UNIVERSIDAD FRANCISCO DE VITORIA

ESCUELA POLITÉCNICA SUPERIOR

GRADO EN INGENIERÍA INFORMÁTICA

PROYECTO FINAL DE GRADO

Herramienta software de redes neuronales para el aprendizaje no supervisado

Autor: Oliver Hoffman García

UNIVERSIDAD FRANCISCO DE VITORIA

ESCUELA POLITÉCNICA SUPERIOR

GRADO EN INGENIERÍA INFORMÁTICA

PROYECTO FINAL DE GRADO

Herramienta software de redes neuronales para el aprendizaje no supervisado

AUTOR:

Oliver Hoffman García

TUTOR: Álvaro García-Tejedor

Julio – 2014

I. AUTORIZACIÓN Y CALIFICACIÓN DEL PFG

Título: Herramienta software de redes neuronales para el aprendizaje no supervisado

Autor: Oliver Hoffman García

Tutor: Álvaro García-Tejedor

VISTO BUENO

VºBº Tutor del PFG:
Fdo:

Lugar y fecha: Universidad Francisco de Vitoria – 30 de junio del 2014

CALIFICACIÓN

CUALITATIVA:	
NUMÉRICA:	

Conforme Presidente:	Conforme Secretario:	Conforme Vocal:
Fdo:	Fdo:	Fdo:

Lugar y fecha: _____

II. RESUMEN Y PALABRAS CLAVE

II.1 RESUMEN

Los mapas auto asociativos de Kohonen son un modelo de red neuronal de aprendizaje no supervisado que ha dado enormes resultados en el área de la inteligencia artificial y el análisis de datos, entre los que destacan el reconocimiento del habla, la clasificación de documentos, el reconocimiento de imágenes y la compresión de datos, entre otros.

Actualmente no hay una gran variedad de herramientas de fácil acceso que permitan el uso de una red de Kohonen para el análisis de datos. En particular ninguna gratuita que ofrezca la posibilidad de subir archivos del usuario con sus propios datos para que estos sean analizados por la red y a la vez ofrezca una visualización amigable de los resultados para su interpretación.

En este proyecto se ha pretendido hacer frente a esto y se ha desarrollado una aplicación que implementa una red de Kohonen para recibir datos introducidos por el usuario para su análisis y permite una posterior visualización de los resultados. También se ha buscado ofrecer un mecanismo de comparación de resultados, en el que se guardan los parámetros y las soluciones ofrecidas por la herramienta tras cada análisis para que resulte mucho más sencilla su comparación. Finalmente, se ha realizado de tal manera que posea una interfaz gráfica intuitiva para el usuario que le permita personalizar los parámetros de la red.

II.2 PALABRAS CLAVE

A continuación se describen las palabras clave más utilizadas en esta memoria:

red de Kohonen, redes neuronales, inteligencia artificial, clasificación de datos, aprendizaje no supervisado, mapas autoasociativos, distancia euclídea, vectores, clases, matriz, mapa de activación

III. TABLAS DE CONTENIDO

III.1 INDICE

III.2 LISTA DE TABLAS

III.3 LISTA DE FIGURAS

1. INTRODUCCIÓN

El proyecto de fin de grado presentado en esta memoria consiste en el desarrollo de una herramienta de aprendizaje no supervisado mediante un modelo de redes neuronales llamado redes de Kohonen o mapas auto asociativos (1).

Este tipo de modelo de redes neuronales tiene la característica de permitir clasificar datos sin la necesidad de conocer a priori la clase a la que puedan pertenecer o los patrones que permitan relacionar dichos datos.

Este modelo de inteligencia artificial es uno muy potente con un éxito demostrado en diferentes tipos de clasificación de datos(1), dentro de los que se incluyen el procesado de imágenes y reconocimiento de figuras, procesamiento de voz y el habla, predicción del clima, predicción de series temporales, clasificación de genes en los campos de la biología, etc.(2).

A su vez, es un modelo que se enseña a los alumnos del Grado en Ingeniería Informática en la Universidad Francisco de Vitoria en la asignatura de Inteligencia Artificial II, y es justamente este ámbito al que la aplicación desarrollada busca hacer referencia. Se ha desarrollado una aplicación que busca servir de ayuda al equipo docente de la universidad para facilitar la enseñanza de esta asignatura, aportando a los alumnos y al profesorado de una herramienta con una interfaz gráfica intuitiva que pueda ser utilizada con los menores recursos y esfuerzo posibles, siempre por supuesto garantizando la fiabilidad de las predicciones y el uso correcto de los algoritmos.

El resultado tras largas horas de trabajo en lo que al diseño y la programación de la aplicación se refiere ha sido un éxito. Se ha podido aprovechar las librerías de javascript que otorgan una capacidad de visualización de datos extremadamente buena (entre ellas la conocida D3.JS desarrollada por los profesores Jeff Heer, Mike Bostock y Vadim Ogievetsky de la Universidad de Stanford(3)) y se ha combinado con la potencia de cálculo de Java en el back-end, garantizando la fiabilidad de los cálculos y las predicciones asociadas.

A su vez, y siguiendo el aspecto formativo que caracteriza a la herramienta desarrollada, se han incluido diferentes de formas de comparar los resultados del entrenamiento de las redes y de las clasificaciones realizadas, mediante el uso de ficheros XML y PDF que relacionan los datos, los parámetros y los pesos de las redes, los resultados de la clasificación, y los resultados de la visualización de la herramienta. Todos estos ficheros facilitan también la labor del profesor en la corrección de las diferentes prácticas que se puedan realizar.

En lo que al uso de la herramienta se refiere, la misma se debe alojar en un servidor dentro de una red local (utilizando Apache Tomcat[1]) al que se accederán desde los diferentes terminales conectados mediante la dirección IP del servidor desde el explorador. Una vez dentro, el uso de la herramienta consiste en la subida de los archivos de datos y la especificación de los parámetros de la red, bien sea de forma manual o automática con datos guardados previamente en un fichero.

Una vez seleccionados los ficheros a utilizar, la herramienta ofrece 3 modos de utilización: entrenamiento, clasificación, y visualización. Todos ellos se relacionan de alguna manera, que será explicada más adelante en esta memoria.

[1] Disponible en: http://tomcat.apache.org/download-70.cgi

Para garantizar la posibilidad de ser utilizada por diferentes personas, la herramienta hace uso de sesiones para cada usuario. Por cada sesión, se crea una carpeta, en la que serán guardados los ficheros subidos por el usuario y aquellos generados por la herramienta. Una vez que la sesión caduca o el usuario la cierra, esta carpeta es eliminada del servidor. En todo momento se da la usuario la posibilidad de descargar todos sus ficheros.

La tecnología utilizada para el desarrollo de la herramienta ha sido la de una *Java Web Aplication* mediante:

- Eclipse como entorno de desarrollo[2]
- Javascript como herramienta de visualización de datos
- Java como lenguaje de programación principal
- JSP como lenguaje intérprete de la web

Cabe destacar que hay ciertos parámetros más avanzados de las redes de Kohonen (entre ellos el uso de mecanismo de consciencia y el tipo de normalización Z-AXIS) que no han sido desarrollados debido a la complejidad de los mismos y que se salen del ámbito de la herramienta.

Como aplicación práctica, la herramienta permite la clasificación de datos mediante el modelo de aprendizaje no supervisado de redes de Kohonen (cuyo funcionamiento será explicado en detalle más adelante). Dichos datos pueden provenir de diferentes fuentes y serán utilizados por la herramienta siempre que cumplan con la estructura de ficheros requerida por la herramienta (ver el apartado 6.1 del presente documento). Una vez analizados dichos datos, se generarán agrupaciones de los mismos (clases) tras haber entrenado la red. Finalmente, una vez que la red sea entrenada correctamente, podrá utilizarse posteriormente para predecir la clase a la que una muestra debería pertenecer. Todos los resultados podrán ser analizados mediante la interfaz gráfica proporcionada y los informes generados.

[2] Disponible en: http://www.eclipse.org/recommenders/download/

2. PROBLEMA A RESOLVER Y OBJETIVOS DEL PROYECTO

2.1 MOTIVACIÓN Y ORIGEN

Los mapas auto asociativos de Kohonen(1) son un modelo de red neuronal de aprendizaje no supervisado que ha dado enormes resultados en el área de inteligencia artificial y análisis de datos, entre los que destacan el reconocimiento del habla, la clasificación de documentos, el reconocimiento de imágenes y la compresión de los datos, entre otros.

Actualmente no hay una gran variedad de herramientas de fácil acceso que permitan el uso de una red de Kohonen para el análisis de datos. En particular ninguna gratuita que ofrezca la posibilidad de subir archivos del usuario con sus propios datos para que estos sean analizados por la red y a la vez ofrezca una visualización amigable de los resultados para su interpretación.

El aprendizaje de este método de clasificación característico de la inteligencia artificial no es sencillo por todos los posibles parámetros que puede tener y la complejidad de los algoritmos, y en muchos casos la visualización de las clasificaciones no es fácil de representar. Esto complica de forma considerable la tarea del profesor de enseñar de forma efectiva y práctica el uso de estas redes.

En este proyecto se ha pretendido hacer frente a esto y se ha desarrollado una aplicación que implementa una red de Kohonen para recibir datos introducidos por el usuario para su análisis y permite una posterior visualización de los resultados. También se ha buscado ofrecer un mecanismo de comparación de resultados, en el que se guardan los parámetros y las soluciones ofrecidas por la herramienta tras cada análisis para que resulte mucho más sencilla su comparación. Finalmente, se ha realizado de tal manera que posea una interfaz gráfica intuitiva para el usuario que le permita personalizar los parámetros de la red.

2.2 DATOS QUE LO SUSTENTAN

Como se ha explicado en el punto anterior, los éxitos que se han conseguido gracias a las redes neuronales, y en particular a las redes de Kohonen, han supuesto un enorme avance en la inteligencia artificial y el análisis de datos. La solución ofrecida por este proyecto busca hacer frente a la necesidad de una herramienta de fácil acceso al personal docente de la universidad para facilitar la enseñanza y mejorar el entendimiento de los alumnos de este tema tan apasionante.

Todos los datos referentes a la investigación realizada para realización de la aplicación se recogen en el apartado 3: *Antecedentes y estado del arte* del presente documento.

2.3 IMPACTO DE LA SOLUCIÓN DEL PROBLEMA

Las redes de Kohonen ofrecen un abanico de posibilidades para al análisis de datos, tales como:

- **Clustering:** (agrupación de los datos en grupos para su interpretación)
- **Reducción de dimensiones:** representan estructuras de espacios de varias dimensiones en un mapa 2-D.
- **Extracción de características:** significativas de los datos de entrada de la red.
- **Predicción:** del grupo al que pertenecerán ciertos datos una vez entrenada la red.

Todas estas posibilidades son visibles en el proyecto final al igual que una visualización sencilla de los resultados, ya que será utilizada para la enseñanza y la explicación de los mapas auto asociativos de Kohonen.

2.4 PROBLEMA DE INVESTIGACIÓN EN FORMA DE PREGUNTA

¿Cómo se puede ofrecer una herramienta completa de aprendizaje no supervisado de forma que la interpretación de sus resultados sea sencilla y su interfaz gráfica sea fácil de utilizar?

2.5 PREGUNTAS DE INVESTIGACIÓN

- ¿Cuáles son los tipos de clasificación que hay?
- ¿Cómo funcionan las redes neuronales?
- ¿Cómo funcionan las redes de Kohonen?
- ¿Cuál es la mejor forma de desarrollar una herramienta que mezcle un alto coste de cómputo con una visualización amigable de datos?
- ¿Cuál va a ser el diseño de la interfaz gráfica y cómo se va a asegurar que sea fácil de manejar?
- ¿Cuáles van a ser las distintas formas de visualización ofrecidas por la herramienta?
- ¿Qué parámetros se van a ofrecer para personalizar los cálculos de la red de Kohonen?
- ¿Cómo va a ser el formato de los archivos de datos a cargar en la herramienta?
- ¿Cómo se van a almacenar los resultados?
- ¿Cómo se va a ofrecer la posibilidad de guardar los resultados de cada cálculo para su comparación posterior? ¿Va a ofrecerse de forma fácil e intuitiva? ¿Cómo se realizará exactamente dicha comparación?

2.6 OBJETIVO GENERAL

El **objetivo general** consiste en la construcción de una herramienta de aprendizaje no supervisado mediante mapas auto asociativos de Kohonen. Se busca que su interfaz gráfica sea amigable al usuario, sus herramientas de visualización sencillas de interpretar, y que exista la posibilidad de personalizar todos los parámetros relevantes de la red de Kohonen.

2.7 OBJETIVOS ESPECÍFICOS

- Estudiar en profundidad el funcionamiento de las técnicas de clasificación y en particular las redes neuronales y redes de Kohonen.

- Desarrollar una herramienta destinada a la enseñanza de la inteligencia artificial que permita la clasificación de datos mediante las redes de Kohonen.

- Diseñar una interfaz gráfica amigable que permita de forma intuitiva la navegación del usuario por la herramienta.

- Debido a la facilidad de los navegadores web para visualizar datos (en gran medida gracias a las herramientas disponibles de Javascript), la aplicación será desarrollada en un entorno web.

- Se seleccionarán los parámetros a modificar por el usuario para el cálculo de la red.

- El formato de los archivos de datos se ha determinado conjuntamente con el tutor. Se ha decidido implementar el formato XML y JSON como principales entradas y salidas a la herramienta. Para las muestras también se acepta el formato .DAT (el cual consiste en un fichero de texto plano cuyos datos son separados por tabulador). Para más información en lo referente al formato de ficheros, ver el apartado 6.3.3 del presente documento.

- El almacenamiento de los resultados también se almacenará en un archivo para su futura visualización.

- En lo que a la comparación de resultados se refiere, se ideará un mecanismo en el que se guarde una lista con las distintas ejecuciones de la aplicación, así como los resultados de cada una para su comparación. Se podría incluso establecer una forma de visualización que mostrara todas a la vez para facilitar el proceso.

- Garantizar la exactitud de los resultados en base a pruebas de un conjunto de datos escogidos por el alumno o conjuntamente con el tutor.

3. ANTECEDENTES. ESTADO DEL ARTE.

3.1 REDES NEURONALES

3.1.1 INTRODUCCIÓN

Los sistemas tradicionales empleados para la predicción de demanda están basados en el tratamiento estadístico de series temporales(4). Los archivos históricos que conservan la evolución temporal de las variables a predecir son utilizados para construir modelos de evolución empleando métodos de regresión (dinámica y estática), medias móviles o modelos de Box-Jenkins. El procedimiento habitual requiere conocer tanto la serie temporal en sí misma como el tipo y grado de influencia que tienen sobre ella otras propiedades o variables. En definitiva, obliga a establecer un modelo estadístico de comportamiento de la serie y por lo tanto, del proceso del mundo real que representa.

A medida que se ha intentado predecir situaciones más complejas, bien por el tipo de relación entre las variables explicativas, bien por la complejidad del dominio en sí, las predicciones basadas en el tratamiento de series temporales comienzan a desviarse de la realidad.

En los últimos años se ha optado por una solución radicalmente distinta, pero cuya eficacia ya ha sido suficientemente probada. Se trata del empleo de las Redes Neuronales aplicadas a la predicción de demanda(5). Esta tecnología está basada en la modelización en el ordenador de los mecanismos biológicos de reconocimiento de patrones (incluidos los patrones temporales que intervienen en las predicciones) y en su capacidad de aprendizaje, a partir de ejemplos, de las relaciones existentes entre las distintas variables que intervienen en una serie temporal. El resultado de aplicar esta tecnología son sistemas mucho más rápidos y flexibles, que son capaces de adaptarse a la evolución de la demanda del mercado, y que no requieren la elaboración de modelos complejos, sino únicamente la existencia de archivos en los cuales se recoja la historia del comportamiento de la demanda en fechas pasadas.

La capacidad de las Redes Neuronales para tratar problemas de predicción de Series Temporales ha enfocado el interés en esta solución alternativa. Los sistemas probabilísticos lineales, desarrollados a partir de los trabajos de Box-Jenkins(4), parten de suposiciones previas como la estacionalidad de la serie, lo cual raramente ocurre en la realidad. En cambio, modelos neuronales relativamente simples, como es el caso del Perceptrón Multicapa, no parten de esas premisas. Las Redes Neuronales presentan un comportamiento que se adapta a la modelización de series temporales.

Las propiedades que resultan más adecuadas para los problemas de predicción son:

- Procesamiento en paralelo, que permite el tratamiento simultáneo de gran cantidad de información.
- No es necesario conocer la estructura del modelo
- Facilidad para reconfigurar el modelo de predicción, añadiendo o eliminando variables sobre el modelo inicial
- Utilización de variables correlacionadas o no-correlacionadas. Una red neuronal es capaz de detectar las correlaciones entre variables sin conocerlas previamente.
- Implementan un modelo no lineal de las relaciones existentes entre las variables del modelo.
- Robustez frente a datos incompletos o afectados de ruido.

3.1.2 ¿QUÉ SON LAS REDES NEURONALES?

Los conocimientos sobre los sistemas biológicos han sido modelados e incorporados como nuevas tecnologías en las ciencias de la computación. El resultado ha sido un creciente interés por lo que se conoce como biocomputación o aplicación de modelos biológicos a la solución de problemas informáticos complejos. Así, los estudios de evolución y genética molecular han dado lugar a los Algoritmos Genéticos, mientras que los avances en neurofisiología cerebral han originado las Redes Neuronales Artificiales.

Los ordenadores actuales son capaces de realizar con precisión y rapidez tareas complejas que han sido formuladas explícitamente. Sin embargo, el cerebro los sobrepasa en aquellas tareas como visión, habla y en general reconocimiento de patrones espacio-temporales complejos en presencia de ruido o datos incompletos. Estas labores son muy complejas para los ordenadores convencionales, a pesar de que un ciclo de cálculo en el cerebro requiere 5 milisegundos frente a los $4x10^{-6}$ milisegundos de un CRAY-3. No obstante, el cerebro distribuye su capacidad de procesamiento entre 10^4 millones neuronas, cada una de ellas conectada con otras diez mil. Este tipo de procesamiento masivamente paralelo del impulso nervioso es el que le otorga tan sorprendentes propiedades y hace que el tiempo medio de reconocimiento de una imagen sea solamente de 500 milisegundos (aproximadamente 100 ciclos).

El desarrollo de neuronas y redes neuronales artificiales comenzó en la década de los 40. El procesamiento de la información realizado por estos sistemas es un modelo alternativo al de Von Neumman en el que se basan los ordenadores digitales(6). Estos tratan secuencialmente instrucciones almacenadas en una memoria que manipulan datos de la misma memoria. Las Redes Neuronales, en cambio, son un grupo de procesadores elementales altamente interconectados entre sí, que emulan algunas de las características de las neuronas biológicas, y que procesan la información en paralelo. En cada elemento de cálculo (neurona), el proceso y la memoria se encuentran íntimamente ligados. La principal cualidad de estos sistemas la constituye su adaptabilidad dinámica, esto es, su capacidad para variar de comportamiento en situaciones cambiantes mediante el aprendizaje o la generalización.

En estos sistemas la información no se almacena en un emplazamiento físico único ya que la capacidad de almacenamiento individual es muy limitada. La información aparece distribuida por toda la estructura de la red, concentrándose en las uniones de los distintos elementos. Las funcionalidades de estos sistemas vienen dadas por la globalidad del sistema más que por la aportación individual de cada elemento.

Los modelos y algoritmos neuronales pueden ser simulados en ordenadores convencionales y de hecho ésta constituye la forma más común de emplear esta tecnología. Sin embargo alcanzan su máxima potencia cuando son implementadas en un hardware específico: los neurocomputadores.

En definitiva, una red neuronal es un sistema compuesto por un conjunto de elementos de procesamiento simples (neuronas), conectados entre sí (red), cuyo comportamiento está determinado por la topología y pesos de sus conexiones.

Se pueden definir a su vez como un sistema de computación que consta de un gran número de elementos simples, altamente interconectados, que procesan la información respondiendo dinámicamente frente a estímulos externos. (Al decir dinámicamente, implica que los pesos de sus conexiones van cambiando y evolucionando).

También se dice que son un conjunto de autómatas conectados entre sí que generan propiedades emergentes.

3.1.3 LA NEURONA: EL ELEMENTO BÁSICO

El componente mínimo de una Red Neuronal es la *neurona*. Es un dispositivo que transforma varias señales de entrada en una única señal de salida. Las entradas pueden proceder bien de otras neuronas, o bien ser entradas a la red desde el exterior. Cada entrada se encuentra modulada por un factor (peso) que afecta a la intensidad de la conexión entre la neurona receptora y emisora. La salida asimismo puede transmitirse a otras neuronas o ser una de las señales de salida de la red.

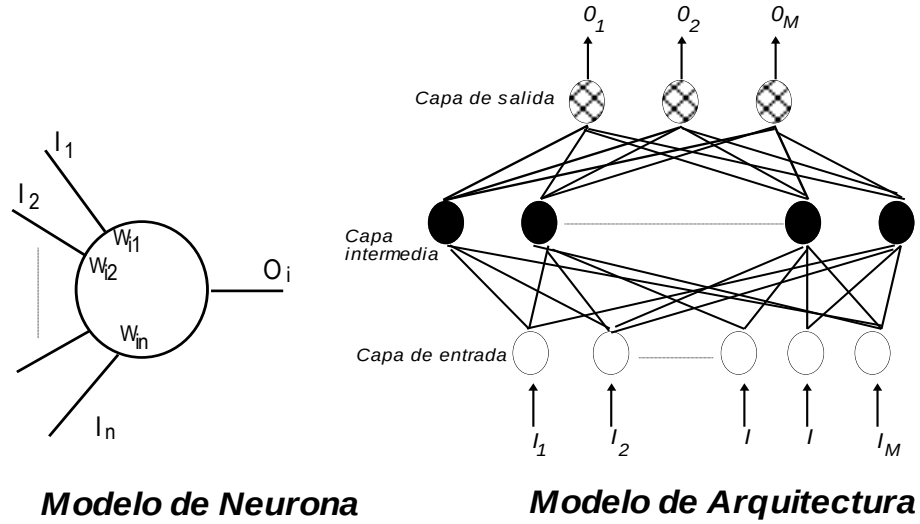

Figura 1: Modelo de neurona y arquitectura neuronal multicapa

Una neurona es en realidad un procesador muy simple: un sumador. Tiene una capacidad limitada de cómputo restringida a un conjunto elemental de instrucciones (sumas y productos) y una memoria pequeña para almacenar pesos y activaciones. La función computada en cada elemento de procesamiento se puede dividir en dos pasos(6):

1. *Suma ponderada* de sus entradas. Las señales de entrada que llegan a la neurona, moduladas por sus pesos correspondientes, son sumadas para dar la entrada neta.

2. *Función de activación*. Es la función aplicada a la entrada neta. Compara la entrada total con un valor umbral que define la cantidad de estímulo necesario para que se active la neurona. Si se sobrepasa este valor se genera la señal de salida. Distintos tipos de funciones pueden ser empleadas, definiendo cada una de ellas un tipo de neurona y un tipo de salida: lineal, escalón, sigmoidea, etc.

3.1.4 ARQUITECTURA DE UNA RED NEURONAL

Para definir totalmente una Red Neuronal no basta con describir el comportamiento individual de sus elementos (neuronas), sino que hay que especificar además la interconexión existente entre ellas. Normalmente la red se ordena en varias capas, cada una de ellas con un conjunto de neuronas de número variable y comportamiento similar. Al conjunto de pesos que ponderan las conexiones entre las distintas neuronas se le denomina matriz de pesos.

Existen tres tipos de capas:

1. *Capa de entrada*. El número y tipo de neuronas que constituyen esta capa depende de los datos de entrada al problema.

2. *Capas intermedias*. Puede ser más de una, dependiendo del tipo y complejidad del problema que va a resolver la red. Mediante el tratamiento adecuado de estas capas se consiguen las propiedades de generalización, extracción de características, adaptabilidad
3. *Capa de salida*. El número de neuronas de esta capa depende del formato esperado de salida de la red.

Cada capa está conectada total o parcialmente a la inmediata posterior, excepto la última capa que constituye la salida de la red. Las diferentes formas de distribuir, conectar e interrelacionar estos tres tipos de capas, junto al tipo de neuronas que constituyen cada una de ellas, nos van a definir los diferentes modelos de red existentes.

La salida de la última capa de la red es la respuesta del problema planteado. Las respuestas esperadas deben ser iguales a las respuestas de salida. De no ser así, la red se ha equivocado. Las entradas son ponderadas con los pesos para dar una respuesta, si la respuesta no es la esperada, se modifican los pesos en la matriz de pesos hasta conseguirla. Un peso con valor 0 implica que el valor de esa entrada le es indiferente al red para el problema planteado.

La arquitectura de una red neuronal también se puede clasificar por el tipo de conexiones entre sus neuronas:

- Por la actividad, pueden ser excitatorias (suman) o inhibitorias (restan).
- Por la topología pueden ser lateral, layer-to-layer o recurrentes. Estas últimas son aquellas en las que la salida de una neurona vuelve a ser la entrada de la misma.
- Por la dirección en la que fluye la información pueden ser feedforward o feedback.

La potencia computacional de una red depende del número de capas que tenga y del número de conexiones.

