy, aunque no aparece en la portada, ha estado siempre ahí también como si fuera otra tutora más.

Para la realización de este trabajo me han sido especialmente útiles los conocimientos adquiridos en tres asignaturas del grado: Metaheurísticas, Minería de Datos y Sistemas Inteligentes de Información. El desarrollo del mismo me ha permitido asentar un poco más los conocimientos adquiridas en las mismas. Aún así, llevarlo a cabo me ha resultado un reto, principalmente por la cantidad de nuevos conceptos que he tenido que asimilar.

También quiero remarcar una dificultad añadida a la propia de la implementación de modelos de aprendizaje; la de hacerlo de modo que puedan ejecutarse de forma distribuida. La experimentación con los métodos implementados, así como del algoritmo genético del que se parte, se ha realizado en un cluster propiedad de la Universidad de Jaén. La utilización del mismo es compartida por varios usuarios y es necesario ponerse de acuerdo con el resto para planificar las ejecuciones de cada uno. Este hecho, unido a que los métodos implementados necesitan de mucho tiempo para ejecutarse, añade dificultad al desarrollo.

Gracias a la realización de este trabajo he conocido el lenguaje de programación Scala. Una vez compilado, el código escrito en este lenguaje se ejecuta en una máquina virtual de Java; con la ventaja de poder aprovechar todo el software disponible para este, incluidas las librerías base. En años anteriores he tenido la oportunidad de escribir programas con Java, y en mi opinión, su predecesor (Scala) permite la escritura de un código más compacto y legible.

Mediante este lenguaje he hecho uso de la plataforma Spark. Me alegro mucho de haber tenido la suerte de utilizarlo, ya que, aunque toma un poco de tiempo el

aprender a utilizarlo, hace prácticamente transparente el hecho de distribuir el trabajo entre distintos nodos en una red.

Finalmente, quiero resaltar que he disfrutado la realización de este proyecto desde un punto de vista personal, pero además, creo que ha sido un acierto su ejecución desde un punto de vista profesional. En los tiempos actuales cada vez se demandan más profesionales con conocimientos en el ámbito de la ciencia de datos. Como ya se ha recalcado varias veces a lo largo del trabajo, vivimos en la era de los datos.

# 6. BIBLIOGRAFÍA

http://www.datacenterknowledge.com/archives/2013/01/18/facebook-builds-new-data-centers-for-cold-storage/, 2013

J. Dean and S. Ghemawat, Mapreduce: Simplified data processing on large clusters, 2008

Tom White, Hadoop, the definitive guide, 2014

https://es.wikipedia.org/wiki/Big_data,

María Dolores Pérez Godoy, Métodos híbridos evolutivos cooperativos-competitivos para el diseño de Redes De Funciones de Base Radial, 2010

César Ferri Ramírez, José Hernández Orallo y Mª José Ramírez Quintana , Introducción a la minería de datos, 2007

https://es.wikipedia.org/wiki/Algoritmo_evolutivo,

Michalewicz, Zbigniew, Genetic algorithms + data structures = evolution programs, 1999

Talbi, El-Ghazali, Metaheuristics: from design to implementation, 2009

Sung-Kwun Oh, Wook-Dong Kim, Witold Pedrycz y Su-Chong Joo, Design of K-means clustering-based polynomial radial basis function neuralnetworks (pRBF NNs) realized with the aid of particle swarm optimizationand differential evolution, 2012

Mahmood Rahmani, Particle swarm optimization of artificial neural networks for autonomous robots, 2008

Alex Alexandridis, Member, IEEE, Eva Chondodima and Haralambos Sarimveis, Radial Basis Function Network Training Using a Nonsymmetric Partition of the Input Space and Particle Swarm Optimization, 2013

https://en.wikipedia.org/wiki/Voronoi_diagram,

Cliff, L, Visualization of Matrix Singular Value Decomposition, 1983

Golub, G. y Van Loan, C., Matrix computations, 1996

Dean y Ghemawat, 2004: , MapReduce: Simplified Data Processing on Large Clusters, 2004

www.keel.es,

Jim Arlow y Ila Neustadt, UML 2, 2006

http://www.uml.org/,

Martin Odersky, Lex Spoon, Bill Venners, Programming in Scala, 2010

Holden Karau, Andy Konwinski,Patrick Wendell & Matei Zaharia, Learning Spark, 2015

Vu Truong Vu, A comparison of particle swarm optimization and differential evolution, 2012

Rivera, Pérez Godoy y otros; 2015: A. J. Rivera, M. D. Pérez-Godoy, F. Pulgar and M. J. del Jesus, GenRBFNSpark: A first implementation in Spark of a genetic algorithm to RBFN design, 2015

# ANEXO A: CÓDIGO FUENTE

Adjunto a este documento se encuentra un fichero llamado *src.zip* que contiene el código fuente de los dos métodos implementados para este trabajo. El código está contenido en un repositorio local *git*; este tiene dos ramas, una para cada uno de los métodos:

- dED: *fork* para el método de evolución diferencial.
- dPSO: *fork* para la técnica basada en optimización mediante enjambre de partículas.

# ANEXO B: MANUAL DE USUARIO

A partir del código fuente descrito en el anexo anterior pueden generarse los ejecutables para cada uno de los métodos: evolución diferencial y optimización mediante enjambres de partículas.