3.1.5 PROCESAMIENTO DE DATOS

Las Redes Neuronales realizan básicamente *comparación de patrones*. Por patrón entendemos el conjunto de datos suministrados a la capa de entrada que definen un problema, o bien la solución al mismo. Pueden ser tanto espaciales (una imagen formada por pixeles, un perfil de cliente de una entidad bancaria) como espacio-temporales (la señal acústica, las series temporales). Este proceso distribuido implica a todas las neuronas de la red, bien en la recepción del patrón de entrada o en la generación del patrón de salida. El mecanismo de comparación funciona bajo dos condiciones distintas:

- Dado un patrón A de entrada a la red, ésta produce el patrón B de salida. Previamente, la red ha sido enseñada a asociar ambos tipos de patrones.
- Dato un patrón de entrada incompleto, la red produce la versión completa del mismo.

En el primero de los casos, la red se está comportando como un *clasificador*, ya que asigna una categoría (patrón de salida) a cada entrada. En el segundo, se comporta como un *predictor*, ya que es capaz de generar elementos ausentes en los datos de entrada. En este caso, la predicción puede ser o no temporal.

Si no se encuentra el patrón de salida exacto, ya que la comparación no es perfecta, se generará aquel que más encaje con las características de la entrada.

3.1.6 APRENDIZAJE EN UNA RED NEURONAL

La principal característica de una Red Neuronal es su capacidad de aprender. La red modifica su propia estructura (matriz de pesos) adaptándola hasta conseguir un algoritmo para solucionar el problema a partir de características extraídas de los ejemplos con que se la entrena. Este algoritmo representa la asociación entre los patrones de entrada y los de salida. Como resultado, la red genera un modelo interno del problema, almacenado de forma distribuida en la matriz de pesos y en la arquitectura de la red.

Al aprender, las neuronas deben producir idéntica respuesta ante idénticos estímulos, para lo cual se modifican los pesos de las conexiones que unen las neuronas. Esta modificación se puede efectuar siguiendo varios criterios, todos ellos con inspiración neuro-fisiológica y soportados en mayor o menor medida por los experimentos de Pavlov y por la ley de Hebb, enunciada en 1949.

Distintas interpretaciones del enunciado original de Hebb han dado lugar a distintos tipos de aprendizaje:

- *Supervisado*. Cada ejemplo de entrenamiento consta de un par formado por las entradas al sistema y la salida esperada para esas entradas. La Red Neuronal necesita de un supervisor que le diga cuan buena es su respuesta frente al estímulo dado. En el caso de que cometa alguna incorrección tendrá lugar cierta modificación interna del sistema tendente a mejorar su rendimiento. El algoritmo más utilizado de corrección es la regla *Delta Generalizada*. A continuación se pasara a explicar este tipo de aprendizaje de forma más concreta.
- *No supervisado*. El conjunto de entrenamiento está constituido solamente por elementos de entrada, sin ninguna salida esperada. El aprendizaje se hace a través de la asociación de la información recibida con la almacenada. Este tipo de aprendizaje se ajusta a problemas de clasificación en que las entradas pueden venir distorsionadas, y de las que no se puede determinar con absoluta seguridad cual debería ser su salida esperada. Más adelante, en el apartado 3.2 del presente documento se explica este tipo de aprendizaje, sobre el cual el proyecto de fin de grado desarrollado se ha basado.

Otras características

- **Generalización**: El aprendizaje conduce a elaborar un algoritmo de solución de un problema. De esta manera son capaces de ofrecer una respuesta ante casos que no habían sido considerados durante la fase de entrenamiento. El patrón de salida es generado de forma que se corresponda lo más posible con el patrón de entrada.
- **Tratamiento simultáneo de grandes cantidades de información.** Como resultado del paralelismo masivo implícito, cada elemento individual calcula una función elemental sin necesidad de cooperar con las demás. Esto proporciona una gran capacidad de cómputo y un tiempo de respuesta muy pequeño para procesar todos los datos de entrada.
- **Tolerancia a fallos**. La memoria es asociativa y distribuida. Esto implica que un dato no se encuentra en un único sitio, sino repartido por toda la estructura. Por lo tanto el sistema no deja de funcionar aunque una serie de sus componentes fallen. El sistema se degrada atenuadamente con una función de error continua.
- **Reconstrucción de datos parciales**. Los datos de entrada no necesitan ser explicitados rigurosamente. Cuando el conocimiento del problema es parcial (ausencia en tiempo real de un determinado dato o entrada afectada de ruido) la red reconstruye en las capas intermedias los datos que necesita.

3.1.6.1 APRENDIZAJE SUPERVISADO MEDIANTE REDES NEURONALES

Los algoritmos de aprendizaje supervisado pueden aprender a reproducir comportamientos y razonamientos mediante estímulos (mecanismo de recompensa/castigo) y simulan el comportamiento de una persona adulta.

Están basados en reglas de naturaleza global (los cambios producidos por el mecanismo de aprendizaje afectan a toda la matriz de pesos). Los datos determinan la relación entrada/salida a la que la red debe aproximarse.

Su campo de aplicación es más amplio y se extiende a *control, predicción, y reconocimiento.*

La predicción en redes neuronales es un procesamiento en paralelo en el que no es necesario conocer la estructura del modelo, sólo disponer de suficientes datos históricos. Se caracteriza por:

- No requieren de expertos altamente especializados.
- Su desarrollo puede ser incremental.
    - Facilidad para reconfigurar el modelo (añadiendo nuevas neuronas en la capa de entrada)
    - Mejora continuada de las aplicaciones a fin de adaptarlas a condiciones cambiantes reentrenando la red con datos más recientes.
- Se obtienen buenas predicciones, incluso aunque no todas las variables explicativas hayan sido consideradas por la red.
- Uso indistinto de variables correlacionadas o independientes, ya que la red descarta las innecesarias.
- Modelo no lineal que permite aprender en dominios no lineales.

Las principales ventajas de las técnicas neuronales para la predicción son mejores niveles de exposición y mayores horizontes de predicción, con coste desarrollo menores.

El rendimiento (eficiencia) de una red neuronal supervisada depende del valor conjunto de parámetros (matriz de pesos) que determinan el ajuste entre las entradas y las salidas. Puesto que es difícil determinar con precisión los valores de múltiples parámetros (tantos como pesos hay en la red), se emplea un método de aprendizaje:

- Se crea una red con valores al azar de los parámetros
- Esta red se emplea para llevar a cabo transformaciones entrada-salida en un problema real.
- Los valores finales de los pesos se obtienen modificándolos adecuadamente de acuerdo con los errores que la red comete en el proceso.

En el caso de los modelos supervisados, el método de entrenamiento se conoce como regla delta o descenso de gradiente. Su proceso es el siguiente:

- Se inicializan los pesos aleatoriamente W(t).
- Se determina, la dirección de la pendiente más pronunciada en dirección hacia abajo.
- Se modifican los pesos para que se encuentren un poco más abajo en la superficie.

$$W(t+1) = W(t) + \Delta W(t)$$

El aprendizaje se consigue mediante:

- **Entrenamiento:** es un proceso iterativo que precisa un conjunto de entrenamiento, del que se conozca la salida esperada y que sea suficientemente amplio para que cubra el espacio muestral. Su objetivo es la minimización del error en el conjunto de entrenamiento. Cuando una pasada completa del conjunto de entrenamiento ocurre sin error, el entrenamiento está completo.
- **Ley de aprendizaje:** es una formulación matemática de ajuste de la matriz de pesos en función de los pares entrada/salida con que se entrena la red. La más importante es el algoritmo de retropropagación del error (regla Delta Generalizada) también conocida como backpropagation. Es utilizada en el perceptrón multicapa (redes multicapa no lineales). Otras reglas delta importantes son la ley de aprendizaje Widrow-Hoff (usada en Adaline) y la ley de aprendizaje de Rosemblatt (usada en el perceptrón).

De esta forma, el aprendizaje correcto permite que la red generalice: una entrada nueva genera una salida correcta.

El algoritmo de aprendizaje es:

1. Iniciación de pesos
2. Se aplica un patrón de entrada (entradas y salida deseada)
3. Se computa la salida lineal que se obtiene de la red.
4. Se calcula el error cometido para dicho patrón.
5. Se actualizan las conexiones mediante la Regla Delta correspondiente.
6. Se repiten los pasos del 2 al 5 para todos los patrones de entrenamiento.
7. Si el error cuadrático medio es un valor reducido aceptable, termina el proceso, sino vuelve al paso 2

Existe también una variante en la cual se calcula el error cuadrático medio sobre todos los patrones y se actualiza los pesos en función de este error.

A continuación se describen diferentes modelos de redes neuronales que siguen el aprendizaje supervisado:

EL PERCEPTRÓN

Frank Rosenblatt ideó los primeros modelos lineales en 1959. Hizo un perceptrón que modeliza la retina. Utilizaba una neurona formal binaria (0,1) con pesos sinápticos ajustables y aplicaba la Ley de Hebb mediante la regla Delta de aprendizaje para entrenar el perceptrón, la cual tenía una función de activación de umbral (6).

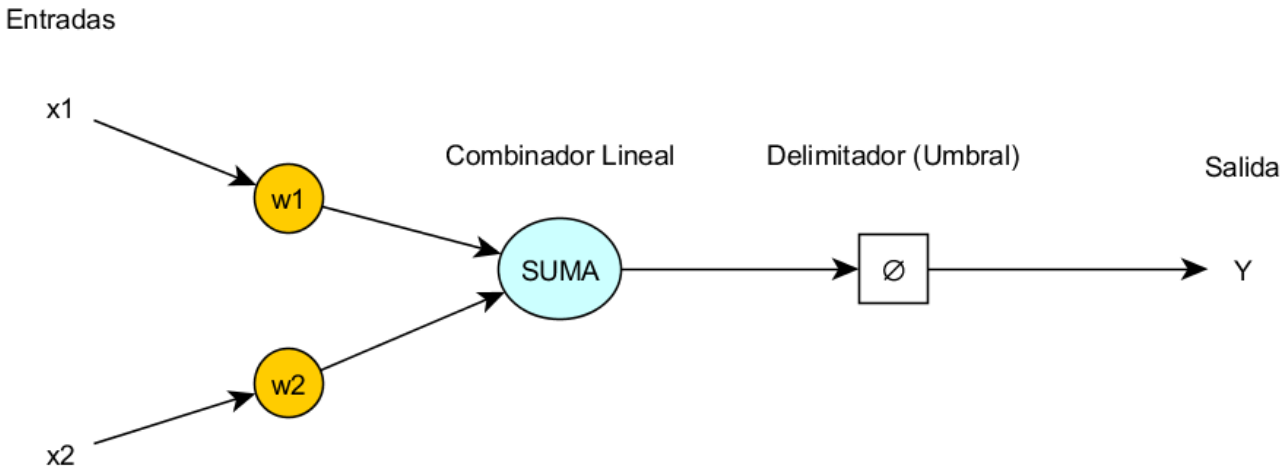

Figura 2: Arquitectura del perceptrón

En la figura 6 se muestra la arquitectura del perceptrón para 2 entradas (x_1 y x_2). La salida Y sólo puede ser 0 o 1 y es activada en base a la función de activación F(S,*0*):

$$F(S,\emptyset) = \begin{cases} +1 \; si \; S \geq \emptyset \\ 0 \; si \; S < \emptyset \end{cases}$$

0 es el umbral único y distinto para cada neurona de salida de la red y equivale a un peso ficticio no conectado a ninguna entrada. S es un combinador lineal, es decir, la suma ponderada de los pesos de las entradas y los datos de entrada:

$$S = \sum_{i=1}^{n} w_i * x_i$$

La función de activación corresponde a la ecuación de un hiperplano definido por una función linealmente separable. El perceptrón actúa como un clasificador que clasifica las entradas en n clases, siendo n la dimensión del hiperplano. La ecuación del hiperplano es encontrada por el perceptrón a través del proceso de aprendizaje mediante datos de entrenamiento.

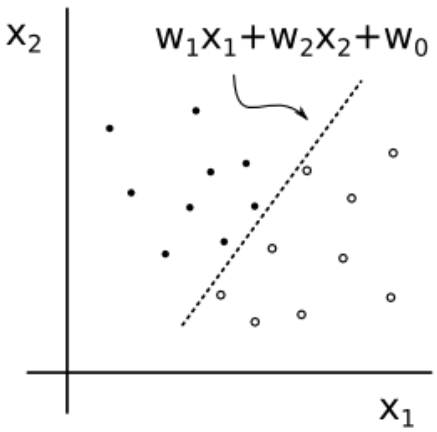

Figura 3: Clases de un perceptrón en funciones linealmente separables

El aprendizaje en el perceptrón es muy simple, reajustando los pesos si la salida es incorrecta en función del error entre el valor obtenido y el esperado:

$$W_{t+1} = W_t + \mu * (O_d - Y_t) * X$$

En la fórmula, μ es el coeficiente de aprendizaje, el cual especifica la magnitud de cambio en la red. O_d es el valor deseado, Y_t es el valor obtenido en el instante t, X es la entrada aplicada al perceptrón, y W_t es el valor del peso (matriz) en el instante t.

El teorema de convergencia del perceptrón explica que si los patrones de entrenamiento son linealmente separables, el aprendizaje con la Regla Delta converge en u n número finito de pasos encontrando un conjunto de pesos que clasificará las entradas correctamente.

Si los patrones de entrenamiento no son linealmente separables:

- No es posible encontrar un perceptrón para todos los elementos del conjunto de entrenamiento de la salida esperada.
- En su lugar, se debe intentar minimizar el error cuadrático obtenido en cada patrón mediante un regla delta, la cual:
    - Algoritmo de búsqueda local
    - Converge asintóticamente hacia mínimos locales del error
    - Se obtiene el mejor hiperplano posible (no necesariamente válido)

En definitiva, el aprendizaje es un proceso mediante el cual la salida de la unidad se incrementa o decrementa dependiendo de si esa unidad contribuye o no a las respuestas correctas para una entrada dada (Ley de Hebb**)**. Se lleva a cabo mediante pequeños ajustes de los pesos para reducir la diferencia entre la salida real y la esperada (Regla Delta**)**.

Sólo ajusta las conexiones entre las unidades de asociación (neuronas de entrada) y las unidades de respuesta (neurona de Salida). Cumple con el teorema de convergencia del perceptrón**:** si hay una solución, la encontrará solo si los problemas son de naturaleza lineal.

EL PERCEPTRÓN MULTICAPA

En 1986 Rumelhart, Hinton y Williams publican el algoritmo de aprendizaje back-propagation (regla delta generalizada)cita. Este algoritmo lo aplican a una mejora del preceptrón de Rosenblatt: el perceptrón multicapa, una combinación de varias capas de neuronas tipo perceptrón pero no lineales. Es el paradigma más extendido y utilizado de redes neuronales. Es una variación del modelo Adaline de Widrow con aprendizaje por minimización del Error Cuadrático Medio mediante Regla delta.

Sus aplicaciones son:

- Clasificación supervisada de patrones espaciales
- Reconocimiento de señales, radar, sonar o electromagnéticas.
- Monitorización en tiempo real.
- Control adaptativo sobre múltiples variables en tiempo real.
- Predicción de series temporales

Básicamente, es buena para problemas en los que es asumible que se necesite previamente un tiempo largo de entrenamiento de la red, y en los que se requieren tiempos cortos para evaluar una nueva instancia (son redes muy rápidas en el cálculo).

Sus **ventajas** son:

- Capacidad de generalización
- Soluciona la mayoría de problemas a los que se aplica
- Una vez entrenada, produce en runtime resultados muy rápidos.

Su arquitectura consiste en una red neuronal con al menos 3 capas: entrada, salida y una o más ocultas. Cada neurona en cada capa está conectada a todas las neuronas de la siguiente capa.

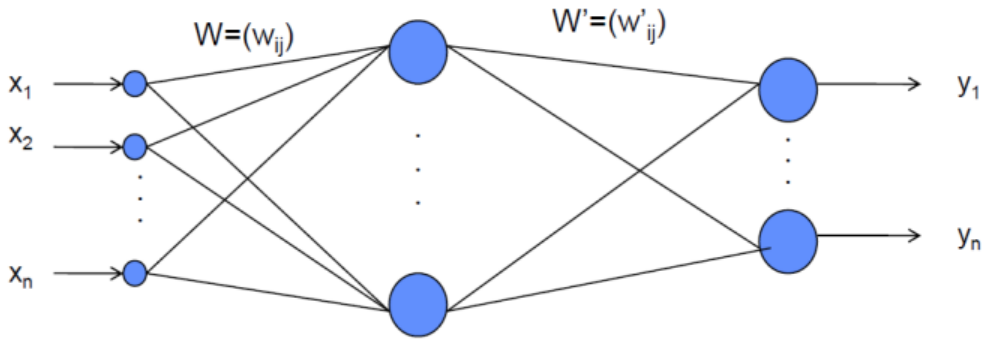

Figura 4: Arquitectura del perceptrón multicapa

Cada neurona procesa la información recibida y la respuesta se propaga actuando como entrada para todas las neuronas de la siguiente capa.

Combinando unidades en distintas capas (y siempre que F sea no lineal) aumenta la capacidad expresiva de la red (el número de funciones/hiperespacios no lineales que puede implementar).

Respecto a la matriz de pesos, cada capa está completamente conectada con la siguiente:

- W_{ih} -> matriz de pesos capa de entrada - capa oculta
- W_{ho} -> matriz de pesos capa oculta - capa de salida

Las conexiones se ajustan durante el entrenamiento:

- El entrenamiento se basa en la presentación sucesiva y reiterada de pares de vectores en las capas de entrada y salida.
- La red crea un modelo a base de ajustar la matriz de pesos en función de los pares de vectores de entrenamiento.

A la hora de establecer la arquitectura de la red, hay ciertos aspectos a tener en cuenta:

- ¿Cuántas capas ocultas utilizar? Teóricamente puede haber cualquier número de capas ocultas, si bien mediante el teorema de aproximación universal (1989) se demostró que generar una sola capa oculta con el número de neuronas adecuado es equivalente a una red con un número arbitrario de capas. De hecho es incluso mejor, ya que una sola capa modela una proyección lineal entre el espacio de entradas y el espacio de salida y aumenta la capacidad de generalización de la red (7).

- ¿Cuántas neuronas usar en cada capa? Si bien el número de neuronas en las capas de entrada y salida es obvio, el número de neuronas totales tendrá que ser:
    - Balance entre la capacidad y velocidad de aprendizaje
    - Menor que el número de patrones de entrenamiento
    - Lo contrario hace que cada neurona de la capa intermedia memorice un patrón de entrenamiento en lugar de generalizar a partir de casos individuales

El número de neuronas en la capa oculta es particularmente difícil de calcular, ya que dicho número dependerá de una forma muy compleja en varios factores:

- Número de neuronas de entrada y salida
- Número de casos de entrenamiento
- Cantidad de ruido en los datos
- Complejidad de la función de clasificación a aprender
- Arquitectura de la red
- Tipo de función de activación de la capa oculta
- Algoritmo de entrenamiento

En la mayoría de las situaciones, no hay forma de calcular el mejor número de neuronas en la capa oculta sin entrenar varias redes y estimar el error de generalización para cada una. Si hay muy pocas neuronas, el error de entrenamiento y de clasificación es muy grande debido al *underfitting* (no se entrena a la red lo suficiente). Si se utilizan demasiadas neuronas, se puede obtener un error de entrenamiento bajo pero un error alto de generalización debido al *overfitting* (se entrena demasiado a la red).(8)

En la literatura se habla de varias "reglas" a utilizar para seleccionar una arquitectura, entre las que destacan:

- "Una regla es que el tamaño de la capa oculta debe estar entre el tamaño de la capa de entrada y de la capa de salida"(9)
- "Nunca se requerirán más del doble de neuronas en la capa oculta como neuronas en la capa de entrada existan"(10)
- "Una regla es que nunca debe ser más del doble de neuronas en la capa de entrada"(11)
- "Por lo general, se deben especificar tantas neuronas en la capa oculta como dimensiones (componentes principales) sean necesarias para capturar 70-90% de la varianza del conjunto de datos de entrada"(12)

Las entradas se definen en función del tipo de problema:

- Tipo de datos: numéricos, simbólicos (cuantitativos/cualitativos)
- Carácter numérico: real, entero
- Naturaleza de los datos numéricos: discretos, continuos
- Rango de los datos: acotados, no acotados.

El número de entradas también se define en función del problema, y suele ser necesario un preprocesamiento de las entradas:

- Normalización al intervalo (0,1)
- Parametrización de variables cualitativas
- Discretización de variables continuas
- Combinación de parámetros de entrada
- Acotación del rango

Respecto a los datos de entrenamiento, el conjunto de estos datos determina lo que la red va a aprender. El tamaño de dicho conjunto:

- Debe cubrir todo el espacio estudiado, seleccionando los patrones más significativos
- 2 ó 3 veces el número de pesos
- Regla de Baum (1989): $N_{patrones} = N_{pesos}/Error_{deseado}$
- La utilización de muchos datos hace excesivamente lento el entrenamiento
- Hay que reservar datos para los conjuntos de:
  - Validación: (20% - 30%)
  - Prueba (casos concretos y difíciles)

Las características del conjunto de entrenamiento son:

- No hay reglas respecto a orden de presentación de los datos
- Se recomienda no entrenar a la red de forma consecutiva con grupos de vectores que proporcionen la misma salida

El procesamiento de los datos en el perceptrón multicapa funciona de la siguiente manera:

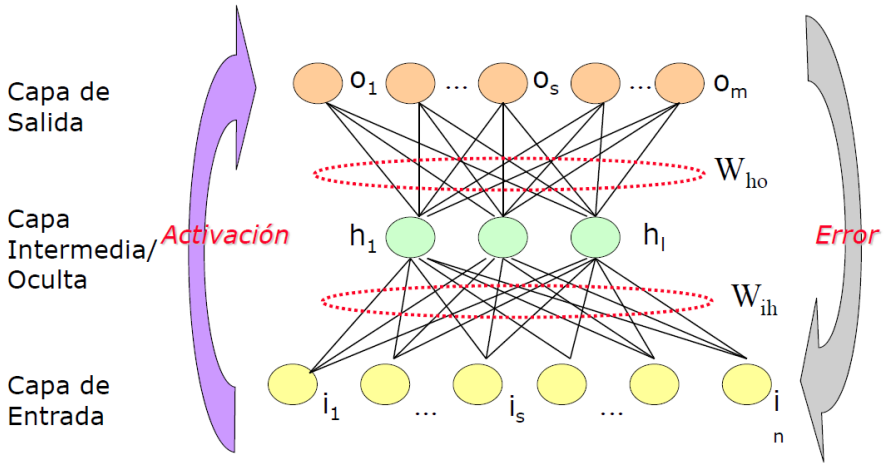

Figura 5: Procesamiento de datos en el perceptrón multicapa

El flujo de datos Forward/ejecución propaga la entrada de la red a través de las capas hasta obtener la salida aplicando la función de transferencia. El modelo es continuo y no lineal debido a la función de transferencia utilizada:

$$R^N \xrightarrow{f} R^M$$

Esta función de transferencia es una función sigmoidea o una tangente hiperbólica (no lineal). Su salida se encuentra en el intervalo (0,1) o (-1,1). A su vez, es continua y derivable en todos los

puntos (un requisito necesario para poder utilizar la regla delta generalizada). Cuanto mayor sea la pendiente (el gradiente de la función), más rápidamente la neurona alcanzará los valores próximos a 0 o 1 y más difícil será cambiar su respuesta si está equivocada.

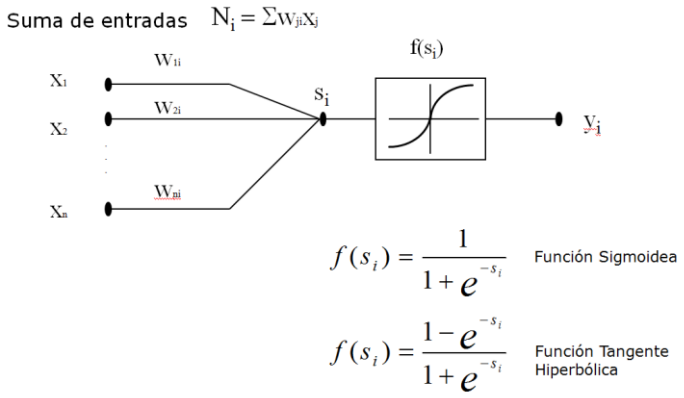

Figura 6: Función de transferencia en el perceptrón multicapa

La **propagación de la entrada** funciona de la siguiente manera:

- Se presenta el p-ésimo vector X_p a la capa de entrada que lo propaga hacia la capa oculta. La salida de cada neurona i es el mismo valor que su entrada:

$$i_k^p = x_k^p \rightarrow k = 1,2,...,n$$

- Cada neurona h de la capa oculta recibe el patrón completo desde la capa de entrada. Su salida se calcula aplicando la función de transferencia F:

$$h_k^p = f(S) = f\left(\sum_{j=1}^{n} w_{jk} i_j^p\right) \rightarrow k = 1,2,...,l$$

- Cada neurona **o** de la capa de salida recibe la salida completa de la capa oculta y la procesa aplicando a su vez F, proporcionando la respuesta de la red:

$$o_k^p = f(S) = f\left(\sum_{j=1}^{l} w_{jk} h_j^p\right) \rightarrow k = 1,2,...,m$$

Por otro lado, el flujo e datos Backward/Aprendizaje propaga el error hacia las capas interiores ajustando a la vez los pesos de las conexiones hasta que la salida de la red sea lo más próxima posible a la salida deseada. Aplica el algoritmo de aprendizaje a toda la red produciendo el ajuste de todos los pesos a la vez.

La ley de aprendizaje se conoce como regla delta generalizada o backpropagation**:**

- Es una técnica de descenso de gradiente

- Es una variación (generalización a varias capas) de la regla delta del modelo ADALINE, que fue el primero en utilizarla.
- Con capas ocultas la regla delta no sirve para el entrenamiento ya que si bien se conoce la salida deseada para cada patrón en la última capa de la estructura, no se conoce la salida deseada para las neuronas de las capas ocultas.

La regla delta generalizada utiliza la superficie de error asociada a la red para buscar el estado estable de mínimo error descendiendo por el gradiente de esa superficie. Usa el valor del error para ajustar los pesos modificándolos en un valor proporcional a lo que hemos descendido por el gradiente de la función y luego propaga ese error hacia las capas ocultas para usarlo en la actualización de sus matrices de pesos.

La definición del error en el perceptrón multicapa funciona de la siguiente manera:

- Para el vector de entrada X_p se compara la salida real obtenida con la esperada calculando el error cuadrático para todas las neuronas e la capa de salida:

$$E^p = \frac{1}{2}\sum_{k=1}^{m}(d_k^p - o_k^p)^2$$

- Después se obtiene la media de errores (**error cuadrático medio**) para todos los patrones del conjunto de entrenamiento.

$$\overline{E} = \frac{1}{2n}\sum_{p=1}^{n}\sum_{k=1}^{m}(d_k^p - o_k^p)^2$$

Donde:

- Y_k es la salida real de la neurona k
- O_k es la salida esperada de la neurona k
- m son las neuronas de la capa de salida
- n son los casos del conjunto de entrenamiento
- p es p-ésimo patrón de entrada

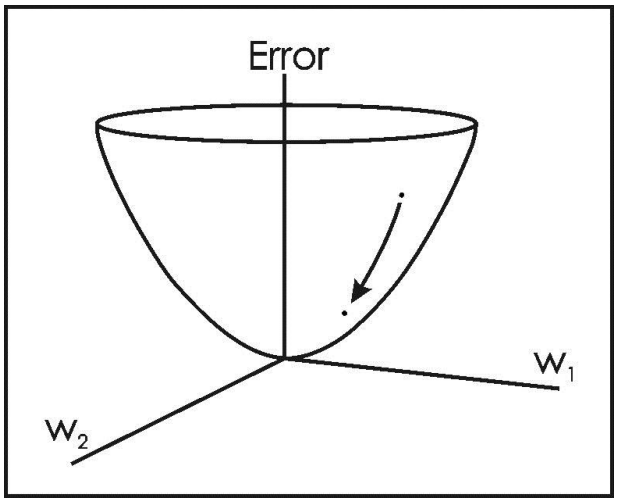

Figura 7: Explicación del error en el perceptrón multicapa

El ajuste de pesos mediante backpropagation Consiste en una corrección de las matrices de pesos propagando el error desde la capa de salida hasta la de entrada en 2 fases. El conjunto de entrenamiento es:

$$C = \left\{\left(\overrightarrow{X_p}, \overrightarrow{Y_p}\right) \middle| \overrightarrow{X_p} \in R^N, \overrightarrow{Y_p} \in R^M, p = 1,2,\dots,k\right\}$$

La primera fase es la actualización de la matriz de pesos de la capa de salida (pesos que unen la capa de salida con la intermedia).

Está basado en la minimización del Error Cuadrático Medio (similar a la regla delta):

- Los pesos de la matriz se modifican:

$$\overrightarrow{W_{t+1}} = \overrightarrow{W_t} + \overrightarrow{\Delta W}$$

- Proporcionalmente en dirección opuesta al gradiente, siendo n el factor que marca esa proporcionalidad:

$$\overrightarrow{\Delta W} = -\eta \nabla E^p(\overrightarrow{W_t})$$

- Como E depende de W_t a través del producto escalar $S = \sum_{i=1}^{n} w_i x_i$

- Se puede aplicar la regla de la cadena a la definición de gradiente:

$$\frac{\partial E^p}{\partial \overrightarrow{W_t}} = \frac{\partial E^p}{\partial S^p} \frac{\partial S^p}{\partial \overrightarrow{W_t}}$$

  - Que para la neurona O_o de la capa de salida se convierte en:

$$\frac{\partial E^p}{\partial w_{ho}^t} = \frac{\partial E^p}{\partial s_o^p} \frac{\partial s_o^p}{\partial w_{ho}^t}$$

- El problema de resolver esta ecuación son los mínimos locales:
  - Complejidad de la estructura
  - Alto número de mínimos locales en la superficie de error
  - Hay que realizar un proceso iterativo

- S^p_o es la entrada de la neurona implicada de la capa de salida.
  - El peso W_{ho} solo influye en la neurona h_h por lo que las demás derivadas parciales son 0 y el resultado es justamente la salida de la neurona oculta de la que parte la conexión:

$$\frac{\partial S_o^p}{\partial w_{ho}^t} = \frac{\partial}{\partial w_{ho}^t}\left(\sum_{j=1}^{l} w_{jo} h_j^p\right) = f'(S) = h_h^p$$

- El término $-\frac{\vartheta E^p}{\vartheta S_o^p}$ (no lineal) se define como δ_o^p o sensibilidad de la neurona de salida o-ésima ante el patrón p:

$$\delta_o^p = -\frac{\partial E^p}{\partial S_o^p} = (d^p - o_o^p) F_o'(S_o^p)$$

- Con lo que ΔW_{ho} sería:

$$\Delta W_{ho} = -\eta \frac{\partial E^p}{\partial S_o^p} \frac{\partial S_o^p}{\partial w_{ho}^t} = \eta \delta_o^p h_h^p$$

- De esta forma, la ley para modificar el peso que une una neurona h de la capa oculta con una neurona de la capa de salida es:

$$w_{ho}^{t+1} = w_{ho}^t + \eta \delta_O^p h_h^p$$

- Donde:
  - $\mathbf{h^p_h}$ es la salida (activación) de la neurona de la capa oculta que a su vez es entrada a la neurona de la capa de salida.
  - $\mathbf{S^p_o}$ es la sensibilidad de la neurona de la capa de salida
  - **n** es el coeficiente de aprendizaje
    - Controla cuanto se desplazan los valores en dirección negativa al gradiente

- Influye en la velocidad de convergencia del algoritmo y en su estabilidad (problemas de mínimos locales)
- Valores altos favorecen convergencia rápida pero existe el riesgo de oscilar
- Valores bajos hacen convergencia lenta pero evitan dicho problema

La segunda fase es la actualización de la matriz de pesos de la capa oculta (pesos que unen la capa de entrada con la capa oculta). El problema es que no se conoce de antemano la salida correcta de esta capa oculta. Se trata por tanto de ir hacia atrás, calculando el error de una neurona de la capa oculta a partir del error de las neuronas de la capa de salida:

- Se utiliza una transformación del error generado por la capa de salida, retropropagándolo a la capa oculta.
- La matriz de pesos W_{ih} es actualizada con una fórmula similar a la obtenida para la matriz de pesos de salida W_{ho}

Intuitivamente:

- Cada neurona **h** es responsable del error que tiene cada una de las unidades a las que envía su salida y lo es en la medida que marca el peso e la conexión entre ellas.
- A su vez, cada neurona de salida **o** distribuye hacia atrás su valor de **SIGMA** a todas las neuronas ocultas que se conectan a ella.

Se parte de: $$\Delta W_{ih} = -\eta \frac{\partial E^p}{\partial S_h^p} \frac{\partial S_h^p}{\partial w^t{}_{ih}}$$

Para el cálculo de la sensibilidad, a diferencia del caso anterior, el peso W_{ih} influye en TODAS las salidas de la red por lo que la sensibilidad tiene más términos:

$$\delta_h^p = -\frac{\partial E^p}{\partial S_h^p} = F_h{}'(S_h^p) \sum_{k=1}^{l} W_{ik} \delta^p{}_k$$

S_h^p es la entrada de la neurona implicada de la capa oculta:

- El peso W_{ih} solo influye en la neurona O_o por lo que las demás derivadas parciales son 0 y el resultado es justamente la salida de la neurona de la capa de entrada, que vimos que coincide con la entrada X_i^p la que parte la conexión:

$$\frac{\partial S_h^p}{\partial w_{ih}^t} = f'(S) = x_i^p$$

Por tanto:

$$\Delta W_{ih} = \eta \delta_h^p x_h^p \qquad w_{ih}^{t+1} = w_{ih}^t + \eta \delta_h^p x_i^p$$

La actualización de cada peso de la capa oculta depende de todos los errores de la capa de salida, por lo que el error se retropropaga hasta la capa de entrada.

Aunque pueda parecer la misma fórmula de actualización de pesos, no es así, ya que:

- Los valores n son distintos para cada matriz de pesos y se ajustan por separado para dar los resultados óptimos.
- En la fórmula de ajuste de los pesos de la capa de salida, el valor que se utiliza como entrada (en el término de corrección) es la salida de las neuronas de la capa intermedia.
- En la fórmula de ajuste de los pesos de la capa oculta el valor que se utiliza como entrada (en el término de corrección) es la entrada a la red.

El momento mejora la convergencia del algoritmo backpropagation:

- Ayuda a mantener la dirección del descenso de gradiente al guardar la memoria de lo que pasó en el instante t-1
- Inercia del camino recorrido descendiendo el gradiente, evita oscilaciones y mejora la convergencia
- Propuesta por Rumelhart en 1986

$$\mu(w^t - w^{t-1})$$

Es en definitiva un parámetro que controla la importancia del término de inercia:

- Valor alto > 1(0.95)
- Debe reducirse a medida que avanza el entrenamiento
- Se calcula/aplica una vez que todo el conjunto de entrenamiento ha pasado por la red

Las ecuaciones definitivas, que constituyen la regla delta generalizada son:

Para la matriz de pesos oculta-salida:

$$w_{ho}^{t+1} = w_{ho}^{t} + \eta\delta_o^p h_h^p + \mu\left(w_{ho}^{t} - w_{ho}^{t-1}\right)$$

Para la matriz de pesos entrada-oculta:

$$w_{ih}^{t+1} = w_{ih}^{t} + \eta\delta_h^p x_i^p + \mu\left(w_{ih}^{t} - w_{ih}^{t-1}\right)$$

Habiendo visto la regla delta generalizada y el error cuadrático medio, conviene destacar todos los pasos del proceso de entrenamiento de la red:

1. Inicializar pesos con valores aleatorios
2. Presentar el vector de entrada X^p
3. Calcular la salida de las neuronas de la capa oculta
4. Calcular la salida de la red (neuronas de la capa de salida)
5. Aplicar la Regla Delta Generalizada
    1. Calcular la sensibilidad de las neuronas de la capa de salida

2. Calcular la sensibilidad de las neuronas de la capa oculta
3. Actualizar los pesos de la matriz de la capa de salida
4. Actualizar los pesos de la matriz de la capa oculta

6. Repetir desde 2 para todo el conjunto de entrenamiento
7. Calcular el Error
8. Repetir desde 2 si no se da la condición de parada

El **criterio de parada** se basa en:

- Numero de iteraciones prefijadas
- Cuando el error sobre el conjunto de entrenamiento se estabiliza por debajo de una cota prefijada
    - Cuando el error se estabiliza es porque los parámetros de la red no cambian apenas de una iteración a otra.

Al concluir el entrenamiento pueden quedar 3 posibles situaciones**:**

- Entrenamiento correcto**:** la red crea un modelo a partir de los datos de entrenamiento (generaliza**),** lo que permite:
    - Que la respuesta para cualquier elemento del conjunto de entrenamiento será correcta
    - Que puede resolver un patrón que no haya visto antes pero no garantiza que el error sea mínimo

  Para generalizar, se obvian los datos irrelevantes y se resaltan las similitudes generales (características de las muestras).
- Sub-entrenamiento: un entrenamiento insuficiente de la red hace que ésta no generalice.
- Sobre-entrenamiento: un exceso de entrenamiento hace que la red memorice exclusivamente los vectores de entrenamiento y tampoco generalice. Esto puede ocurrir por un exceso de neuronas en la capa oculta. Esto también se conoce como over-fitting, y será explicado más adelante.

Finalmente cabe destacar ciertos problemas que pueden surgir con el uso del algoritmo y posibles soluciones:

- Tiempo de computación necesario para el entrenamiento
- Parálisis de la red
    - La entrada a una neurona toma valores muy altos -> saturación
    - Cuanto más tiempo de entrenamiento transcurre, más lento es el aprendizaje
- Existencia de mínimos locales en la función
- Conjunto de entrenamiento
    - Debe de cubrir todo el espacio muestral
    - Tamaño aumenta de forma exponencial al número de neuronas
- Sobre-entrenamiento
- Tamaño de la red

  - Depende del problema
  - Solución al problema de los mínimos locales pero el aumento de capas internas hace que crezcan los problemas anteriores
  - No existen criterios generales para establecer el número de neuronas, ni de capas ni el valor de los parámetros

La solución a estos problemas se basa en técnicas optimizadoras del entrenamiento:

- Utilización de expertos en el problema
  - Obtención de nueva información
  - Experimentos o simulación de ejemplos
  - Elección de datos significativos o confusos, especialmente en la frontera de las regiones del espacio

- Extracción de características (datos de entrada). Sus objetivos son:
  - Disminuir el número de entradas
  - Facilita la separación de clases
  - Reduce el número de datos sin perder precisión
  - Acelera el proceso de entrenamiento

- Entrenamiento Adaptativo (Adaptive Learning Rate)
  - Aplicable a datos no estacionarios, cuando su variación es razonablemente lenta
  - Sus objetivos son:
    - Ajuste a datos no estacionarios
    - Si $\overline{\Delta E}$ mantiene el mismo signo (positivo o negativo) a lo largo de varios ciclos, aumentar el valor de n
    - Si $\overline{\Delta E}$ va alternando (entre positivo y negativo) a lo largo de varios ciclos, disminuir el valor de n.

- Uso de otras funciones no lineales de transferencia
  - Tangente Hiperbólica

- Simulated Annealing
  - Es una técnica basada en la incorporación de ruido aleatorio al sistema
  - Evita posibles mínimos locales de la superficie de error

3.2 REDES DE KOHONEN

SOM (Self Organizing Map, Mapa Auto-organizativo, Redes Auto-organizadas o Red de Kohonen) es un modelo neuronal de aprendizaje no supervisado basado en los trabajos desarrollados por Teuvo Kohonen en la Universidad Tecnológica de Helsinki (Finlandia), entre 1979 y 1982(13).

Los modelos desarrollados por Kohonen están relacionados con los estudios anteriores de Willshaw y von der Malsburg. Ya en 1973 von der Malsburg enunció los primeros conceptos relacionados con lo que él llamó *aprendizaje competitivo*(14), más tarde perfeccionado por Kohonen. Von der Malsburg demostró que este tipo de aprendizaje es adecuado para problemas en los que es necesario detectar regularidades y características estadísticamente sobresalientes en un conjunto de patrones de entrada.

En sus primeros estudios, Kohonen definía este tipo de sistemas como:

- Una matriz de elementos de proceso que recibe entradas coherentes de un espacio de sucesos evaluando funciones discriminantes simples a partir de ellas.
- Un mecanismo que compara las funciones discriminantes y selecciona la unidad con el mayor valor de la función.
- Algún tipo de interacción local que active simultáneamente la unidad seleccionada y sus vecinas.
- Un proceso adaptativo en virtud del cual los parámetros de las unidades activas aumenten sus funciones discriminantes en relación a la entrada actual.

La capa de Kohonen en una red SOM actúa de forma similar a los sistemas biológicos, de forma que preserva el orden y compacta la representación de una nube dispersa de datos de un espacio n-dimensional proyectándola sobre un mapa bidimensional. Al preservar el orden, dos patrones cercanos en el espacio inicial producen salidas cercanas en el mapa bidimensional. Existen evidencias de que el cerebro se organiza en ciertas áreas de forma que las informaciones captadas del entorno se representan en forma de mapas bidimensionales. Un ejemplo muy claro de esto lo constituye el *cortex visual*. Topológicamente, la idea fundamental de los sistemas auto-organizativos consiste en que unidades cercanas responden de una forma similar.

Las características fundamentales que diferencian este algoritmo del resto de las redes neuronales son su arquitectura, su procesamiento y la ley de aprendizaje. Estas características le proporcionan dos claras ventajas:

- Rapidez: un SOM es muy rápido incluso durante la fase de entrenamiento.
- Aprendizaje no supervisado: se puede aplicar en casos en que los ejemplos sólo constan de la entrada y no se conoce la salida esperada para ella.

Las aplicaciones de los mapas auto-asociativos son muy variadas. Pueden ser utilizadas en clasificación, ya que gracias a sus características son especialmente eficientes para representar las estructuras jerárquicas (orden) de espacios de dimensión mayor en un mapa de características en dos dimensiones. También pueden ser utilizados en predicción, haciendo que la salida de la red de Kohonen sea la entrada de dos nuevas capas (oculta y salida) que utilizan el algoritmo de aprendizaje de backpropagation. De esta forma, existen dos etapas: primero se entrena la red de Kohonen utilizando un algoritmo no supervisado, y después se entrena el resto de la red con un algoritmo supervisado.

Una de las aplicaciones más destacadas es el *dictáfono de Kohonen*. Kohonen desarrolló este sistema como aplicación de las redes auto-organizadas(15) para llevar a cabo el reconocimiento automático del habla y la trascripción de habla continua a una secuencia de fonemas. Una vez generada y entrenada, la red es capaz de generar texto escrito a partir de fonemas contenidos en una cinta grabada en voz (es necesario un pre-procesamiento de los datos sonoros). Por cada fonema emitido, se activa una neurona de la red asociada a ese fonema).

En general las redes de Kohonen han sido utilizadas con éxito en aplicaciones como:

- Reconocimiento del habla.
- Identificación de imágenes.
- Control de robots.
- Problemas de optimización.
- Reconocimiento de objetos.

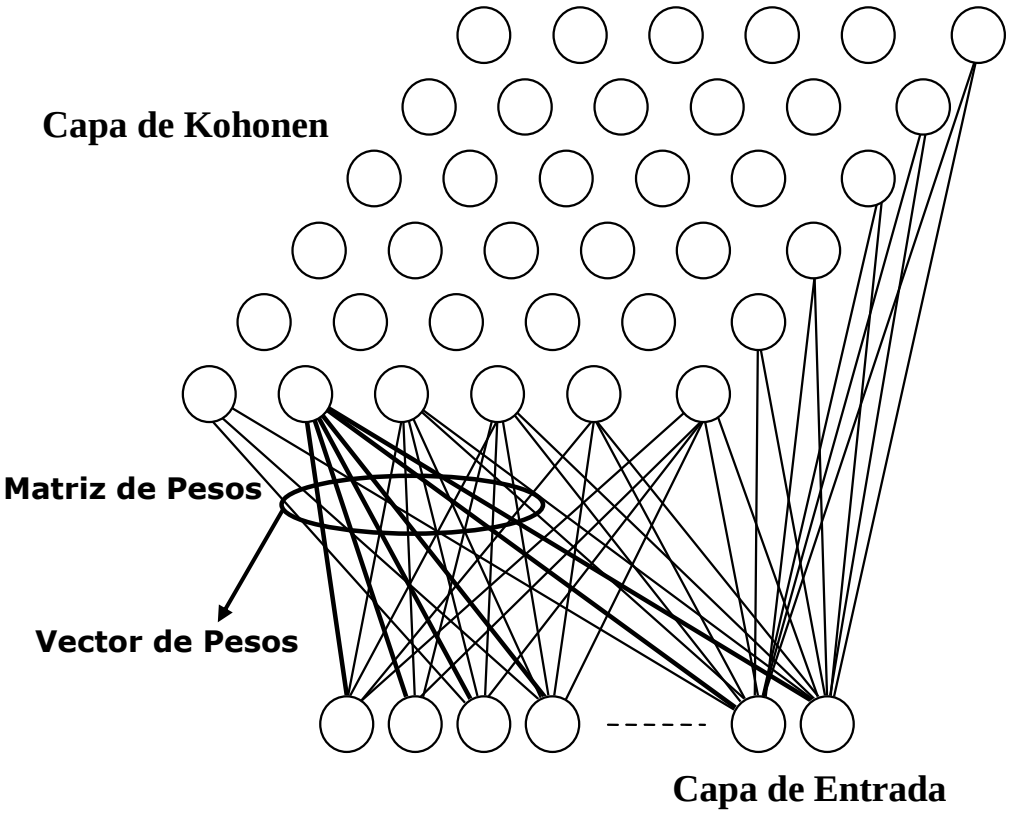

Figura 8: Representación de una red de Kohonen

3.2.1 CARACTERÍSTICAS DE UN MAPA AUTOASOCIATIVO

ARQUITECTURA DE UNA RED DE KOHONEN

La arquitectura de los modelos auto-organizativos es distinta a la de las redes de tipo backpropagation. Su estructura no es un sistema jerárquico de capas unidimensionales, sino que está basada en una capa multidimensional, por lo general bidimensional, completamente interconectada, además existen las correspondientes capas que sirven de conexión con el exterior, para el intercambio de información con el sistema en el que se integra el mapa auto-asociativo.

La arquitectura de este tipo de redes neuronales está compuesta, en el modelo básico, por una capa de entrada y una capa de Kohonen. La capa de entrada contiene tantas neuronas como componentes tengan los vectores de los patrones presentados a la red. La capa de Kohonen está compuesta por una malla cuadrada (o rectangular) de neuronas completamente interconectadas, estas conexiones determinan la activación de una neurona y la inhibición del resto, ante un patrón determinado, permitiendo que cada patrón presentado se asocie a una única neurona de esta capa.

PROCESAMIENTO

Sea una Red de Kohonen donde

- N el tamaño de la capa de entrada,
- K el tamaño del lado de la capa de Kohonen
- $X=(x_1,x_2, ..., x_N)$ el vector de entradas
- $W_j=(w_{1j}, w_{2j}, ..., w_{Nj})$ el vector de pesos de las conexiones de las neuronas de la capa de entrada con la neurona j de la capa de Kohonen

El procesamiento de la red se realiza en modo “feedforward” y por competición en tres etapas que se describen a continuación:

1. Capa de Entrada: se asigna el valor de cada una de las componentes del vector de entrada a la neurona correspondiente de esta capa.
2. Capa de Kohonen: el valor de entrada de una neurona se calcula multiplicando el valor de cada una de las neuronas de la capa de entrada por el valor de los pesos que las unen con la neurona de la capa de Kohonen y realizando la suma, sobre todas las componentes, de este producto. Esta operación equivale a realizar el producto escalar del vector de entradas por el vector de pesos que une cada neurona con las de la capa de entrada:

$$E = XW_j = \sum_{i=1}^{N} x_j . w_{ij}$$

$$O_j = \begin{cases} 1 & E_j = \max\limits_{i=1..K^2} E_i \\ 0 & E_j \neq \max\limits_{i=1..K^2} E_i \end{cases}$$

Este procesamiento equivale, siempre que los vectores de los patrones de entrada estén normalizados con la norma euclídea, al cálculo de la distancia euclídea entre el vector de

entradas y el vector de pesos asociado a cada neurona de la capa de Kohonen, activándose exclusivamente aquella cuyo valor sea el mínimo:

$$D_j = \|X - W_j\| = \sqrt{\sum_{i=1}^{N} (x_j - w_{ij})^2}$$

APRENDIZAJE

El proceso de entrenamiento/aprendizaje consiste en la adaptación de los pesos de las neuronas de la capa de Kohonen a medida que se presentan vectores de entrada correspondientes a las muestras.

El aprendizaje en una red de Kohonen se realiza de forma *no supervisada,* es decir, durante la etapa de *entrenamiento* la red dispone exclusivamente de información sobre los patrones de entrada, sin conocer la salida que se espera para cada una de ellas. Este tipo de aprendizaje obliga a la red a auto-organizarse de forma que sea capaz de detectar las características de los datos de entrada que le han sido presentados durante el entrenamiento y la forma en que estos se distribuyen en el espacio N-dimensional al que pertenecen.

Este proceso se denomina *clustering.* La red debe aprender mediante mecanismos de estímulo-reacción, de forma similar a como el ser humano aprende a hablar. El tipo de aprendizaje propuesto por Kohonen conserva características con clara inspiración biológica, pero simplificadas para que sean computables.

El objetivo del entrenamiento de un mapa auto-organizativo consiste en que se compacten los conjuntos de neuronas que responden de forma similar a ciertas entradas. Al comienzo estos conjuntos se encuentran dispersos por la estructura de la red, de forma que no están completamente definidos. Al final, se podrán diferenciar claramente unos de otros, dependiendo siempre del grado de dispersión de las muestras en el espacio original y del tamaño de la capa bidimensional definida.