Ambos métodos aceptan los siguientes parámetros de línea de comandos:

1. Ruta al fichero de configuración (**obligatorio**). Es la ruta al fichero con los parámetros de configuración para el algoritmo. Su contenido será descrito posteriormente.

2. Ruta de las particiones (**obligatorio)**. Ruta al fichero de texto que a su vez contiene las rutas a los ficheros de los conjuntos de datos.

3. Ruta informe (**opcional**). Si se indica, un informe con los resultados obtenidos será escrito en dicha ruta.

4. Ruta informe csv (**opcional**). Si se indica, un informe reducido se escribirá en la ruta definida.

5. Número de particiones (**opcional**). Si se indica, el sistema dividirá el conjunto de datos en las particiones indicadas. Por defecto se utiliza una única partición.

El ejecutable construido tiene como objetivo el ser tratado con la herramienta de spark: *spark-submit*. Suponiendo que el nombre del ejecutable generado sea *RBJN.jar*, un ejemplo de ejecución del mismo podría ser el siguiente:

```
spark-submit --master localhost --name "ED app" ../RBFN.jar
../rbfn.props partitions.txt report.out results.csv 32
```

En los tres apartados que siguen se van a definir correspondientemente los formatos de los ficheros de particiones y de configuración para el método de evolución diferencial y de optimización mediante enjambres de partículas. En todos ellos, cualquier línea que comience con el carácter # se interpretará como un comentario y no se tendrá en cuenta.

## Fichero de particiones

Se tratará de un fichero de texto plano. En cada línea del mismo se indicará una de las particiones a tratar. Cada partición viene definida por dos ficheros; uno de training y otro de test. Estos vendrán definidos en cada línea en ese orden y separados por un espacio entre ellos.

Un ejemplo de contenido de un fichero de particiones es el siguiente:

```
letter-10-1tra.dat letter-10-1tst.dat
letter-10-2tra.dat letter-10-2tst.dat
letter-10-3tra.dat letter-10-3tst.dat
letter-10-4tra.dat letter-10-4tst.dat
letter-10-5tra.dat letter-10-5tst.dat
letter-10-6tra.dat letter-10-6tst.dat
letter-10-7tra.dat letter-10-7tst.dat
letter-10-8tra.dat letter-10-8tst.dat
letter-10-9tra.dat letter-10-9tst.dat
```

## Configuración: Evolución Diferencial

En este fichero se determina la configuración para la ejecución del algoritmo. Está formado por una lista de parámetros que no tienen porque seguir un orden estricto. Cada parámetro se indicará en una linea distinta. El formato del parámetro es el siguiente:

<nombre del parámetro>:<valor del parámetro>

Los parámetros que acepta este método son los siguientes:

- **k** (entero, obligatorio). Número de agrupamientos que utilizará el algoritmo de las k-medias.
- **cr** (real, obligatorio). Porcentaje de cruce en el intervalo [0 , 1].
- **sf** (real, obligatorio). Factor de escala.
- **nInd** (entero, obligatorio). Tamaño de la población.

- **n_gen** (entero, obligatorio). Número de generaciones.

- **iPart** (entero, opcional). Número de particiones internas, para distribución sobre el cluster. Si se pasa el valor por parámetro de línea de comandos tiene preferencia sobre este.

- **nRepetitions** (entero, obligatorio). Número de repeticiones del algoritmo que se realizarán sobre cada partición del dataset. Esto es, se ejecutará el medio esas veces sobre el conjunto de datos. Su sentido es que se está tratando con métodos estocásticos.

- **reportFile** (cadena, opcional). Ruta para guardar un informe de los resultados obtenidos. El pasado como parámetro de línea de comandos tiene preferencia.

- **csvFile** (cadena, opcional). Ruta para guardar un informe resumido de los resultados obtenidos en formato csv. El pasado como parámetro de la línea de comandos tiene preferencia sobre este.

## Configuración: optimización mediante Enjambres de Partículas

En este fichero se determina la configuración para la ejecución del algoritmo. Está formado por una lista de parámetros que no tienen porque seguir un orden estricto. Cada parámetro se indicará en una linea distinta. El formato del parámetro es el siguiente:

<nombre del parámetro>:<valor del parámetro>

Los parámetros que acepta este método son los siguientes:

- **k** (entero, obligatorio). Número de agrupamientos que utilizará el algoritmo de las k-medias.

- **s** (entero, obligatorio). Tamaño del enjambre.

- **n_gen** (entero, obligatorio). Número de iteraciones.

- **w_max** (real,obligatorio). Peso máximo de inercia.
- **w_min** (real,obligatorio). Peso mínimo de inercia.
- **c1** (real, obligatorio). Constante de aceleración 1.
- **c2** (real, obligatorio). Constante de aceleración 2.
- **iPart** (entero, opcional). Número de particiones internas, para distribución sobre el cluster. Si se pasa el valor por parámetro de línea de comandos tiene preferencia sobre este.
- **nRepetitions** (entero, obligatorio). Número de repeticiones del algoritmo que se realizarán sobre cada partición del dataset. Esto es, se ejecutará el medio esas veces sobre el conjunto de datos. Su sentido es que se está tratando con métodos estocásticos.
- **reportFile**(cadena, opcional). Ruta para guardar un informe de los resultados obtenidos. El pasado como parámetro de línea de comandos tiene preferencia.
- **csvFile** (cadena, opcional). Ruta para guardar un informe resumido de los resultados obtenidos en formato csv. El pasado como parámetro de la línea de comandos tiene preferencia sobre este.