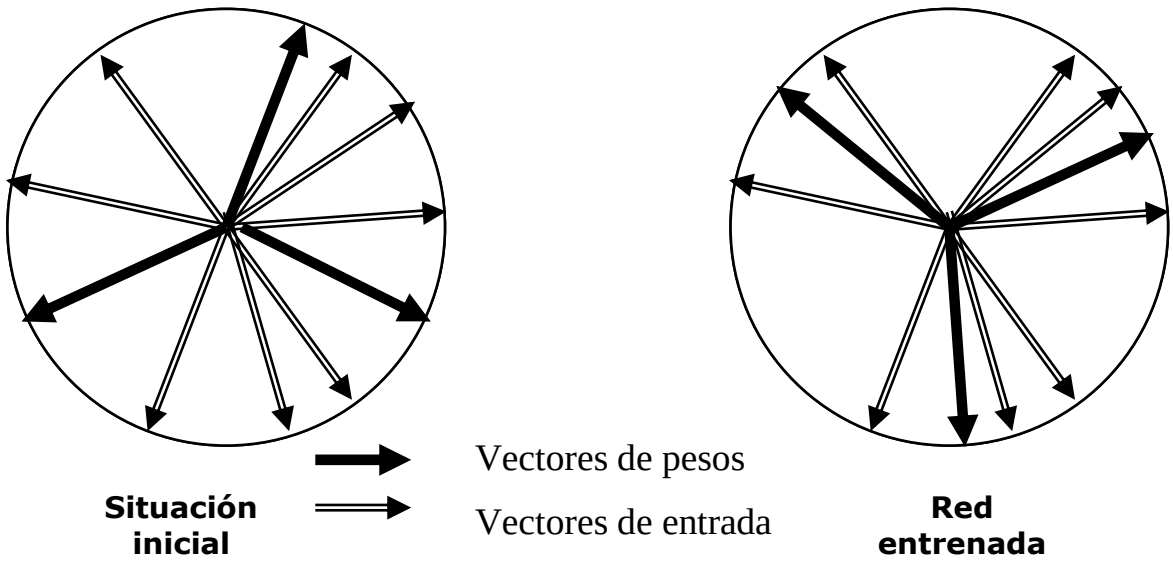

Figura 9: Modelo de aprendizaje en un SOM

Desde un punto de vista matemático, la función de densidad puntual de los vectores de pesos tiende a aproximar la función de densidad (probabilística) de los vectores de entrada. Además, los vectores de pesos tienden a ordenarse respecto a su similitud dentro del mapa bidimensional.

PROCESO

Los parámetros que rigen este modelo de entrenamiento son:

- Coeficiente de aprendizaje: η_0, η_f
- Periodo: p
- Vecindario: V_0
- Número de neuronas del lado de la capa de Kohonen: *n*

Inicialmente los pesos que conectan la capa de entrada con la capa de Kohonen tienen valores aleatorios, de forma que los vectores de pesos resultantes estén normalizados:

$$w_{ij} = \frac{w'_{ij}}{\left[\sum_{i=0}^{N} (w'_{ij})^2\right]^{\frac{1}{2}}}$$

Durante la etapa de aprendizaje se presentan uno a uno los patrones del conjunto de entrenamiento, también normalizados. Llamaremos "*presentación"* al hecho de aplicar un vector patrón a la entrada de la red. El número de presentaciones alcanza un valor prefijado llamado *periodo (p)*. En cada una de estas presentaciones se procesa la entrada obteniendo la activación de una única neurona en la capa de Kohonen. Esta neurona se denomina *neurona ganadora.*

Una vez hallada la neurona ganadora para un patrón de entrada, se determinan las neuronas que se encuentran en su *vecindario* que serán todas aquellas que se encuentren en su entorno. Normalmente se define el *vecindario* de una neurona como un cuadrado alrededor de ella, pero también pueden utilizarse otras formas como rombos, etc.

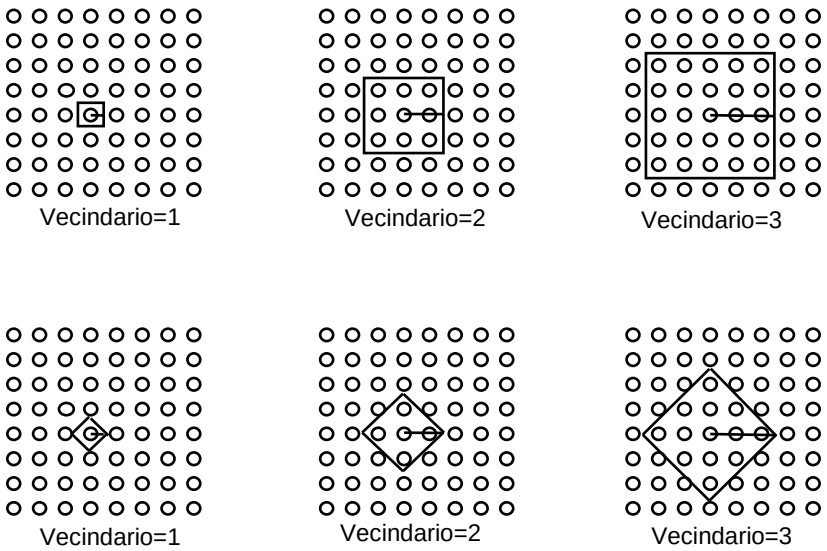

Figura 10: Tipos de vecindarios

Todas las neuronas que se encuentran en el *vecindario* de la ganadora, incluida ésta, aprenden el patrón de entrada que se presenta a la red. Esto se realiza aproximando los vectores de pesos de cada una de estas neuronas de la capa de Kohonen al vector del patrón de entrada según un *coeficiente de aprendizaje* η.

La modificación del valor de los pesos para cada neurona se lleva a cabo según la expresión:

$$W_j(t+1) = W_j(t) + \eta(t)\left[X(t) - W_j(t)\right]$$

$$w_{ij}(t+1) = w_{ij}(t) + \eta(t)\left[x_i(t) - w_{ij}(t)\right]$$

Como se ha señalado, la neurona ganadora es la que tiene el vector de pesos más parecido al patrón de entrada (considerando la distancia entre vectores). Por tanto la diferencia entre estos dos vectores es pequeña para ella. Sin embargo, la diferencia es mayor para las vecinas por lo que al aplicarles el mismo coeficiente que a la ganadora, el desplazamiento que sufren es mayor, tendiendo a reconocer con más eficacia el mismo tipo de patrones. De esta forma se consigue que la red agrupe sus neuronas en *mapas topológicos,* cada uno de ellos especializado en el reconocimiento de un tipo de patrón.

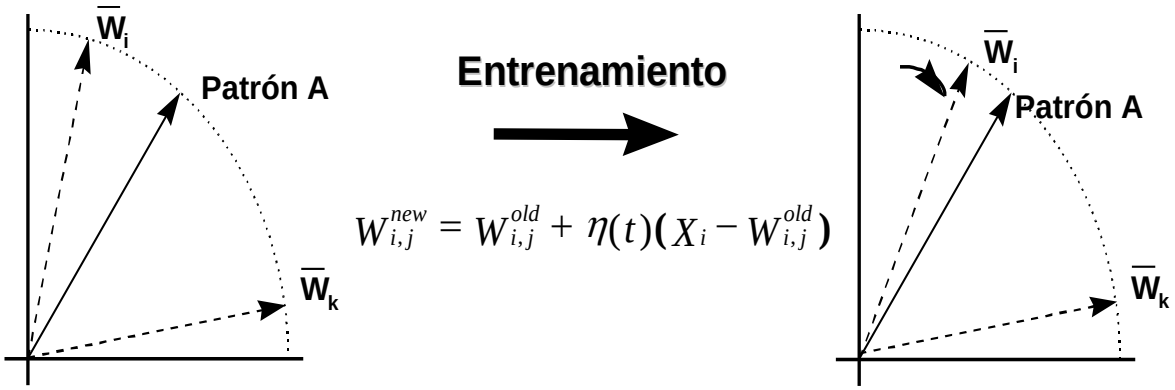

Figura 11: Modificación de la matriz de pesos

La función de aprendizaje descrita inicialmente por Kohonen, denominada sombrero mejicano**(16)**, está inspirada en el modelo biológico, en ella el coeficiente de aprendizaje depende no sólo del tiempo, sino también de la distancia a la neurona ganadora. Esta función asigna un valor mayor al coeficiente de aprendizaje de la neurona ganadora, decreciéndolo con la distancia, dentro del vecindario, hasta llegar a valores negativos cerca de la frontera de éste, siendo nulo para el resto de las neuronas de la capa. El principio es:

- Neuronas más cercanas: aprenden positivamente
- Neuronas menos cercanas: aprenden negativamente
- Neuronas alejadas: no modifican sus pesos

Debido a su definición, el uso de esta función incrementa la complejidad del algoritmo, por lo que usualmente se aproxima por otras más sencillas, la más utilizada es la función, denominada de chistera, en la que η depende linealmente del número de iteraciones, sin llegar a tomar valores negativos, y permanece constante en función de la distancia a la neurona ganadora.

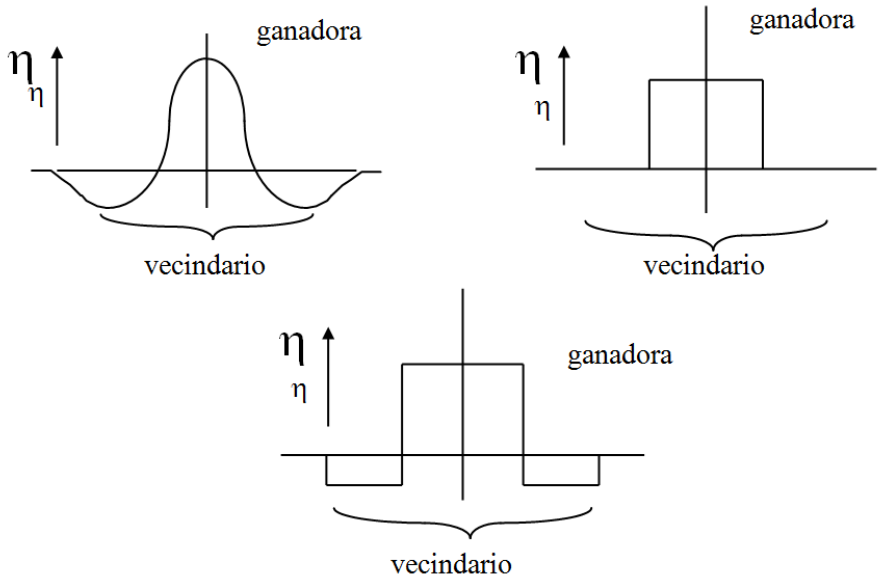

Figura 12: Funciones "sombrero mejicano", "sombrero chistera" y "sombrero de copa"

En oposición al comportamiento de otras redes, como *backpropagation*, no existe el peligro de que el tamaño del periodo de entrenamiento sea excesivo y se produzca un sobre entrenamiento *(overtraining)*(17), ya que la red de Kohonen debe converger a un estado final que representa un mapa bidimensional correspondiente a la distribución en el espacio N-dimensional de los patrones de entrada.

A medida que avanza el aprendizaje algunos de estos parámetros van variando:

- el coeficiente de aprendizaje (η) va disminuyendo
- el vecindario (v) va disminuyendo
- el número de presentaciones o "tiempo" transcurrido (t) va aumentando

La función con la que desciende el coeficiente de aprendizaje, desde su valor η_0 hasta η_f puede ser cualquier función decreciente. Las más utilizadas son de tipo lineal o exponencial.

- Descenso lineal: El entrenamiento de la red se realiza presentando los patrones del conjunto de entrenamiento repetidamente hasta que el número de presentaciones alcanza un valor prefijado llamado *periodo (p).* El aprendizaje termina cuando el coeficiente de aprendizaje alcanza su valor final. El coeficiente de aprendizaje es tal que su valor inicial es η_0 y al presentar p muestras es η_f:
- Descenso exponencial: el aprendizaje termina cuando el coeficiente de aprendizaje alcanza su valor final con un grado de aproximación

Tabla 1: Curvas y variación del proceso de aprendizaje

CURVA:	**Variación del coeficiente de aprendizaje**
LINEAL	$\eta_{(t)} = \eta_0 + \frac{\eta_f - \eta_0}{p} \cdot t$
EXPONENCIAL	$\eta_{(t)} = (\eta_0 - \eta_f) \cdot 2^{-t/p} + \eta_f$

donde

- t es el número de presentaciones transcurridas hasta el momento
- p es el período

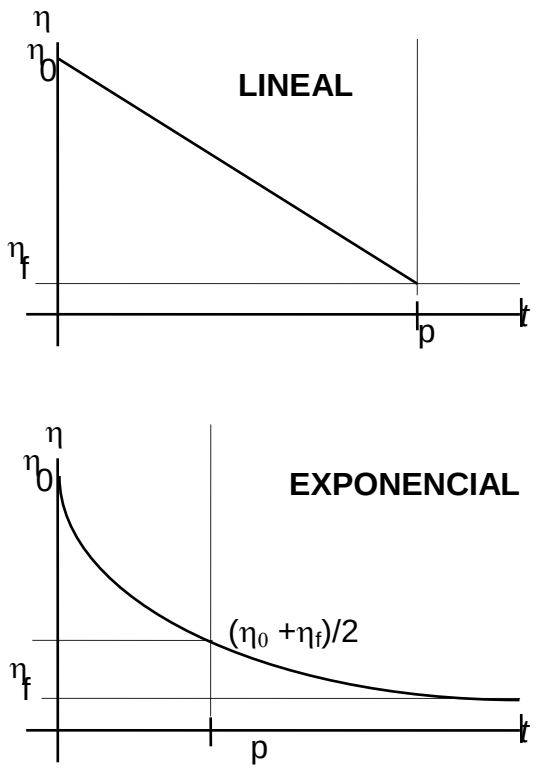

Figura 13: Curva lineal y exponencial

El tamaño del *vecindario* desciende con el número de presentaciones desde su valor inicial (v_0) hasta que abarca una sola neurona. Este descenso se lleva a cabo de forma discontinua.

A medida que se van realizando presentaciones se disminuye el valor del vecindario de neuronas cuyos pesos van a ser modificados junto con los de la ganadora. La variación del vecindario se calcula como:

$$\text{Parte entera de } \left[1 + \text{VecIni} \times \text{PPC}\right]$$

$$\text{PPC} = \left(1 - \text{PYT}\right)$$

$$\text{PYT} = \frac{\text{Numero de muestra}}{\text{Longitud de periodo}}$$

donde

- PPC es el porcentaje de período por completar expresado en tanto por uno
- PYT es el porcentaje ya transcurrido de presentaciones.

Respecto al número de presentaciones:

- Si el descenso de η se hace de forma lineal se harán tantas presentaciones como indica Periodo
- Si el descenso de η es de forma exponencial entonces se calcula el número de presentaciones como:

$$\text{Presentaciones} = \text{Periodo} \times \frac{\log\left(\eta_{inicial} - \eta_{final}\right) + 5 \cdot \log(10)}{\log(2)}$$

El período de presentaciones termina cuando el valor de η alcanza el de η_{final}, con un cierto margen de error.

3.2.2 MODIFICACIONES AL ALGORITMO BÁSICO

Sobre el algoritmo básico descrito inicialmente por Kohonen se pueden realizar una serie de modificaciones para optimizar el proceso de entrenamiento, adaptándolo a las características de cada uno de los problemas que se presentan.

REFUERZO

Esta nueva implementación describe una función de descenso del coeficiente r' que aproxima una función exponencial(18).

Después del primer periodo de aprendizaje, se realizan sucesivos períodos de refuerzo, tantos como se desee, semejantes al anterior con las siguientes modificaciones:

- Se reduce el valor de η_0, multiplicándolo por un nuevo parámetro llamado η_{shrink}
- Se amplía el valor de p, multiplicándolo por un nuevo parámetro llamado T_{factor}.

De ésta forma, en cada periodo de refuerzo se cumple que:

$$p_r = p \cdot T_{factor}^r$$

$$\eta_{o,r} = \eta_0 \cdot \eta_{shrink}^r$$

El proceso de aprendizaje termina cuando se han realizado los refuerzos deseados

Nuevos parámetros:

- Número de refuerzos: r
- Factor de extensión del periodo de aprendizaje: T_{factor}
- Factor de reducción de η: η_{shrink}

Si se van a utilizar refuerzos entonces

- el valor del atributo Periodo se multiplica por el factor de extensión del período
- se aplica el factor de compresión a $\eta_{inicial}$
- el vecindario toma el valor constante 1 (lo que quiere decir que en la fase de refuerzos sólo la neurona ganadora ve modificados sus pesos).

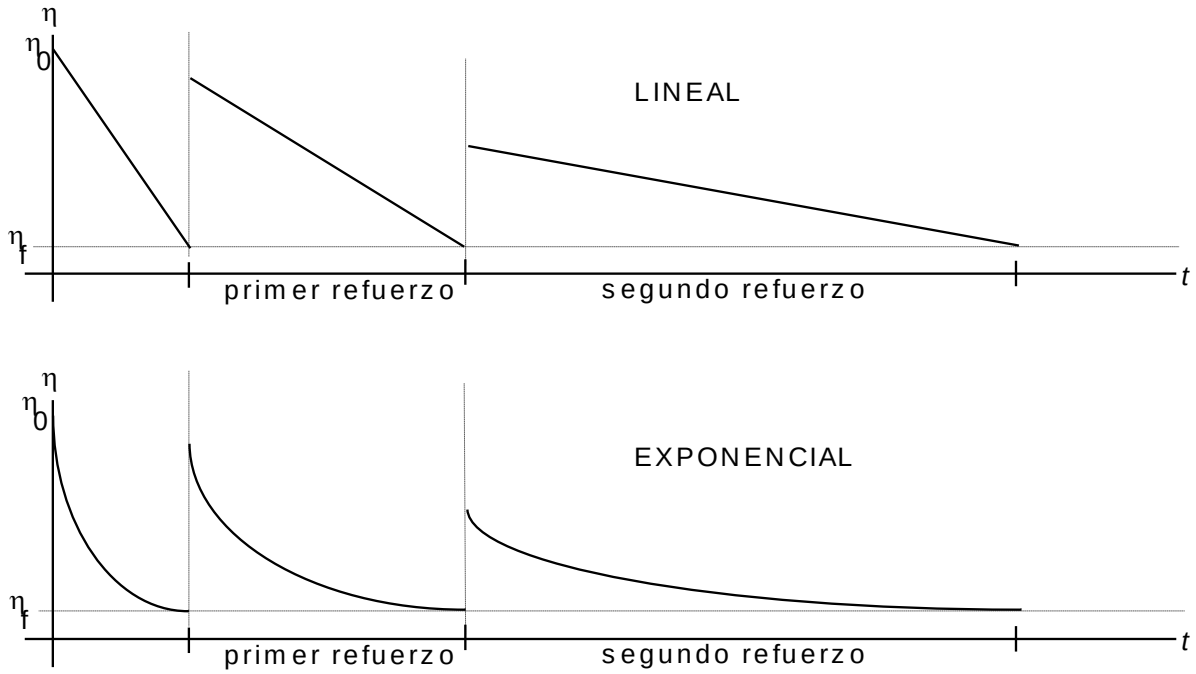

Figura 14: Refuerzos con curva lineal y exponencial

MECANISMO DE CONSCIENCIA

Este mecanismo se implementa mediante una función de penalización respecto a la frecuencia de activación. Esta función facilita que aquellas neuronas de la capa de Kohonen que no se activan ante ningún patrón de entrada tengan una mayor probabilidad de activación y que, sin embargo, aquellas que se activan con excesiva frecuencia tengan una penalización que dificulte su activación. Esto se realiza ajustando la distancia con una desviación B_i calculada a partir de la frecuencia de activación de cada neurona:

$$D'_j = D_j - B_j$$

donde

$$B_j = \gamma\left(\frac{1}{K^2} - F_j\right)$$

para cada neurona F_j es la frecuencia de activación, inicialmente tiene el valor $1/K^2$ y en sucesivas iteraciones se calcula por la siguiente fórmula:

$$F_j^{new} = \begin{cases} F_j^{old} + \beta\left(1.0 - F_j^{old}\right) & ganadora \\ F_j^{old} + \beta\left(0.0 - F_j^{old}\right) & resto \end{cases}$$

Parámetros:

- Factor de actualización de la frecuencia de activación: β_0

- Factor de penalización según la frecuencia de activación: γ_0

$$\beta = \begin{cases} tipo\_curva = lineal \Rightarrow \beta_0 \left(1 - \frac{n}{p}\right) \\ tipo\_curva = \exp onencial \Rightarrow \beta_0 2^{-n/p} \end{cases}$$

$$\gamma = \begin{cases} tipo\_curva = lineal \Rightarrow \gamma_0 \left(1 - \frac{n}{p}\right) \\ tipo\_curva = \exp onencial \Rightarrow \gamma_0 2^{-n/p} \end{cases}$$

El mecanismo de consciencia no ha sido implementado en el proyecto debido a que se sale del ámbito de la aplicación.

Las principales ventajas aportadas son:

- En lo referente al desarrollo de la aplicación, se consiguen innegables ventajas al no tener que elaborar un modelo de la serie, sino únicamente necesitar el suficiente número de datos históricos. Esto hace que las aplicaciones sean más accesibles al no requerir para su desarrollo y mantenimiento la presencia de expertos altamente especializados. Su desarrollo puede ser incremental, obteniéndose buenas predicciones incluso aunque no todas las variables explicativas hayan sido consideradas por la red.
- En cuanto a la predicción en sí misma, los niveles de precisión alcanzados son considerablemente mejores; se consiguen mayores horizontes de predicción, con costes de desarrollo menores y las aplicaciones resultantes pueden ser mejoradas continuamente a fin de adaptarlas a condiciones cambiantes de la demanda sin más que procediendo al reentrenamiento de la red neuronal con los datos más recientes.

3.3 HERRAMIENTAS DE REDES NEURONALES

En este apartado se recogen las herramientas de redes neuronales más conocidas, analizando sus características principales y ventajas. Si bien se han realizado varios experimentos en el desarrollo de software de redes neuronales, cabe destacar que no hay una gran variedad de herramientas disponibles al público en general.

3.3.1 NEUROFORECASTER

NeuroForecaster es una herramienta de redes neuronales utilizada para la predicción de eventos. Tiene una interfaz gráfica amigable y utiliza las últimas tecnologías de redes neuronales, lógica difusa y dinámica no lineal. Sus mejores aplicaciones son:

- Predicción de series temporales (como la bolsa o cambios en los valores de divisas)
- Clasificación (como la selección de acciones, evaluación de bonos de crédito, valoración de propiedades, etc.
- Indicación de análisis: identificación de buenos indicadores de las entradas.

En lo que a sus características se refiere, cabe destacar que:

- Ha sido desarrollada para Windows 32-bit
- Es utilizada para negocios en general y para predicciones financieras.
- Realiza análisis de series temporales, clasificación por secciones y análisis de indicadores
- Contiene 12 paradigmas distintos de redes neuronales
- Tiene funciones de AutoTest, AutoSave, AutoStop
- Genera hojas de cálculo compatibles con Excel para la fácil manipulación de datos
- Lee datos de Metastock, CSI, Computrac, FutureSource, Excel y formatos ASCII
- Es fácil de instalar y ejecutar
- Trae muchos ejemplos de uso

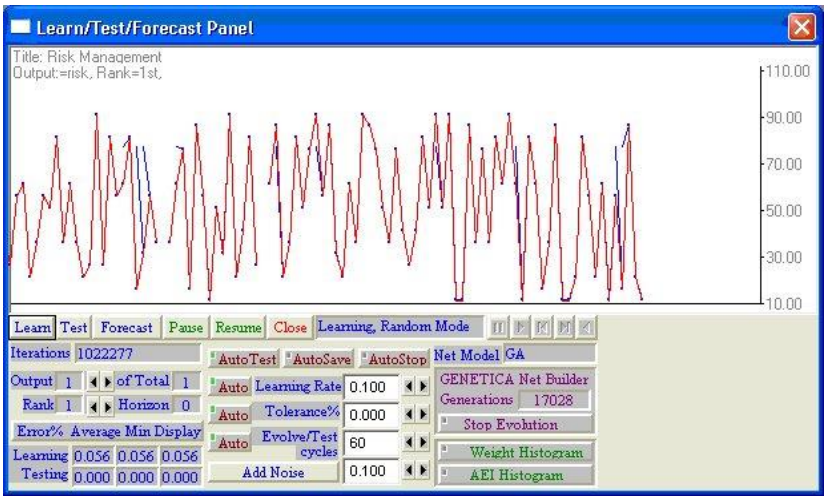

Figura 15: Ejemplo NeuroForecaster

3.3.2 MATLAB- NEURAL NETWORK TOOLBOX

El Neural Network Toolbox de MatLab provee de funciones y aplicaciones para modelar sistemas no lineales complejos que no son fácilmente modelables con otro tipo de ecuaciones. Permite el aprendizaje supervisado feed-forward, base radial y redes dinámicas. También soporta aprendizaje no supervisado con mapas auto-asociativos y capas competitivas. Con esta herramienta es posible diseñar, entrenar, visualizar y simular redes neuronales. Se puede utilizar para *data fitting*, reconocimiento de patrones, clustering, predicción de series temporales, y modelado y control de sistemas de forma dinámica.

Para acelerar el entrenamiento y gestión de grandes conjuntos de datos, es posible distribuir el cálculo en varios procesadores, GPUs, clusters de ordenadores, etc.

Dentro de sus características principales destacan:

- Redes supervisadas
- Redes no supervisadas
- Aplicaciones para el data-fitting, reconocimiento de patrones y clustering
- Soporte de computación paralela y GPU para acelerar el entrenamiento
- Preprocesamiento y postprocesamiento para mejorar la eficiencia del entrenamiento de la red y evaluar el rendimiento de la misma
- Representación modular de la red para gestionar y visualizar redes de tamaño arbitrario

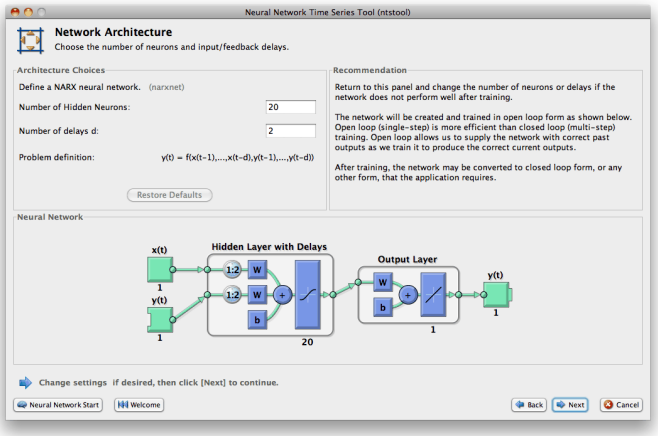

Figura 16: MatLab Neuronal Network Toolbox

3.3.3 NEUROSOLUTIONS

NeuroSolutions es una herramienta para el desarrollo de redes neuronales disponible para Microsoft Windows. Combina una interfaz gráfica modular basada en iconos cono procedimientos avanzados y optimización genética. Es utilizado para el análisis de clusters, predicción de ventas, predicción de resultados deportivos, clasificación de enfermedades, entre otros.

Sus características principales son:

- Facilidad de uso: no requiere conocimientos previos de redes neuronales y se integra fácilmente con Excel y MatLab. Incluye un asistente neuronal para que tanto los usuarios novatos como los avanzados puedan sacarle el máximo provecho.
- Diseño flexible: permite la construcción de redes neuronales completamente personalizables. pero también ofrece la posibilidad de escoger de numerosas opciones de arquitecturas de redes neuronales ya construidas. Permite modificar el número de capas ocultas, elementos de procesamiento y algoritmo de aprendizaje.
- Motor potente: ofrece un potente motor para el cálculo tanto para Windows de 32 bits como 64 bits. Para redes neuronales muy grandes, permite utilizar el procesamiento de NVIDIA CUDA y OpenCL mediante el procesador del ordenador o la tarjeta gráfica, pasando el entrenamiento a durar minutos en lugar de horas.

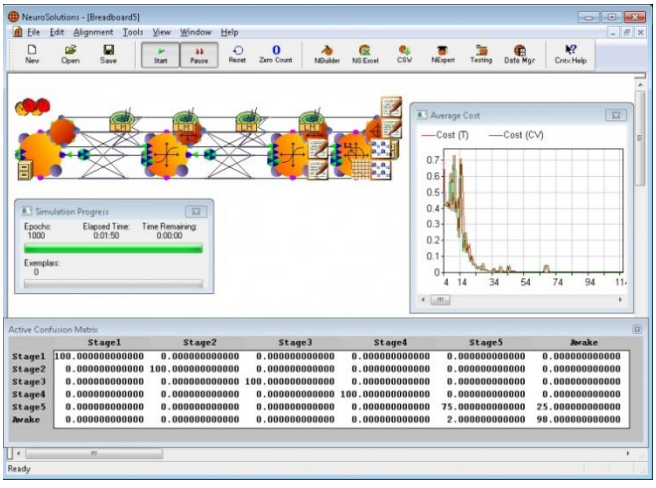

Figura 17: NeuroSolutions

3.4 VISUALIZACIÓN DE DATOS VÍA JAVASCRIPT

Los conjuntos de datos por defecto pueden resultar muy difíciles de comprender a simple vista. Incluso expertos con conocimientos técnicos avanzados pueden encontrar muy complicado extraer información útil de los datos sin algún mecanismo de ayuda. Para hacer que los datos sean fáciles de comprender y amigables a los usuarios, estos deben ser procesados y preparados. El procesamiento y visualización de datos son esenciales para facilitar la interpretación de los datos y de toda la información que se encuentran en ellos.

En este apartado se busca resumir brevemente el estado del arte en la visualización de datos en formato Web, particularmente en *Javascript*, ya que ha sido esencial para el desarrollo del proyecto.

A continuación se describen las principales herramientas disponibles para el análisis gráfico de datos y una breve descripción de las mismas en función de sus características técnicas:

3.4.1 APLICACIONES WEB PARA LA VISUALIZACIÓN

Existen muchas herramientas disponibles que ofrecen opciones de visualización. Si bien algunas son las más conocidas tablas y gráficos, muchas otras ofrecen diversas posibilidades como diagramas de árboles y nubes de palabras.

3.4.1.1 GOOGLE FUSION TABLES

Es una aplicación web para organizar, gestionar, visualizar y publicar datos en la web de una manera simple. Maneja grandes conjuntos de datos para ser estandarizados y gestionados en archivos Excel, .ods, .csv o .kml. La aplicación permite mostrar los datos utilizando gráficos de tarta, barra, dispersión de puntos, y series temporales, así como organizarlos en *Google Maps*.

Si bien las características técnicas de la herramienta se salen del ámbito del proyecto, cabe destacar que la tecnología utilizada es *Javascript y Flash*, es una aplicación tipo Web que tiene una API para conectarla con otras herramientas, y su licencia de uso es gratuita.

Toda la documentación para utilizar esta aplicación se encuentra en: https://developers.google.com/fusiontables/docs/v1/getting_started

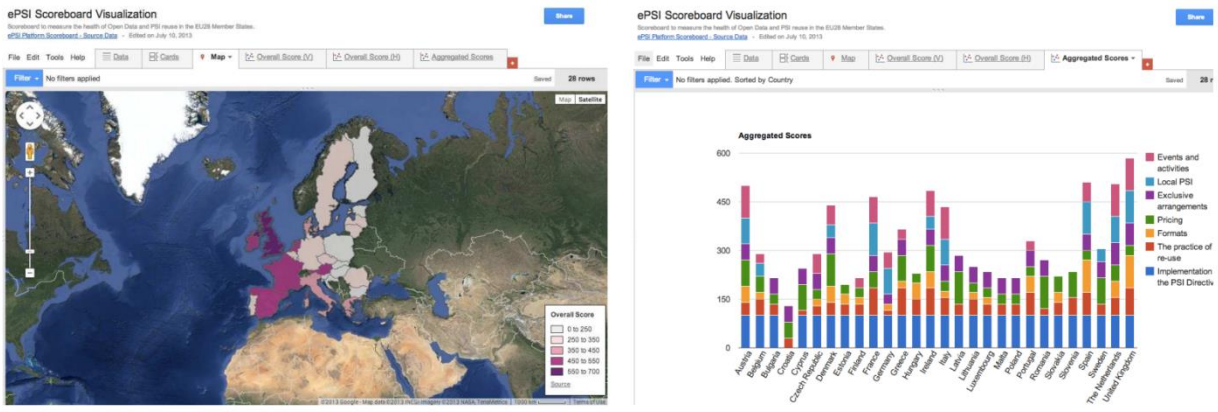

Figura 18: Ejemplo Google Fusion Tables

3.4.1.2 TABLEAU PUBLIC

Es una herramienta gratuita para la visualización de datos mediante gráficos que combinan una interfaz gráfica atractiva, rápida y eficiente con los elementos tradicionales de las herramientas de business inteligence, tales como por ejemplo el modelo organizativo de variables utilizando dimensiones y medidas, o las conexiones con otros sistemas de gestión de información (como por ejemplo las bases de datos o las hojas de cálculo). A continuación se describen las características más importantes de esta herramienta:

- Adquisición de datos de forma rápida y sencilla. Permite trabajar con bases de datos y hojas de cálculo de cualquier tamaño. Acepta formatos de Microsoft Excel, Access, y ficheros de texto plano.
- Permite trabajar con una gran variedad de gráficos: barras, tartas, mapas con polígonos, líneas, puntos, etc.
- Permite combinar diferentes fuentes de datos en una sola vista.
- Su salida son gráficos interactivos.
- Tiene repositorios públicos de datos a los que sus usuarios pueden acceder.
- Los datos "crudos" pueden ser descargado desde la pestaña de visualización.

Si bien es principalmente una herramienta de escritorio, utiliza *Javascript* como tecnología principal de visualización, algo muy peculiar puesto que suele ser sólo para herramientas Web. La licencia de uso es gratuita.

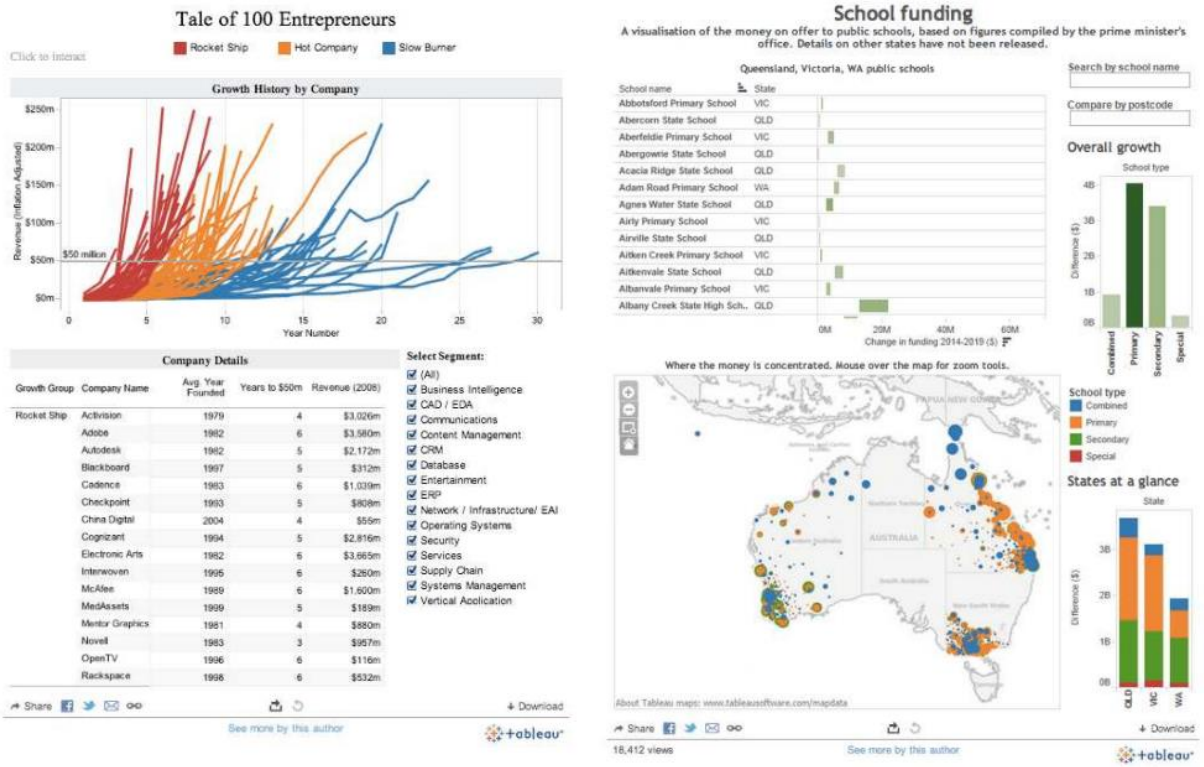

Figura 19: Visualización de datos mediante Tableau Public

3.4.1.3 MANY EYES

Es una aplicación web que permite al usuario crear, compartir y discutir la representación gráfica de los datos descargados por los usuarios. La compañía detrás de esta herramienta es IBM.

Con Many Eyes los usuarios pueden compartir sus visualizaciones, potenciando el intercambio de opiniones mediante diferentes enfoques de los mismos datos. Es un herramienta para el uso público, es decir, todos los datos y visualizaciones se hacen disponibles para todos los usuarios, y no puede ser utilizada de forma privada.

Da lugar a muchas formas de visualización:

- Relaciones entre puntos
- Comparación de valores mediante diagramas de barras, burbujas e histogramas
- Cambios temporales mediante gráficos de líneas, barras, etc.
- Partes de un todo mediante diagramas de tarta, mapa arbóreo, etc.
- Analizador de texto que genera árboles de palabras, nubes de palabras y redes de frases
- Datos geográficos sobre mapas

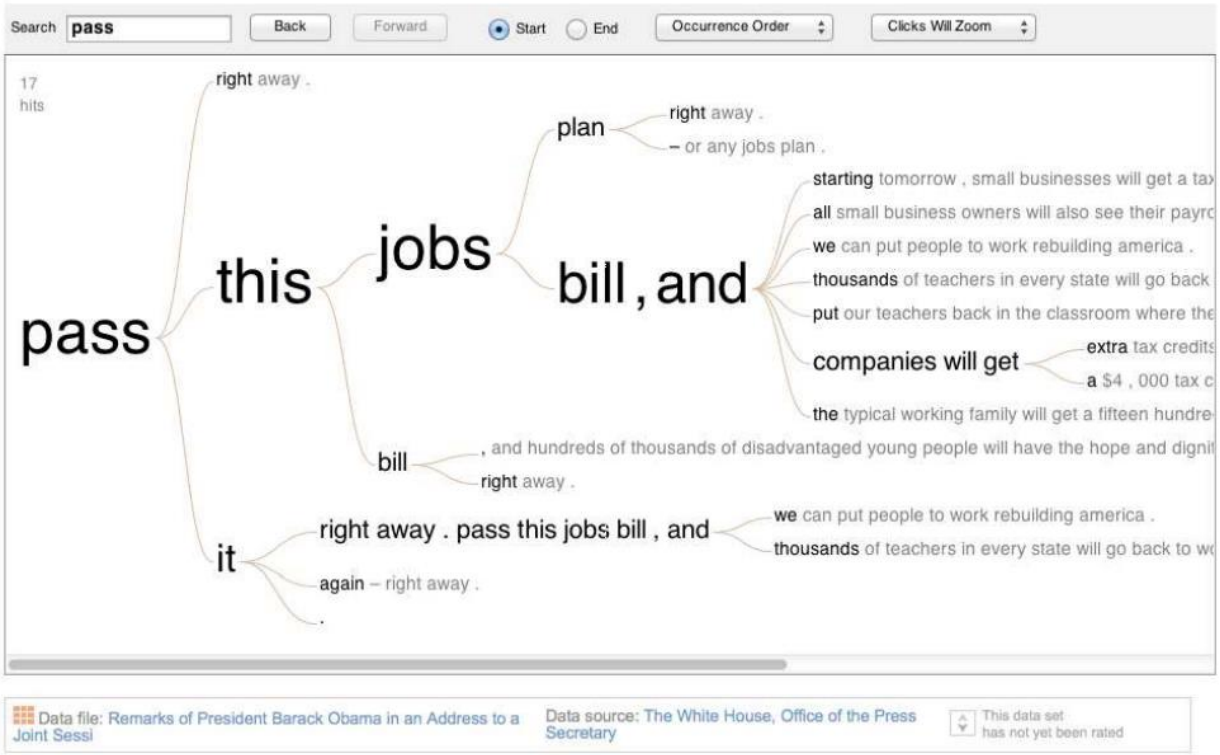

Figura 20: Ejemplo de analizador de texto con Many Eyes

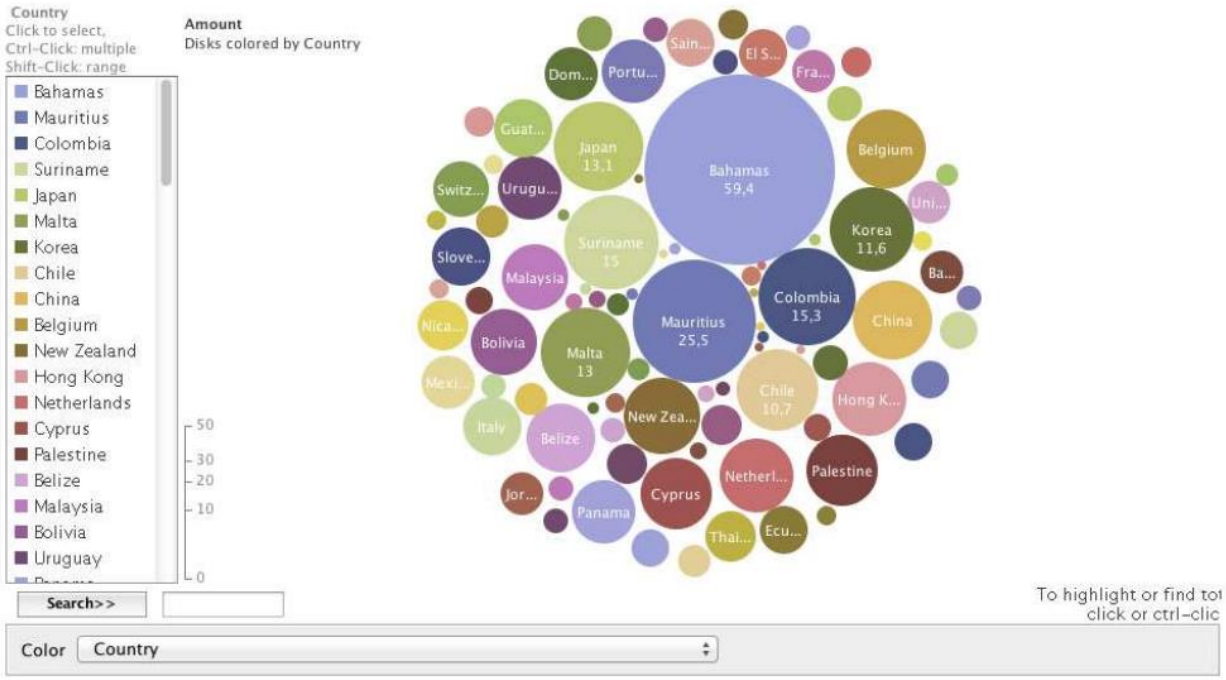

Figura 21: Ejemplo de visualización de burbuja con Many Eyes

3.4.1.4 CARTODB

Es una base de datos geoespacial alojada en la nube, bajo el servicio de *Amazon Web Services*, y permite una gran escalabilidad y flexibilidad de sus servicios. Es un proyecto open source, que también es ofrecido como servicio bajo demanda. Es una aplicación web cuya tecnología se basa en *Javascript, PostgreSQl,* y la extensión geoespacial *PostGIS*. Fue desarrollado por Vizzuality en EEUU.

CartdoDB busca facilitar el desarrollo de aplicaciones y mapas que trabajan con datos geográficos. Permite el diseño y desarrollo de mapas en tiempo real que trabajan con plataformas web y móviles universales. Entre su gran variedad de características, destacan:

- Diseño de mapas para capas de datos. Se puede utilizar *CartoCSS* para editar formatos fácilmente y la forma en la que se ven los mapas.
- Integración con otros servicios de mapas. *CartoDB* produce capas de datos que son representados en Google Maps y MapBox. Estos mapas incluyen funciones básicas como zoom, scroll, etc.
- Integración con otras librerías. CartoDB tiene muchas librerías que pueden extender su uso o ser integradas con otros servicios.
- Geocodificación: la información geográfica se puede obtener de elementos diferentes a coordenadas.
- Fácil importe de datos: CartoDB permite la introducción directa de datos y su transformación a tablas, así como añadir datos mediante comandos SQL o desde URLs. Otros conjuntos de datos pueden ser importados mediante varios formatos.

- Permite realizar queries SQL basadas en componentes espaciales. Mediante PostGIS, CartoDB puede hacer consultas y combinar conjuntos de datos utilizando datos geoespaciales.
- Tablas públicas y privadas. CartoDB permite a los usuarios definir la privacidad de sus tablas.

Está enfocada a desarrolladores sin experiencia en los sistemas de información geoespaciales, usando una interfaz gráfica amigable. Muchas instituciones prestigiosas tales como Naciones Unidas, Google, NASA, la Universidad de Oxford, Yale, entre otras, utilizan CartoDB.

Figura 22: Ejemplo CartoDB

3.4.1.5 GEOCOMMONS

Es una plataforma basada en Web para la gestión, visualización, mapeo, y análisis espacial de datos geoespaciales. También ofrece una API, y permite cargar datos de diferentes tipos de fuentes de datos: hojas de cálculo, ficheros KML, servidores web con soporte espacial, servicios OGC tales como WMS y TMS, y desde su propio repositorio público. Su tecnología está basada principalmente en Javascript y Ruby. Tiene diferentes tipos de licencia en función de la aplicación que se le vaya a dar.

Esta herramienta soporta técnicas para la representación cartográfica de mapas con símbolos personalizables en el color, tamaño, transparencia, forma, estilo de iconos y líneas, y secuencia de colores en los mapas. GeoCommons también incluye la posibilidad de realizar animaciones. Los mapas pueden ser exportados al formato KML, los datos al formato KML, hojas de cálculo, ficheros de formas, entre otros. Por último, destacar que los mapas pueden ser introducidos en otras páginas web.

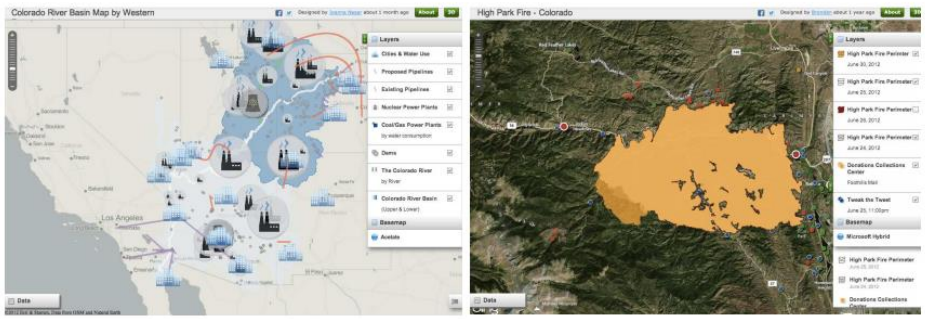

Figura 23: Ejemplo de mapas con GeoCommons

3.4.2 LIBRERÍAS Y APIS PARA LA VISUALIZACIÓN WEB

Además de las herramientas existentes, los desarrolladores Web disponen de una gran cantidad de librerías y APIS para generar sus propias herramientas de visualización, tal y como se ha hecho en este proyecto.

3.4.2.1 GOOGLE CHART TOOLS

Es una librería basada en tecnología Javascript que permite a los desarrolladores de Google generar gráficos de imágenes en PNG. Su operación está basada en llamadas HTTP a una URL específica (http://chart.apis.google.com).

Existe una gran cantidad de gráficos que se ofrecen como clases Javascript diferentes. Una de las mayores ventajas de este sistema de generación de gráficos es que los usuarios no necesitan instalar ningún componente o entorno ni un servidor, de forma que cada gráfico se pueda generar al instante y sin complicaciones.

El enlace con toda la documentación técnica de esta librería se encuentra disponible en el enlace: https://google-developers.appspot.com/chart/interactive/docs/reference

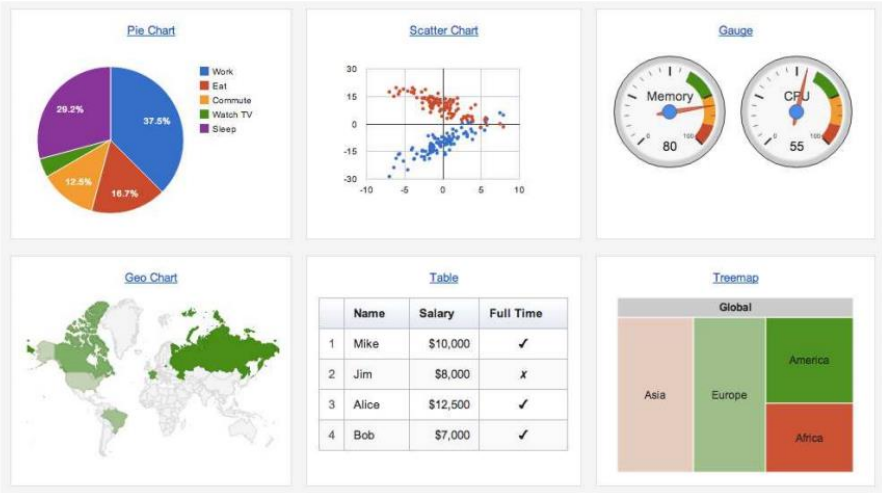

Figura 24: Posibilidades de gráficos mediante Google Chart Tools

3.4.2.2 JAVASCRIPT INFOVIS TOOLKIT

Es una librería basada en tecnología Javascript y Python que funciona bajo la licencia MIT. De lugar a herramientas que permiten crear visualizaciones interactivas en aplicaciones web (mapas estratégicos, árboles jerárquicos, mapas relacionales, etc.). Debido a su extensiva variedad de representaciones, es una herramienta utilizada por muchos desarrolladores.

Algunas de las características más relevantes de esta librería son:

- Diferentes tipo de representación de datos
- Interacción con los datos en tiempo real
- Compatibilidad con la mayoría de exploradores.

- Open Source, lo que facilita su integración futura con otras herramientas.
- Es muy extensible.
- Combina varios tipos de visualización para generar nuevas formas de representación.
- Velocidad de procesamiento rápida para estructuras complejas.

Desde el punto de vista técnico, cabe destacar que al igual que muchas otras librerías Javascript, la representación de los datos utiliza el formato de ficheros JSON (uno también utilizado en este proyecto). Es un formato de poco peso basado en 2 estructuras: una colección de pares de nombres y valores asociados (objetos, record, estructuras, diccionario, etc.), y una lista ordenada de valores (arrays, listas o secuencias). Estas estructuras universales de JSON permiten que todos los lenguajes de programación que las utilicen se puedan adaptar de una forma sencilla.

Las aplicaciones que tiene la librería mencionada son:

- Desarrollo en entornos de Business Intelligence.
- Gráficos organizacionales
- Mapas estratégicos
- Mapas de datos estadísticos
- Mapas relacionales

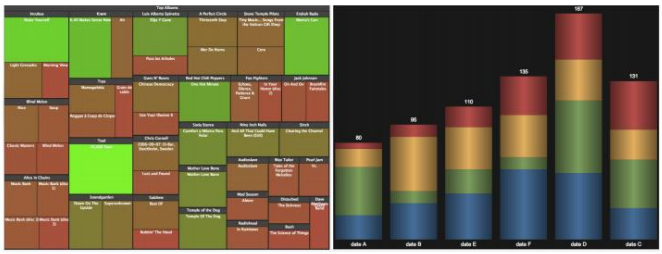

Figura 25: Ejemplos de visualización con InfoVis

3.4.2.3 D3.JS

Es una librería basada en tecnología Javascript utilizada para crear múltiples tipos de visualización y gráficos interactivos. Básicamente permite a los usuarios manipular documentos basados en datos utilizando los estándares de la web. Los exploradores pueden renderizar estas complejas visualizaciones sin necesidad de trabajar con software propietario. Los desarrollos son abiertos y pueden ser modificados por otros usuarios. Las posibilidades son enormes debido a todos los elementos geométricos que pueden ser utilizados (burbujas, diagramas, nodos, enlaces, etc.).

D3 permite incrustar datos al DOM (Modelo de objetos para la representación de documentos) y aplicar las transformaciones. Por ejemplo, puede generar una tabla HTML a partir de una serie de números, y luego utilizar los mismos datos para crear un SVG interactivo con transiciones y interacciones.

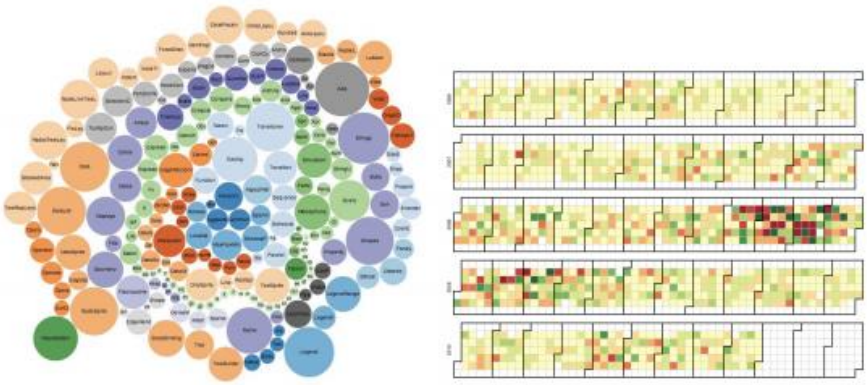

Figura 26: Ejemplo D3.JS

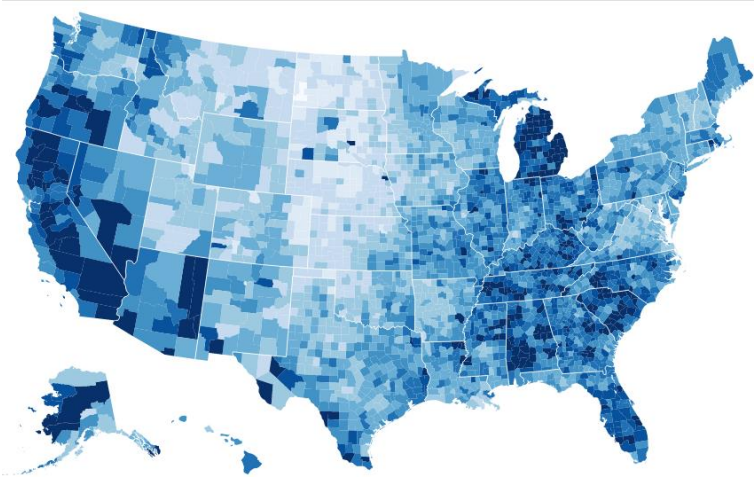

Figura 27: Distribución de datos en EEUU con D3.JS

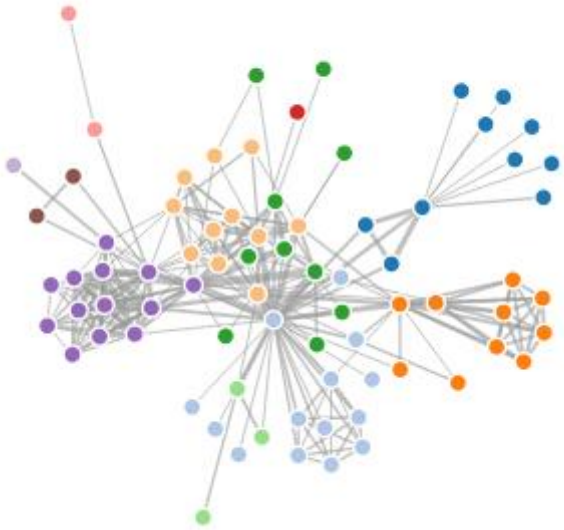

Figura 28: Visualización de conexión de nodos con D3.JS

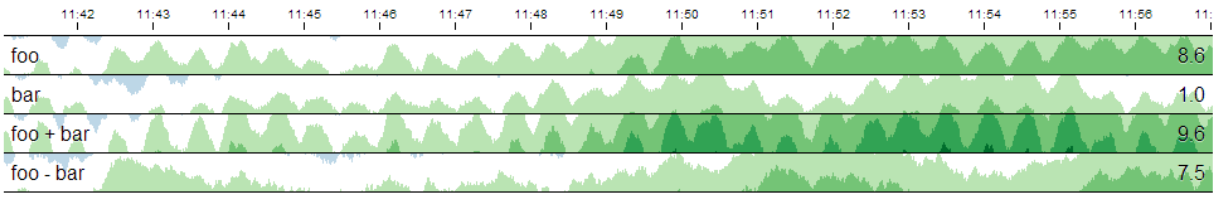

Figura 29: Visualización de series temporales con D3.JS

3.4.2.4 PROTOVIS

Es una librería de gráficos Javascript que permite la muestra de visualizaciones. Da a los desarrolladores una gran variedad de componentes y herramientas, permitiendo la personalización de los resultados.

Algunas de las características más relevantes son:

- Flexibilidad ilimitada: está basada en una gramática declarativa y un framework basado en datos.
- Configuración de gráficos sencilla, basada en el método de encadenamiento de objetos y funciones Javascript.
- Se basa en gráficos estadísticos y su método de desarrollo permite visualizaciones de datos estructurados.
- Incorpora ciertas funciones estadísticas para la preparación de datos.

Una de las principales desventajas de Protovis es lo pesada que puede ser su librería (700k Kb).

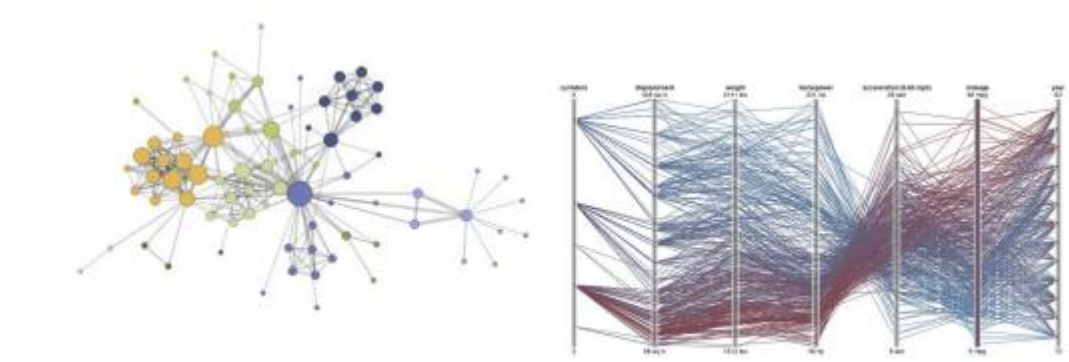

Figura 30: Ejemplo visualización Protovis

3.4.2.5 RECLINE.JS

Es una librería de gráficos Javascript diseñada para la integración en varios sitios web y aplicaciones. Está enfocada para desarrolladores con conocimientos mínimos de programación, quienes prefieren interfaces simples para visualizar y editar datos. Estas vistas están disponibles en modo gráfico, mapas, y series temporales.

Recline corre bajo *Backbone*, una estructura que provee un soporte excelente para construir aplicaciones que pueden manejar grandes cargas de datos, utilizando modelos para la gestión de la información y vistas para mostrarlos. A su vez, es fácilmente extensible mediante conexiones a bases de datos.

Esta librería dispone de muchas capacidades para la manipulación de bases de datos, incluyendo la inserción, búsqueda y actualización. Permite la carga de ficheros CSV, Excel, Google Docs, ElasticSearch, CouchDB, y DataHub, entre otros. Permite el limpiado de datos y contiene mecanismos de actualización utilizando simples scripts.

Recline está compuesta de 3 módulos:

1. Modelo: define la estructura de los datos (el data set a utilizar en base a su formato y tipo).
2. Backend: conexión de los datos utilizada por la API de Recline de forma directa a la fuente de los datos (base de datos, fichero, etc).
3. Vistas: gestión de la información obtenida en los 2 módulos anteriores.

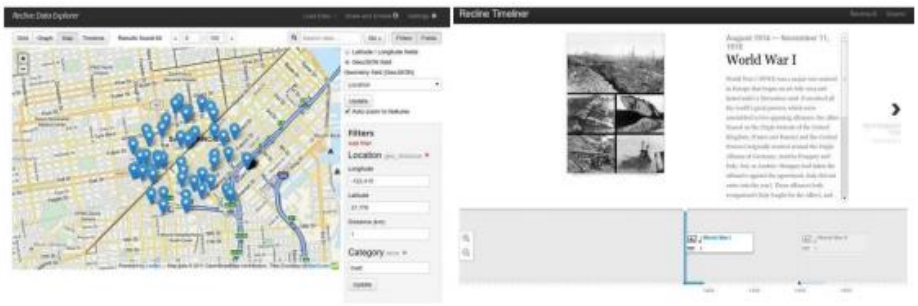

Figura 31: Ejemplo InfoVis

4. LIMITACIONES Y CONDICIONANTES

4.1 LENGUAJE DE CONSTRUCCIÓN

A la hora de plantear el proyecto, una de las mayores preocupaciones era la posibilidad de integrar una visualización muy buena de los datos manteniendo la potencia de cálculo necesaria para la exactitud de la clasificación. En ese sentido, la herramienta desarrollada ha sido programada utilizando diferentes tecnologías que se interrelacionan entre sí para dar lugar a los resultados obtenidos. Dichas tecnologías se describen a continuación:

4.1.1 FRONT-END

El front-end, es decir, toda la parte de interfaz gráfica y procesamiento en el cliente que utiliza la herramienta, se ha desarrollado mediante el lenguaje de maquetación web JSP, el cual tiene una gran similitud con HTML y de hecho al ser procesado es transformado en el mismo.

Esta maquetación mediante JSP permite el uso de ciertas etiquetas que facilitan enormemente el desarrollo, las llamadas etiquetas lógicas que permiten una comunicación mucho más sencilla con el servidor y la carga de datos e información en el explorador.

Por otro lado, en el front-end también cabe destacar el uso constante de Javascript principalmente por 2 motivos: el primero es que permite reducir cierta carga del servidor al poder realizar varios cálculos en el ordenador cliente sin tener que enviar la información al servidor. A modo de ejemplo, en la herramienta desarrollada para poder utilizar los pesos de una red ya conocidos anteriormente es necesario que los parámetros como el lado de Kohonen de la red sean correctos. Este es un cálculo realizado mediante Javascript, evitando así que haya que enviarle toda la información al servidor para que devuelva si es posible o no utilizar dichos datos. A su vez, Javacript permite ocultar y mostrar ciertos campos de los formularios sin tener que recargar la página, algo que da lugar a una mayor usabilidad en la interfaz, uno de los objetivos planteados en el proyecto.

Sin embargo, el motivo más importante por el cual se ha buscado desarrollar la herramienta con Javascript es debido a su capacidad de visualización de datos. Ya se ha expuesto en el estado del arte las enormes posibilidades que ofrece este lenguaje de programación en lo que a la visualización se refiere.

En este caso, se ha optado por el uso de la librería D3.JS ya que su potencia con el uso de nodos y detección de colisión en el explorador es muy buena y los resultados así lo han demostrado. A su vez, se partía de una experiencia previa con la librería que ha facilitado adentrarse en el desarrollo de la visualización de los resultados.

A su vez, D3.JS funciona muy bien a la hora de cargar archivos JSON, por lo que ha sido un beneficio a tomar en cuenta.

4.1.2 BACK-END

El back-end de la aplicación, es decir, todo el cálculo realizado en el servidor, ha sido desarrollado con Java. Como se ha explicado anteriormente, esto da lugar a una potencia de cálculo que muchos lenguajes de programación web no pueden alcanzar, garantizando así la fiabilidad de las predicciones y la clasificación.

A su vez, Java contiene una serie de librerías que facilitan considerablemente el manejo de archivos XML, JSON y PDF. Esto ha permitido reducir enormemente el tiempo necesario para pasar los resultados de clasificación, muestras, visualización, etc. a dichos formatos.

También cabe destacar que Java posee librerías para comprimir archivos en ZIP, lo que permite que el usuario descargue cuando desee todos sus archivos sin muchas complicaciones.

En definitiva, la combinación de Javascript con Java mediante una Java Web Application en el framework Struts, el cual facilita considerablemente la transmisión de información entre los archivos JSP y las clases Java, y a su vez otorga una serie de funcionalidades como el manejo de sesiones, han hecho esta tecnología la mejor para cumplir los objetivos propuestos.

4.2 EQUIPO FÍSICO

En lo que a desarrollo de la aplicación se refiere, no ha habido restricciones a tomar en cuenta. Sin embargo, para la ejecución final de la aplicación si ha de tenerse en cuenta:

- La necesidad de tener una red de área local para que diferentes ordenadores cliente puedan conectarse al servidor en el cual se aloja la aplicación
- El servidor donde se aloja la aplicación debe ser un Apache Tomcat.
- Las pantallas de los monitores deben tener una resolución de al menos 1280x1024px.

4.3 EQUIPO LÓGICO

Para el desarrollo de la aplicación se ha utilizado Eclipse UNO como entorno de desarrollo. También se ha necesitado de las librerías de Java iTEXT para la generación de PDF, las librerías integradas para la generación de XML y las librerías de Google para la generación de los archivos JSON.

Para la ejecución de la aplicación, será necesario tener Apache Tomcat instalado en el servidor de una red local a la que accederán el resto de ordenadores.

4.4 TIEMPO DISPONIBLE PARA EL DESARROLLO

Las siguientes fases no representan las iteraciones sino los hitos del proyecto. Se ha dejado tiempo al final considerando los posibles retrasos que pueden derivar de un proyecto software de esta índole así como para la terminar la memoria final (que se ha desarrollado en conjunto con la herramienta):

- Fase 1 (4 de febrero al 14 de febrero): estudio completo de las redes de Kohonen.
- Fase 2 (15 de febrero al 17 de febrero): establecimiento y correcto funcionamiento del entorno de desarrollo.
- Fase 3 (17 de febrero al 10 de marzo): implementación de todos los algoritmos correspondientes a las redes de Kohonen.
- Fase 4 (10 de marzo al 15 de marzo): evaluación de los algoritmos y comprobación de que los resultados sean exactos y correctos.
- Fase 5 (16 de marzo al 20 de marzo): aplicación de las correcciones oportunas en base a los resultados de las pruebas.

- Fase 6 (21 de marzo al 25 de marzo): creación del formato de archivo e implementación del módulo para guardar los resultados en el mismo.
- Fase 7 (25 de marzo al 31 de marzo): implementación de módulo de carga de datos desde archivos de las extensiones decididas.
- Fase 8 (1 abril al 5 de abril): diseño e implementación del boceto la interfaz gráfica de la aplicación en HTML y CSS.
- Fase 9 (5 de abril al 7 de abril): implementar la posibilidad de cargar los archivos externos de datos desde el explorador.
- Fase 10 (8 de abril al 13 de abril): implementar la posibilidad de modificar parámetros de ejecución de los algoritmos desde el explorador. Imprimir los resultados de los algoritmos por el explorador y generar el archivo para la descarga de dichos resultados.
- Fase 11 (14 de abril al 5 de mayo): generar la visualización de los resultados usando Javascript. Hacerlo amigable e interactivo para el usuario.
- Fase 12 (6 de mayo al 18 de mayo): generar la posibilidad de comparar los resultados obtenidos de manera visual y textual e implementarla en la interfaz gráfica.
- Fase 13 (18 de mayo al 25 de mayo): Finalización de la interfaz gráfica, pruebas finales y elaboración de un pequeño manual de instalación para su fácil uso.
- Fase 14 (25 de mayo al 31 de mayo): Documentación de la memoria final.

A continuación se muestra el diagrama de Gantt elaborado:

Figura 32: Diagrama de Gantt (1)

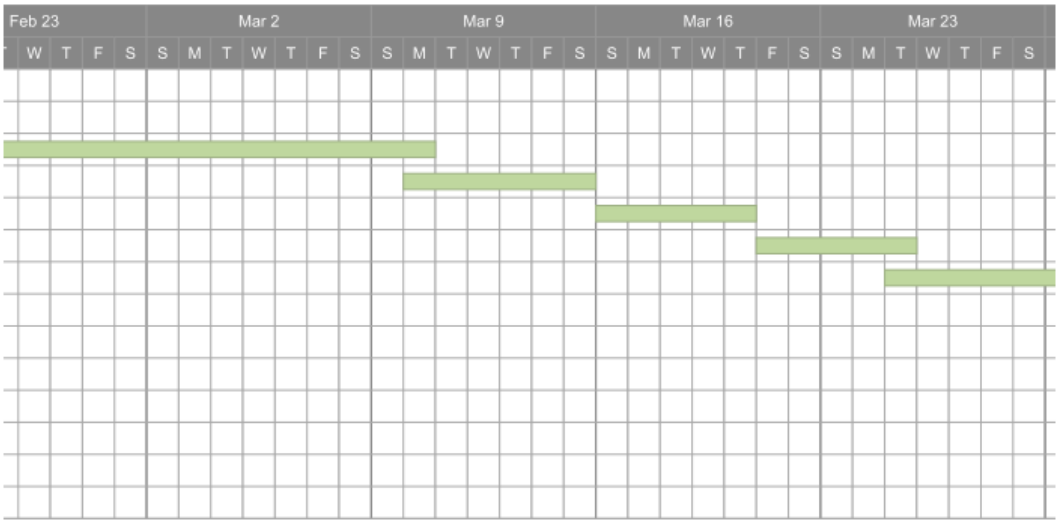

Figura 33: Diagrama de Gantt (2)

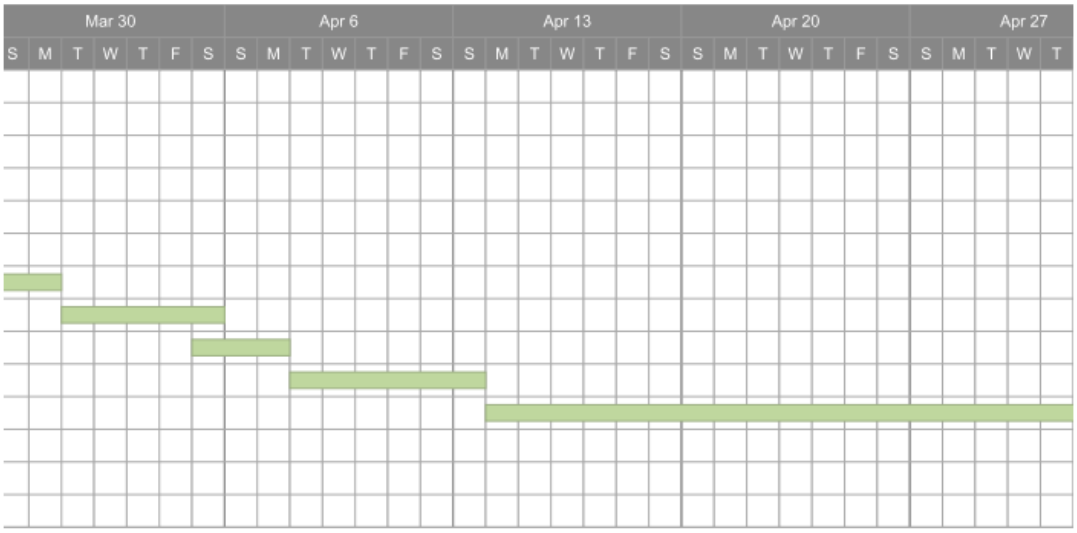

Figura 34: Diagrama de Gantt (3)

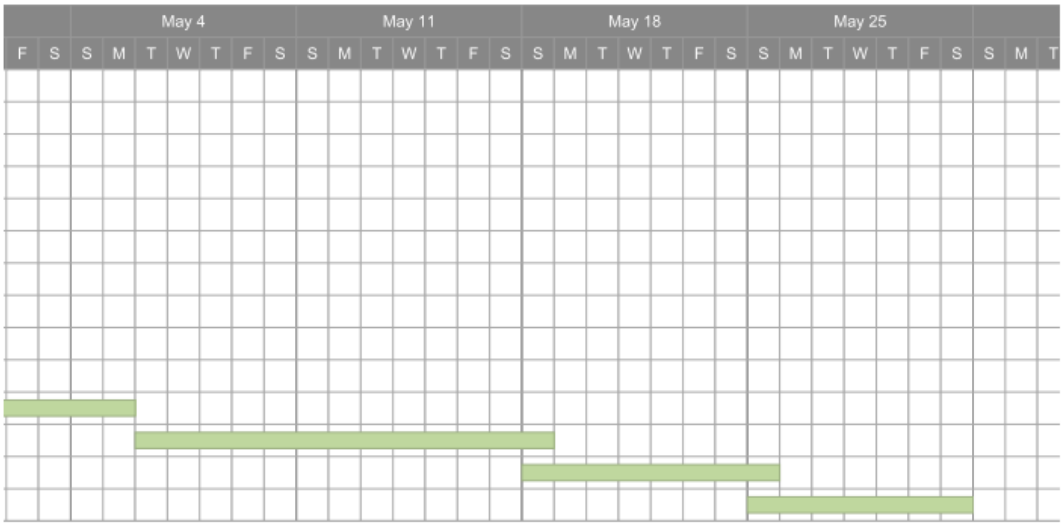

Figura 35: Diagrama de Gantt (4)

5. METODOLOGÍA

5.1 FORMA DE TRABAJO

La forma de trabajo ha consistido en:

- Un estudio de los mapas auto asociativos de Kohonen para comprender su funcionamiento y sus limitaciones. De esta manera se ha conocido los parámetros que debían ser incluidos en la herramienta y que el usuario pudiera modificar, así como un conocimiento general para saber el enfoque de los cálculos a realizar.
- Tras esto, se ha pasado al desarrollo de la misma, que consta de 2 partes:
  - **Backend de la aplicación**: consiste en la implementación de todos los algoritmos, clases, métodos, variables, etc. que garantizan el correcto funcionamiento y cálculos de la aplicación. Consiste en la implementación de la red de Kohonen, la carga de archivos, la interpretación de los mismos, las estructuras de datos que los almacenan, la declaración de los posibles parámetros, el control de errores de los datos, etc.
  - **Frontend de la aplicación**: consiste en toda la parte visual y de interacción con el usuario. Es aquí donde se ha incluido la visualización de los datos en sus distintas formas, la forma de cargar los datos, la interacción con la aplicación, el establecimiento de parámetros, el almacenamiento de resultados en archivos externos, etc.
- Cabe destacar que la aplicación es una Java Web Application. Esto se ha determinado conjuntamente con el tutor ya que se busca que la aplicación utilice las ventajas de presentación en navegadores Web al igual que la potencia de cálculo y precisión de resultados de un lenguaje de programación como Java.
- También es importante mencionar el uso de librerías externas de Javascript que servirán de ayuda para la visualización de datos. Esto no afecta el criterio de originalidad expuesto en la normativa del proyecto ya que tiene una importante carga de trabajo por parte del alumno para la adecuación de las mismas a la herramienta y sigue requiriendo de creatividad, originalidad y programación extensiva. Se ha utilizado la herramienta D3.JS.
- La aplicación ha sido desarrollada con programación orientada a objetos y utilizando el modelo de programación de MVC (modelo-vista-controlador) Struts.
- Para su demostración y futura utilización se aloja en un servidor local de Apache Tomcat.

5.2 MODELO DE SOFTWARE UTILIZADO

En lo que a metodología se refiere, el proyecto se ha desarrollado basado en un modelo incremental, en el que se han ido estableciendo objetivos en los incrementos cada 2 semanas para su cumplimiento, ofreciendo así una mejora continua.

- Inicialmente, se ha realizado un diseño arquitectónico de alto nivel para tener un esquema de la aplicación.
- Los requisitos completos se han definido previamente al desarrollo y se han ido asignando a las iteraciones en función de las fases explicadas en el punto 4.4 de este documento.
- Se ha diseñado de forma conceptual los pasos a seguir en el desarrollo

- La implementación se ha hecho en base a lo expresado en el punto de forma de trabajo desarrollado anteriormente. En cada fase se ha implementado lo especificado por los objetivos de cada iteración y en base a los comentarios del tutor.
- La evaluación de cada iteración ha sido rigurosa de forma que una vez finalizada la aplicación las pruebas realizadas sólo han sido necesarias para confirmar el buen funcionamiento de todo el sistema.

El motivo de la elección de esta metodología ha sido por 2 razones:

1. Debido al ámbito heterogéneo en el que se encuentran las redes neuronales, a la gran variedad de parámetros, y a la mejora de las mismas en base a los errores que cometan, era necesario que la metodología ofreciera la posibilidad de introducir cambios importantes en el enfoque y en el desarrollo si fuera necesario. Un desarrollo incremental se adecuaba perfectamente a esto. Por ejemplo, tras varios meses de desarrollo, se determinó que era necesario incluir la visualización de resultados en PDF para la facilidad del usuario en caso de no disponer de la herramienta. Esto fue fácilmente adecuado a la organización del proyecto añadiendo un incremento más al final.

2. Si bien existía un conocimiento previo por parte del alumno del uso de javascript para la representación de datos y de Java como lenguaje de programación, era conveniente cierta flexibilidad en caso de que sean necesarios cambios de estrategia de la implementación. También cabe destacar que debido a que se había determinado que habrían reuniones con el tutor cada 2 o 3 semanas, la aplicación se mejoraría de forma iterativa. Finalmente, un diseño previo de la interfaz puede no ser la mejor opción ya que en base a ensayo y error se puede conseguir una mejor usabilidad de la misma.

EXPLICACIÓN DEL MODELO INCREMENTAL

Como se muestra en la figura a continuación, el modelo incremental aplica secuencias lineales de forma escalonada mientras progresa el tiempo en el calendario. Cada secuencia lineal produce un incremento del software. El primer incremento generalmente es un producto esencial denominado núcleo.

En una visión genérica, el proceso se divide en 4 fases: análisis, diseño, código, y pruebas. Con esto se mantiene al cliente en constante contacto con los resultados obtenidos en cada incremento. Es el mismo cliente el que incluye o desecha elementos al final de cada incremento a fin de que el software se adapte mejor a sus necesidades reales. El proceso se repite hasta que se elabora el producto completo. De esta forma el tiempo de entrega se reduce considerablemente.

El Modelo Incremental es de naturaleza interactiva brindando al final de cada incremento la entrega de un producto completamente operacional al cliente, quien opinará del resultado obtenido y se procederá a realizar las modificaciones oportunas en el siguiente incremento. Este modelo es particularmente útil cuando no se cuenta con una dotación de personal suficiente (en el caso de este proyecto sólo es una persona la encargada del desarrollo). Los primeros pasos los pueden realizar un grupo reducido de personas y en cada incremento se añadirá personal, de ser necesario. Por otro lado los incrementos se pueden planear para gestionar riesgos técnicos.

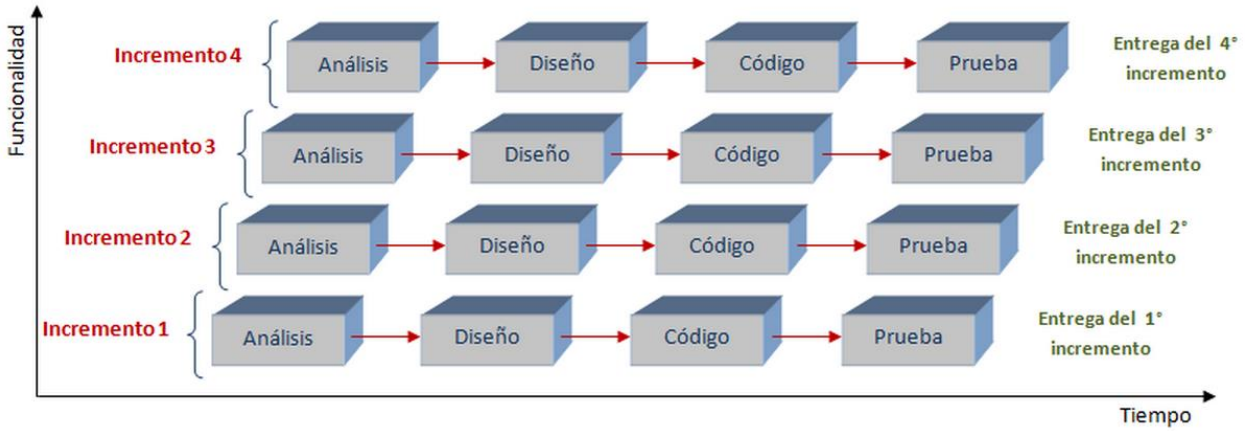

Figura 36: Modelo de software incremental

Durante el proceso se trata de llevar a cabo al proyecto en diferentes partes que al final terminará siendo la solución completa requerida por el cliente, pero éstas partes no se pueden realizar en cualquier orden, sino que dependen de lo que el cliente este necesitando con más urgencia, de los puntos más importantes del proyecto, los requerimientos más básicos, difíciles y con mayor grado de riesgo, ya que estos se deben hacer al comienzo, de manera que se disminuya la dificultad y el riesgo en cada versión.

De este modo se puede terminar una aplicación ejecutable (primera versión) que podrá ser entregada al cliente para que éste pueda trabajar en ella y el programador pueda considerar las recomendaciones que el cliente efectúe para hacer mejoras en el producto. Estas nuevas mejoras deberán esperar a ser integradas en la siguiente versión junto con los demás requerimientos que no fueron tomados en cuenta en la versión anterior. Es importante destacar que cada incremento no tiene por qué ser necesariamente una versión en sí misma.

El modelo incremental consiste en un desarrollo inicial de la arquitectura completa del sistema, seguido de sucesivos incrementos funcionales. Cada incremento tiene su propio ciclo de vida y se basa en el anterior, sin cambiar su funcionalidad ni sus interfaces. Una vez entregado un incremento, no se realizan cambios sobre el mismo, sino únicamente corrección de errores. Dado que la arquitectura completa se desarrolla en la etapa inicial, es necesario conocer los requerimientos completos al comienzo del desarrollo.

Al iniciar del desarrollo, los clientes o los usuarios, identifican a grandes rasgos, las funcionalidades que proporcionará el sistema. Se confecciona un bosquejo de requisitos funcionales y será el cliente quien se encarga de priorizar que funcionalidades son más importantes. Con las funcionalidades priorizadas, se puede confeccionar un plan de incrementos, donde en cada incremento se indica un subconjunto de funcionalidades que el sistema entregará. La asignación de funcionalidades a los incrementos depende de la prioridad dada a los requisitos. Finalizado el plan de incrementos, se puede comenzar con el primer incremento.

Las principales características de este modelo de software son:

- Se evitan proyectos largos y se entrega "algo de valor" a los usuarios con cierta frecuencia.
- El usuario se involucra más.
- Difícil de evaluar el costo total.
- Difícil de aplicar a los sistemas transaccionales que tienden a ser integrados y a operar como un todo.

- Requiere gestores experimentados.
- Los errores en los requisitos se detectan tarde.
- El resultado puede ser positivo.

A su vez, las ventajas principales del modelo incremental son:

- Con un paradigma incremental se reduce el tiempo de desarrollo inicial, ya que se implementa la funcionalidad parcial.
- También provee un impacto ventajoso frente al cliente, que es la entrega temprana de partes operativas del software.
- El modelo proporciona todas las ventajas del modelo en Cascada realimentado, reduciendo sus desventajas sólo al ámbito de cada incremento.
- Resulta más sencillo acomodar cambios al acotar el tamaño de los incrementos.

Por último, las principales desventajas del modelo son:

- El modelo incremental no es recomendable para casos de sistemas de tiempo real, de alto nivel de seguridad, de procesamiento distribuido y/o de alto índice de riesgos.
- Requiere de mucha planificación, tanto administrativa como técnica.
- Requiere de metas claras para conocer el estado del proyecto.

6. APLICACIÓN DE LA METODOLOGÍA Y RESULTADOS OBTENIDOS

6.1 DESCRIPCION DEL PROYECTO

El objetivo de la herramienta desarrollada es permitir al usuario definir, entrenar y ejecutar Mapas Autoasociativos. Mediante una interfaz gráfica y una visualización amigable también se busca interpretar de forma sencilla los resultados obtenidos.

A continuación se describen los procesos más relevantes que se han tenido en cuenta y que se consideraron necesarios para el correcto funcionamiento de la herramienta:

6.1.1 PREPROCESAMIENTO

La utilización del programa requiere una manipulación previa de los datos para adaptarlos al formato de los ficheros de entrada utilizados. Estos procesos se agrupan en un conjunto de operaciones denominado *preprocesamiento*.

De entre todos los significados del preprocesamiento neuronal los más relevantes son aquellas operaciones que adaptan el formato de los datos para que la información suministrada a la red sea comprensible por ésta, especialmente la normalización de los datos aunque también pueden realizarse todos aquellos análisis estadísticos que sirvan para una mejor comprensión del problema.

Dentro del programa se debe poder realizar la normalización euclídea de los vectores de entrada para que el algoritmo de aprendizaje de la red de Kohonen conserve su estabilidad, y los vectores de pesos se mantengan dentro de la esfera unidad. Este preprocesamiento incluido en el programa no impide que antes de introducir los datos al programa se realice cualquier otro tipo de preprocesamiento que se considere necesario sobre los datos.

6.1.2 PROCESO DE ENTRENAMIENTO

El proceso de entrenamiento/aprendizaje consiste en la adaptación de los pesos de las neuronas de la capa de Kohonen a medida que se presentan vectores de entrada correspondientes a las muestras.

El fichero de muestras que se van a presentar a la red se carga en memoria como un conjunto de vectores de valores normalizados. Independientemente de si se utiliza la norma Euclídea o no se normaliza, cada muestra debe tener tantos datos como neuronas haya en la capa de entrada.

Una vez cargados los patrones se lleva a cabo la presentación de las muestras (aplicar un vector patrón a la entrada de la red). El número de presentaciones está definido por el *periodo (p)*.

El número de veces (ciclos) que el fichero de datos es procesado durante el entrenamiento es:

$$ciclos = \frac{periodo}{número\_de\_patrones}$$

En cada una de estas presentaciones se procesa la entrada obteniendo la activación de una única neurona en la capa de Kohonen. Esta neurona se denomina *neurona ganadora*. Una vez hallada la neurona ganadora para un patrón de entrada se determinan las neuronas que se encuentran en su *vecindario*. Todas ellas aprenden el patrón de entrada aproximando los vectores de pesos de cada una de estas neuronas de la capa de Kohonen al vector del patrón de entrada.

A medida que avanza el aprendizaje algunos de los parámetros del aprendizaje van variando:

- El coeficiente de aprendizaje (η) va disminuyendo
- El número de presentaciones o "tiempo" transcurrido (t) va aumentando
- El vecindario va disminuyendo

El descenso de η se puede hacer de forma lineal o exponencial. En cada caso, el número de presentaciones, el vecindario y el valor de η se calculan de forma diferente.

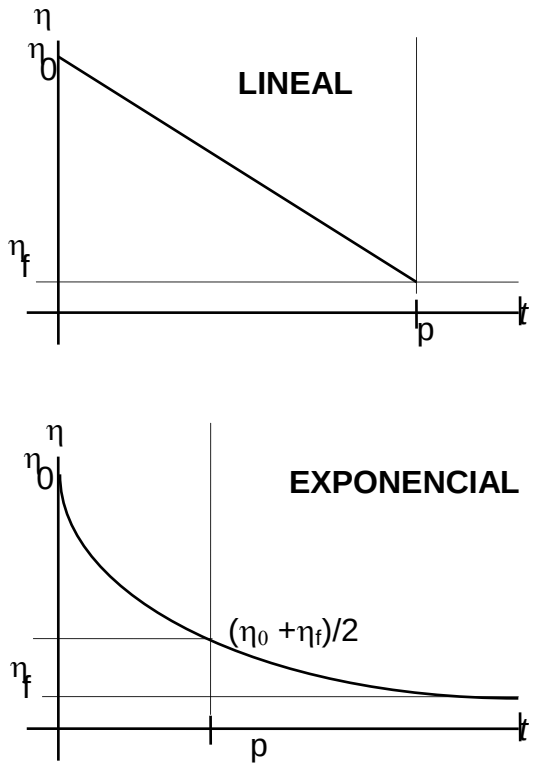

Figura 37: Variación lineal y exponencial del aprendizaje

REFUERZO

Si se van a utilizar r refuerzos entonces

- el valor del periodo se multiplica por el factor de extensión del período T_{factor}
- se aplica el factor de compresión η_{shrink} a η_0
- el vecindario toma el valor constante 1 (lo que quiere decir que en la fase de refuerzos sólo la neurona ganadora ve modificados sus pesos).

La condición de fin de entrenamiento y el tratamiento de los refuerzos seguirá el siguiente esquema:

```
Si (η-ηf<10^-5)
      Si (refuerzos=0)
            Fin de entrenamiento
      sino
            Refuerzos    ←    refuerzos-1
            nºmuestra    ←    0
            v0           ←    1
            p            ←    p. Tfactor
            η0           ←    η0 . ηshrink
```

6.1.3 PROCESO DE CLASIFICACIÓN

El proceso para clasificar un conjunto de datos con una red utilizando esta librería es prácticamente igual que el de entrenamiento, salvo que no se modifica ningún peso y no se utiliza ningún parámetro de los que son necesarios para el entrenamiento. Tan sólo se *calcula la neurona ganadora* para cada muestra, escribiendo en el fichero de clasificación la etiqueta de la muestra y las coordenadas de la neurona ganadora como fila y columna, teniendo en cuenta que comienzan a numerarse en el 0.

6.1.4 PROCESO DE ANÁLISIS DE RESULTADOS

Aunque los ficheros de salida de la herramienta pueden ser interpretados por separado, ya que contienen toda la información obtenida durante un entrenamiento o una clasificación, se han desarrollado un conjunto de funciones de *postprocesamiento* para garantizar una mejor interpretación de los resultados utilizando la librería Javascript D3.JS (lo que permite al usuario ver visualmente las muestras clasificadas) así como la generación de informes PDF con un resumen de los resultados obtenidos y un conjunto de estadísticas de los mismos.

De esta manera se pretende tener una forma rápida de conocer información útil sobre las clasificaciones obtenidas para un conjunto dado de patrones.

A partir de un fichero de clases (resultado de una clasificación) y de los ficheros de datos que se han utilizado para obtener dicha clasificación, el programa realiza mapas de activación, tablas de muestras, ordenar las entradas, según la clasificación y determinar una medida del error cometido. Toda esta información se guarda en un fichero JSON que contiene la información relevante a la visualización de los resultados y en un fichero PDF utilizado para resumir dichos resultados.

6.1.5 FORMATO DE LOS FICHEROS DE ENTRADA/SALIDA

Toda la entrada/salida de la red se realizará mediante ficheros con un formato determinado. Existirán varios tipos de ficheros que se podrán identificar por su nombre y/o extensión. La siguiente tabla enumera los distintos tipos de fichero que se utilizarán, su contenido y su empleo como entrada o salida por parte de la herramienta. En lo que al contenido y estructura de los ficheros mencionados, los mismos son explicados más adelante en el proceso de desarrollo.

Tabla 2: Ficheros utilizados por la herramienta

Extensión	Contenido	Tipo
MUESTRA.dat	Datos de entrada (muestras)	Entrada
MUESTRA.xml	Datos de entrada (muestras), otra opción a utilizar como datos de entrada.	Entrada
MUESTRA_RED.xml	Fichero de red que contiene los parámetros de la red y en caso de estar entrenada los pesos de la misma. Su nombre permite saber el nombre de la red y el nombre de la muestra con la cual ha sido entrenada. Es utilizado como fichero de entrada si se quiere modificar una red y sus pesos mediante otro entrenamiento. Es un fichero de salida tras entrenar una red.	Entrada/Salida
MUESTRA_RED_CLS.xml	Resultados de la clasificación. Su nombre permite saber el tipo de fichero (clasificación), el nombre de las muestras clasificadas, y el nombre de la red utilizada.	Salida
MUESTRA_RED_RPT.pdf	Informe de resultados de clasificación.	Salida
MUESTRA_RED_VIEW.json	Fichero con la información necesaria para la visualización de los resultados. Puede ser utilizada para cargar nuevamente una visualización.	Salida

6.1.6 FUNCIONAMIENTO DEL SISTEMA

La herramienta es un *Java Web Applicaction* que correrá en un servidor Apache Tomcat. El usuario podrá decidir en qué forma utilizar la herramienta para conseguir su propósito. Los modos de trabajo son tres, y definen tanto que datos se proporcionan al sistema como que proceso se hace con ellos y que salida devolverá:

- Modo de entrenamiento

  Se utiliza el fichero de muestras y de forma opcional el fichero de red para cargar los parámetros de la red y los datos mediante la cual la misma será entrenada. En caso de no utilizar un fichero de red, el usuario podrá seleccionar la introducción de nuevos

parámetros de red. La salida son todos los ficheros mencionados en la tabla y una visualización mediante D3.JS para la interpretación de los resultados.

- Modo de clasificación

  Se utiliza un fichero de muestras a clasificar y un fichero de red ya entrenada para realizar la clasificación. La salida son todos los ficheros mencionados en la tabla y una visualización mediante D3.JS para la interpretación de los resultados.

- Modo de visualización

  Se utiliza el fichero JSON descrito en la tabla de ficheros para volver a cargar una visualización anterior. La salida es únicamente dicha visualización.

6.2 REQUISITOS INICIALES DE USUARIO

A continuación de muestran los requisitos de usuario que han sido determinados antes de comenzar con el desarrollo de la aplicación:

1. La aplicación deberá mantener una exactitud de los resultados de clasificación/entrenamiento de al menos 6 decimales.
2. La aplicación será desarrollada para su acceso en red siendo la misma alojada en un servidor.
3. La aplicación dispondrá de una interfaz gráfica que permitirá al usuario escoger si desea entrenar un red, clasificar una serie de muestras utilizando una red, o visualizar los resultados de una clasificación.
4. La aplicación funcionará por sesiones de usuario, que caducarán a las 2 horas y media de inactividad.
5. Cada sesión dispondrá de una carpeta que almacenará los ficheros subidos por el usuario y aquellos generados por la herramienta.
6. Se debe disponer de la posibilidad de descargar los archivos de dicha carpeta.
7. Se almacenarán en la sesión todas las muestras subidas y redes utilizadas para entrenar/clasificar, de modo que el usuario pueda volverlas a seleccionar para realizar modificaciones sobre las mismas y comparaciones.
8. En la selección de "Entrenar red", al subir una muestra se debe poner por defecto el valor de la capa de entrada, el cual no es modificable.
9. En la selección de "Entrenar red", se deberá permitir al usuario subir una muestra de archivos en formato .DAT o .XML. Dichos archivos deben ser validados antes de aceptarse como muestra válida. La estructura de dichos archivos quedan a criterio del desarrollador.
10. En la selección de "Entrenar red", se deberá permitir al usuario escoger si desea cargar una red con unos parámetros ya establecidos desde un fichero o si desea introducirlos manualmente.
11. En la selección de "Entrenar red", en caso de seleccionar subir un fichero, se deberá escoger si se desea mantener los pesos del fichero como pesos iniciales del entrenamiento o si se desea comenzar con pesos aleatorios.
12. En la selección de "Entrenar red", los parámetros que el usuario debe poder introducir manualmente o se deben poder cargar desde el fichero se describen a continuación. Para

comprender el significado de dichos parámetros referirse al punto 6.1 del presente documento:

- Normalización: será un desplegable que permita escoger entre "Euclídea" o "Ninguna".
- Presentación: será un desplegable que permita escoger entre "Secuencial" o "Aleatoria".
- Lado Kohonen: será un TextBox que no dispondrá de ningún valor por defecto y que debe ser introducido por el usuario.
- ETA Inicial: será un TextBox con valor por defecto 0.8
- ETA Final: será un TextBox con valor por defecto 0.0
- Período: será un TexBox con valor por defecto 500.
- Curva: será un desplegable que permitirá escoger entre "Lineal" o "Exponencial".
- Vecindario Inicial: será un TextBox con valor por defecto 1.
- Refuerzo: una selección de "Si" o "No".
- Número de Refuerzos: un TextBox con valor por defecto 3.
- Extensión Período: un TextBox con valor por defecto 2.
- Compresión: un TextBox con valor por defecto 0.7.

13. En la selección de "Entrenar Red", en caso de no utilizar refuerzo, no se permitirá al usuario introducir los valores de Número de refuerzo, extensión período y compresión. En caso de que los parámetros sean subidos desde un fichero, los mismos no serán tomados en cuenta.
14. En la selección de "Entrenar Red", una vez introducidos los parámetros, se procederá a utilizar el algoritmo de la red de Kohonen para entrenar, clasificar, y visualizar los resultados.
15. En la selección de "Clasificar Muestra", se deberá permitir al usuario subir una muestra y una red ya entrenada para la clasificación.
16. En la selección de "Clasificar Muestra", se debe validar que la capa de entrada de la red coincida con la dimensión de la muestra.
17. En la selección de "Clasificar Muestra", una vez cargados los ficheros, se procederá a utilizar el algoritmo de Kohonen para clasificar la muestra y visualizar los resultados.
18. En la sección de "Visualizar Resultados " se podrá subir un fichero que contenga la información necesaria para la visualización de los resultados y mostrará la visualización de forma gráfica.
19. En la sección de "Visualizar Resultados", se validará la estructura del fichero a subir.
20. En la sección de "Visualizar Resultados", se guardará una lista de los ficheros subidos para poder volver a seleccionarlos.
21. A la hora de entrenar y clasificar, se generará un fichero PDF con toda la información relevante a los resultados de la clasificación. Dicho fichero se podrá descargar.

22. Tras cada entrenamiento y clasificación, se permitirá al usuario descargar un ZIP con todos los ficheros relevantes a dicho entrenamiento/Clasificación.
23. En todo momento el usuario podrá descargar un ZIP con todos los contenidos de su carpeta de sesión.
24. La sección de "Visualizar Resultados", contendrá 3 modos de visualización:
    - Clases: permitirá ver las clases y sus muestras asociadas, indicando para la clase los datos relevantes de la misma al igual que para las muestras.
    - Muestras: permitirá ver las muestras agrupadas por clases, pero sólo mostrando información de las muestras.
    - Mapa de activación: mostrará la matriz de activación de la red y la información relevante de cada clase.

6.3 DISEÑO ARQUITECTÓNICO

Antes de proceder a las fases de los incrementos, se ha desarrollado un diseño arquitectónico de alto nivel sobre el cual se ha enfocado el resto del diseño del software. De esta manera, se dispone desde el principio con los diferentes módulos que deberían haber en la herramienta y las posibles interfaces que se han de utilizar. El diseño arquitectónico conseguido es el siguiente:

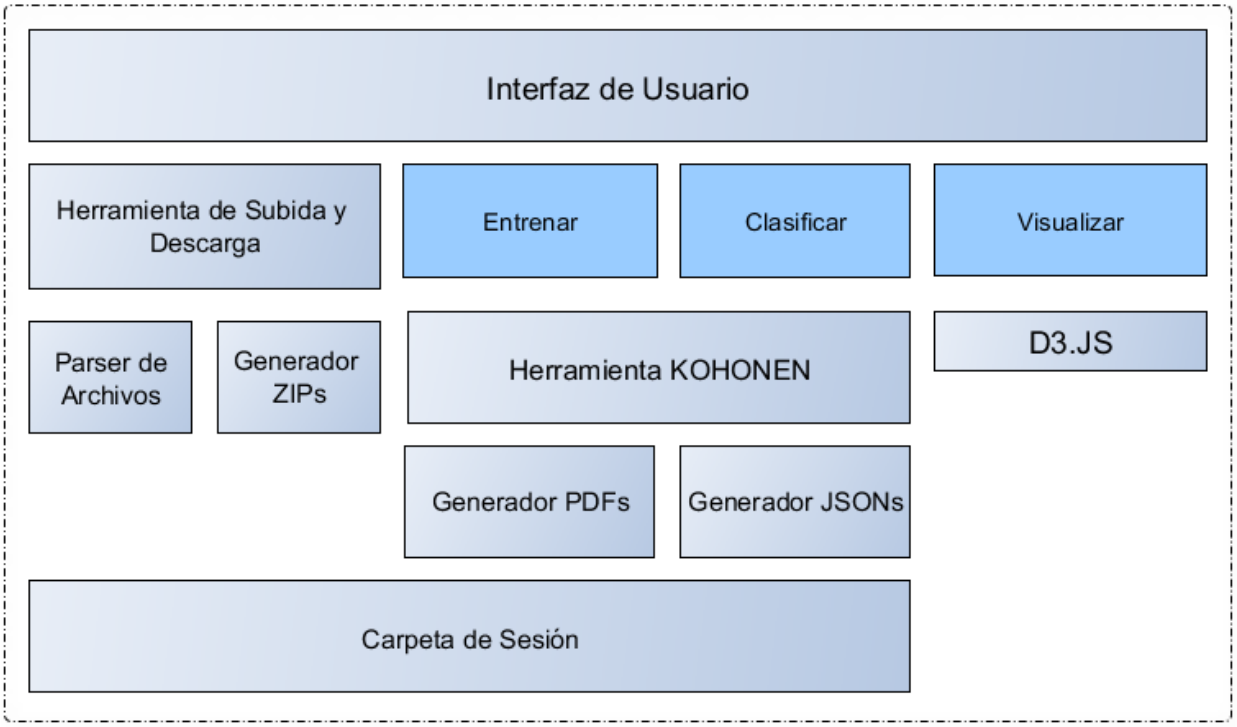

Figura 38: Diseño arquitectónico de la herramienta

Como se puede observar en la figura 41, se ha determinado que los componentes que la herramienta ha de tener son:

- Interfaz de usuario: la cual sirve de comunicación entre el usuario y la herramienta. La misma tiene acceso a la herramienta de subida y descarga de archivos, y a las pestañas de Entrenar, Clasificar y Visualizar.

- Herramienta de Subida y Descarga: permite al usuario subir y descargar los archivos de su sesión. A esta herramienta se accede desde alguna de las 3 ventanas de Entrenar, Clasificar y Visualizar. Al ser común a toda la interfaz se ha puesto al mismo nivel que las ventanas. Esta herramienta se puede conectar a la carpeta de sesión para la descarga de los archivos, al Parser de Archivos para validar los mismos, y al Generador de ZIPs para generar un sólo enlace de descarga.
- Parser de Archivos: permite interpretar el contenido de los archivos y transformarlos según sea necesario. Por ejemplo, permite leer un fichero de muestras y determinar si la estructura es válida, o uno de red para mostrar los parámetros de la misma.
- Generador ZIPs: permite generar un archivo ZIP con todos los contenidos de la carpeta de sesión del usuario que podrá descargarse. También permite generar el fichero ZIP que contenga sólo los archivos de una clasificación en particular.
- Herramienta KOHONEN: es el módulo más importante ya que contiene todos los algoritmos necesarios para la ejecución de la red. Es una red de neuronas, que carga muestras y permite entrenar la red y clasificar muestras.
- Generador de PDFs: permite generar archivos PDF con un resumen de un entrenamiento/clasificación. Dicho fichero se carga tras la ejecución de la herramienta de KOHONEN.
- Generador de JSON: permite generar archivos JSON con la información necesaria para la visualización de la red. Es información contiene los datos más relevantes de las muestras y las clases, entre las que destacan las distancias entre las mismas y sus etiquetas.
- Carpeta de Sesión: es la carpeta que se crea en el momento que se inicia sesión y que contiene todos los archivos relevantes a la misma, tanto los que sube el usuario como aquellos generados por la herramienta. Esta carpeta es destruida cuando caduca o se cierra la sesión.
- D3.JS: es una librería Javascript que permite la visualización gráfica de los resultados del entrenamiento/clasificación de la red. Utiliza archivos JSON para cargar la información necesaria.

6.4 DESARROLLO DE LA HERRAMIENTA DE KOHONEN Y PARSER DE ARCHIVOS

Debido a la importancia de mantener la exactitud de los resultados y a que este era el módulo base sobre el cual podrían actuar el resto, se ha desarrollado la Herramienta de Kohonen al principio de la fase de desarrollo.

A continuación se muestra el proceso seguido para el desarrollo las primeras 7 fases del proyecto descritas en el punto 4 del presente documento, que hacen referencia al diseño e implementación de los algoritmos de Kohonen, comenzando por un estudio del funcionamiento de estas redes (ver el apartado 3.2), el establecimiento y correcto funcionamiento del entorno, implementación de los algoritmos, pruebas de los algoritmos y correcciones de los mismos, y la utilización de ficheros para guardar y cargar resultados.

Es importante destacar que se explican los métodos más relevantes de las clases utilizadas, pero para una mayor comprensión de todo el funcionamiento de los mismos se ruega ver el código del proyecto.

6.4.1 PRIMER INCREMENTO

Antes de comenzar con este incremento se ha estudiado el funcionamiento y las limitaciones de las redes de Kohonen y se ha establecido y configurado el entorno sobre el cual ha sido desarrollada la herramienta.

6.4.1.1 OBEJTIVOS PRINCIPALES

- Implementar las clases necesarias para el correcto funcionamiento de los algoritmos de Kohonen, incluyendo los atributos, los nombres de los métodos a utilizar así como sus parámetros.

6.4.1.2 ANÁLISIS DE RIESGOS

- Posibilidad de que los algoritmos sean más complicados de implementar de lo que parecen: para hacer frente a este riesgo se ha estudiado considerablemente el funcionamiento de las redes de Kohonen y se ha dado un margen lo suficientemente largo de tiempo para evitar cualquier problema.

6.4.1.3 EJECUCIÓN

Se ha determinado que la herramienta de Kohonen ha de tener la siguiente estructura:

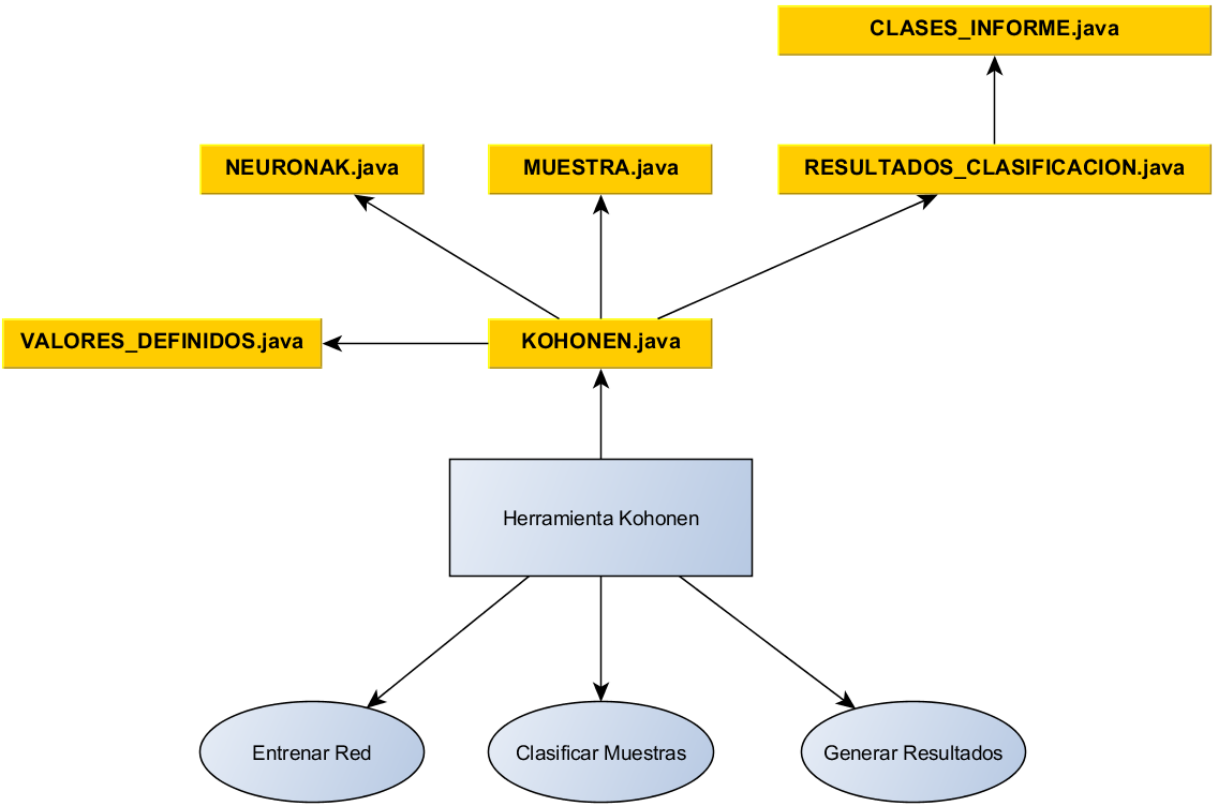

Figura 39: Estructura herramienta de Kohonen

Como se puede observar, la Herramienta de Kohonen está compuesta por las siguientes clases:

- *KOHONEN.java*: los objetos de esta clase son redes de Kohonen. Esta es la única clase que es necesario utilizar de todas las que contiene la librería, para manipular a través de sus métodos redes de este tipo.
- *NEURONAK.java*: es una clase de soporte utilizada por KOHONEN para representar cada neurona de la capa de Kohonen.
- *MUESTRA.java*: es una clase de soporte utilizada por la clase KOHONEN para representar cada muestra o patrón como una etiqueta y un vector de valores reales.
- *VALORESDEFINIDOS.java*: es una clase de soporte utilizada para representar y poder acceder directamente a los valores posibles de los parámetros de la red. También contiene los valores por defectos, etc.
- *RESULTADOS_CLASIFICACION.java:* es una clase de soporte utilizada para almacenar todos los datos necesarios de los resultados de una clasificación, de forma que el acceso a los mismos sea sencillo.
- *CLASES_INFORME.java:* es otra clase de soporte utilizada para almacenar los datos necesarios de los resultados de una clasificación, únicamente que son utilizados para generar la matriz de activación.

La relación existente entre ellas tiene como elementos simples de procesamiento los objetos de la clase NEURONAK, que se denominan *neuronas*. Las neuronas son capaces de realizar las operaciones básicas relacionadas con sus pesos, de forma independiente del resto de neuronas del sistema y al estado actual del mismo.

Un objeto *capa* contiene una matriz bidimensional de neuronas, similares en estructura. El número y las características de las neuronas que contienen cada capa viene determinado por los atributos del objeto red, de la clase KOHONEN, que contiene dicha capa.

El objeto *red* contiene y gestiona todos los parámetros que establecen el estado del sistema en cada momento, tanto en su creación, como en su evolución a lo largo del tiempo, cualquiera que sea el modo en que se ejecute el sistema, clasificación o entrenamiento.

A continuación se describen más detalladamente las clases que componen la herramienta de Kohonen:

KOHONEN.java

Los objetos de esta clase son redes de Kohonen. Esta es la única clase que es necesario utilizar de todas las que contiene la librería, para manipular a través de sus métodos redes de este tipo. Contiene toda la información sobre la topología de la red, sobre su modo de ejecución, y sobre su estado en cada instante del entrenamiento o clasificación.

Realiza la construcción de la red según los parámetros que aparezcan en la línea de comandos y en el fichero de configuración. Gestiona la evolución temporal de los parámetros de la red y controla los procesos de entrenamiento y clasificación enviando mensajes a la capa que contiene.

Sus atributos más importantes son:

- **nx** (int): número de columnas de *n* neuronas en la capa de Kohonen.

- **ny** (int): número de filas de neuronas en la capa de Kohonen. En esta herramienta es igual a nx, dando lugar a una capa de Kohonen cuadrada.
- **ninput** (int): número de neuronas de entrada de la red, es decir, longitud de cada vector de patrones.
- **CapaK** (NEURONAK[][]): matriz de neuronas (objetos NEURONAK) que representan la capa de Kohonen.
- **EtaIni** (float): valor inicial de η. Toma valores entre 0 y 1.
- **EtaFin** (float): valor final de η. Toma valores entre 0 y 1.
- **EtaShrink** (float): valor de compresión de η. Toma valores entre 0 y 1.
- **Curva** (int): Indica si el descenso de η se hace de forma lineal o exponencial. Puede tomar los valores LINEAL o EXPENONECIAL de la clase *VALORESDEFINIDOS.java*
- **Periodo** (int): Número de presentaciones que se harán a la red en caso de que Curva sea LINEAL. Si la curva de descenso de η es exponencial entonces el período lo calcula el método de entrenamiento.
- **Refuerzo** (int): Indica si se va a utilizar refuerzo en el entrenamiento. Puede tomar los valores SI o NO de la clase *VALORESDEFINIDOS.java*
- **Refuerzos** (int): Número de refuerzos. Este valor sólo se utiliza si el atributo Refuerzo tiene un valor SI.
- **ExtPeriodo** (int): Factor de extensión del período. Sólo se utiliza si Refuerzo es SI, en cuyo caso, cada ciclo de refuerzo tendrá un período que será igual al período del ciclo anterior multiplicado por ExtPeriodo.
- **VecIni** (int): Vecindario inicial del entrenamiento de la red. Esto indica el número de neuronas alrededor de la ganadora que verán modificados sus pesos en el entrenamiento.
- **TipoPresentación** (int): Indica de que forma se van a presentar los patrones a la red. Puede tomar los valores SECUENCIAL o ALEATORIA de la clase *VALORESDEFINIDOS.java*.
- **TipoNormalización** (String): Indica qué normalización se va a aplicar al vector de entrada. Puede tomar los valores EUCLIDEA o NINGUNA de la clase *VALORESDEFINIDOS.java*.

Los métodos más importantes de la clase *KOHONEN.java* son:

- **`public KOHONEN(int Num_Ent, int x):`**

  Constructor de objetos de la clase KOHONEN. Num_Ent es el tamaño de la capa de entrada y x es el lado de la capa de Kohonen, o sea que en la capa habrá x^2 neuronas. Todos los atributos de la red toman los valores por defecto definidos en *VALORESDEFINIDOS.java* y si se desea modificar su valor se han de utilizar otros métodos, después de haber construido la red.

- **`public KOHONEN (String nombre_fichero, String directorio):`**

Constructor de objetos de la clase KOHONEN. Se utilizan los parámetros para ubicar el fichero XML mediante el cual se cargará la red y los atributos de la misma.

- **`public void cargar_pesos(String nombre_fichero, String directorio):`**

  Permite cargar los pesos de la red de un fichero de red XML en una red ya creada.

- **`protected NEURONAK Ganadora(MUESTRA entrada):`**

  Calcula la neurona ganadora para una muestra en particular utilizando la distancia Euclídea. Toma como parámetro una MUESTRA y devuelve una NEURONAK, siendo esta la neurona ganadora. También modifica los atributos *fila* y *columna* correspondiendo aquella fila y columna de la matriz de la capa de Kohonen en la que se encuentra la neurona ganadora.

- **`public void RellenarIniciales():`**

  Función que permite cargar en los atributos auxiliares los valores actuales de una serie de atributos que cambian en el entrenamiento.

- **`public void RestaurarIniciales():`**

  Función que permite devolver los valores iniciales de los atributos utilizando los atributos auxiliares tras un entrenamiento.

- **`public int Entrenar(List<MUESTRA> Muestras, int num_muestras):`**

  Ejecuta el proceso de entrenamiento de la red, de acuerdo a los valores de los atributos de ésta. Los patrones se toman de la lista de muestras. Esta función puede devolver de la clase *VALORESDEFINIDOS.java* los valores ERROR_FICHERO (indicando que hay un error en los parámetros) u OK (indicando que el entrenamiento ha ido bien).

- **`public void Salvar(String nombre_fichero, String directorio):`**

  Guarda la red con sus parámetros y pesos en el directorio indicado con el nombre de fichero introducido.

- **`public List<CLASES_INFORME> Clasificar(String nombre_fichero_muestras, String nombre_red, String directorio):`**

  Permite clasificar una serie de muestras recogidas automáticamente de un fichero en base a los pesos y parámetros de la red. El nombre del fichero de red correspondiente a la red actual es utilizado para generar el fichero en el que se guardarán los resultados de la clasificación. Devuelve una lista de las clases (neuronas) que han sido activadas con la información de las muestras que pertenecen a las mismas y otros datos de interés.

NEURONAK.java

Clase utilizada por KOHONEN para representar cada neurona de la capa de Kohonen. Son los elementos básicos de cada una de las redes generadas por la herramienta y equivalen al concepto *neurona* de una red de Kohonen.

Sus atributos son:

- **Pesos** (List<Double>): lista que contiene los pesos asociados a cada neurona.

- **Frec** (double): frecuencia de activación.

Los métodos más importantes de la clase *NEURONAK.java* son:

- **`public void ReservaPesos(int size_pesos):`**

  Reserva memoria para el número de pesos indicado por el parámetro de entrada numpesos.

- **`public void ModificarPesos(double eta, MUESTRA entrada):`**

  Modifica los pesos de la neurona. Recorre cada uno de los pesos asociados a la neurona y les aplica la siguiente función de modificación: Pesos[i] += eta * (entrada[i] - Pesos[i]).

- **`public int GetLongitudPesos():`**

  Devuelve el tamaño de la lista de pesos.

- **`public double Distancia(MUESTRA entrada):`**

  Se busca la mínima distancia entre la muestra y la neurona.

MUESTRA.java

Clase utilizada por la clase KOHONEN para representar cada muestra o patrón como una etiqueta y un vector de valores reales.

Sus atributos son:

- **etiqueta** (String): etiqueta de la muestra
- **datos** (List<Double>): lista que contiene los valores

Los métodos de la clase *MUESTRA.java* son:

- **`public int GetLongitudDatos():`**

  Devuelve la dimensión de la muestra.

- **`public void ReservaDatos(int size_datos):`**

  Reserva en memoria el tamaño de la lista de datos.

- **`public void NormalEuclidea():`**

  Se normaliza cada dato con la normal euclídea.

VALORESDEFINIDOS.java

Esta clase contiene todos los valores que pueden adoptar algunos atributos de la clase KOHONEN así como aquellos valores por defecto. Cabe destacar que todos sus valores son estáticos por lo que no es necesario crear un objeto de la clase.

Sus atributo son:

Códigos de Error:

- **OK** (int) = 0: indica que no hay ningún error.
- **ERROR_FICHERO** (int) = 1: indica que hay un error con el fichero en general
- **ERROR_FORMATO** (int) = 2: indica que hay un error con el formato del fichero
- **ERROR_CAPAENTRADA** (int) = 3: indica que hay un error debido a la capa de entrada y el tamaño de las muestras
- **ERROR_CAPAK** (int) = 4: indica que hay un error en la capa de Kohonen
- **ERROR_EOF** (int) = 5: indica que hay un error al final de la lectura del fichero

Curva:

- **EXPONENCIAL** (int) = 0: valor asignado para la curva exponencial
- **LINEAL** (int) = 1: valor asignado para la curva lineal

Tipos de presentación

- **ALEATORIA** (int) = 0: indica que el orden de presentación de muestras es aleatorio
- **SECUENCIAL** (int) = 1: indica que el orden de presentación de muestras es secuencial

Tipos de normalización

- **EUCLIDEA** (String) = "E": indica que el tipo de normalización de datos es mediante la distancia euclídea.
- **NINGUNA** (String) = "N": indica que no se normalizan los datos de entrada.

Otros valores

- **RANDOM** (String) = "RANDOM"
- **SI** (int) = 1
- **NO** (int) = 0
- **TRUE** (int) = 1
- **FALSE** (int) = 0
- **RAND_MAX** (int) = 0x7FFFFFFF: indica el máximo valor que puede alcanzar un número aleatorio
- **LONG_MAX_ETIQ** (int) = 15: indica el máximo tamaño que puede alcanzar una etiqueta

- **LONG_MAX_FICH** (int) = 100: indica el máximo tamaño que puede tener un fichero (MB).

Valores por defecto (pertenecientes a una clase interna llamada DEFAULT)

- **ETAIN** (double) = 0.25: valor por defecto de eta inicial
- **ETAFIN** (double) = 0.01: valor por defecto de eta final
- **ETASHRINK** (double) = 0.7: valor por defecto de la compresión de eta.
- **CURVA** (int) = *LINEAL*: valor por defecto de la curva
- **PERIODO** (int) = 500: valor por defecto del período
- **EXTPERIODO** (int) = 2: valor por defecto de la extensión del período
- **REFUERZO** (int) = *NO*: valor por defecto de la opción de utilizar refuerzo
- **NUMREFUERZOS** (int) = 3: valor por defecto del número de refuerzos
- **VECINI** (int) = 5: valor por defecto del vecindario inicial
- **PRESENTACIÓN** (int) = *ALEATORIA*: valor por defecto del tipo de presentación de datos.
- **NORMALIZACIÓN** (String) = *EUCLIDEA:* valor por defecto del tipo de normalización de datos.

Los métodos más importantes de la clase *VALORESDEFINIDOS.java* son:

- **`public static double aleatorio():`**

  Genera un número aleatorio real.

- **`public static int aleatorio(int min, int max):`**

  Genera un número aleatorio entero entre los valores indicados por los parámetros *min* y *max.*

RESULTADOS_CLASIFICACION.java

Esta clase contiene una estructura de datos que representa una muestra tras el proceso de clasificación indicando la distancia de la muestra a la neurona de Kohonen (clase) a la que pertenece.

Sus atributos son:

- **etiqueta** (String): indica la etiqueta de la muestra a la que se hace referencia

- **fila** (int): indica la fila en la matriz de la capa de Kohonen de la clase a la que pertenece la muestra.
- **columna** (int): indica la columna en la matriz de la capa de Kohonen de la clase a la que pertenece la muestra.
- **distancia** (double): indica la distancia de la muestra a la clase (neurona).

Los métodos de la clase *RESULTADOS_CLASIFICACION.java* son todos los necesarios para introducir el valor de los atributos y recuperarlos.

CLASES_INFORME.java

Esta clase contiene una estructura de datos que representa una clase tras el proceso de clasificación indicando las muestras que contiene (sus etiquetas y distancia) y las distancia media de la clase.

Sus atributos son:

- **id** (int): ID asignado a la clase en cuestión
- **fila** (int): fila a la que pertenece la clase en la matriz de la capa de Kohonen
- **columna** (int): columna a la que pertenece la clase en la matriz de la capa de Kohonen
- **numero_clases** (int): número de muestras que pertenecen a la clase en cuestión (se ha de actualizar el nombre de este atributo en futuras versiones).
- **Etiquetas** (List<String>): lista con todas las etiquetas de las muestras que pertenecen a esta clase.
- **Distancias** (List<String>): lista con todas las distancias de las muestras que pertenecen a esta clase colocadas en el mismo orden que la lista de etiquetas.
- **distancia_media** (Double): distancia media de todas las muestras que pertenecen a la clase.

Los métodos más importantes de la clase *CLASES_INFORME.java* son:

- **`public CLASES_INFORME():`**

  Constructor de la clase. Establece el número de clases inicialmente a 0 e inicializa las listas Distancias y Etiquetas.

- **`public Double getDistancia_media():`**

  Devuelve la distancia media de las muestras de la clase.

- **`public Double getMaxDistance():`**

  Devuelve la máxima distancia de una muestra a la clase.

- **`public Double getMinDistance():`**

  Devuelve la mínima distancia de una muestra a la clase.

6.4.1.4 PLANIFICACIÓN, PROBLEMAS Y PRUEBAS

Se ha definido correctamente la estructura de la Herramienta de Kohonen con todas las clases, atributos y métodos que la componen. No ha habido ningún problema importante en este incremento. Se ha comprobado que la sintaxis de todo el programa sea la correcta para garantizar así la compilación del programa.

No se ha podido determinar la exactitud de los cálculos y sobre todo de la clasificación debido a que aún no ha sido desarrollado el módulo para cargar los archivos.

Se ha determinado que en el siguiente incremento es necesario comenzar con el desarrollo de dicho módulo para garantizar que la herramienta entrena y clasifica correctamente (hasta ahora sólo se ha demostrado que compila una vez implementados los algoritmos).

Por tanto, para el próximo incremento es necesaria la implementación de las funciones para leer las muestras, leer una red desde un fichero, salvar la red en un fichero, y guardar los datos de la clasificación en un fichero.

6.4.2 SEGUNDO INCREMENTO

Una vez implementados los algoritmos correspondientes a la herramienta de Kohonen, se ha determinado necesario la posibilidad de cargar datos de muestra desde un fichero, datos de red desde un fichero, guardar los datos de red en un fichero, y guardar los datos de clasificación en un fichero.

6.4.2.1 OBJETIVOS PRINCIPALES

- Diseñar la estructura correspondiente a los diferentes tipos de ficheros que tratará directamente la Herramienta de Kohonen.
- Implementar las funciones necesarias para tratar dichos ficheros y validarlos.

6.4.2.2 ANÁLISIS DE RIESGOS

- Posibilidad de que el manejo de datos desde Java sea más complicado de lo inicialmente previsto. Frente a esto se han considerado varias librerías de manejo de datos.
- Diseñar una estructura para los ficheros de datos que no sea la más adecuada para su representación. Para ello se ha buscado hacer una estructura muy simple y que pueda extenderse en caso de que sea necesario.
- No validar los ficheros correctamente. Frente a ello se ha dado un margen de tiempo para las pruebas con ficheros.

6.4.2.3 EJECUCIÓN

Lo primero que se ha hecho ha sido determinar la estructura que han de tener los ficheros. En lo que a las *muestras* se refiere, en un principio se ha ideado un fichero XML que tiene la siguiente estructura:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<muestras>
    <dimension>DIMENSION_MUESTRAS</dimension>
    <muestra>
          <etiqueta>ETIQUETA_MUESTRA</etiqueta>
        <dato>0.548952</dato>
        <dato>0.52904</dato>
        <dato>0.519982</dato>
          .
          .
        .
      </muestra>
      <muestra>
      .
      .
      .
      </muestra>
</muestras>
```

El formato es muy sencillo y fácil de leer, pero ha dado problemas a la hora de abrirlo con Excel ya que el XML no sigue una estructura estándar. Frente a esto se ha determinado que el fichero de muestras debería poder ser fácilmente modificable en Excel, por lo que se ha decidido permitir otro tipo de fichero de muestras: un fichero en texto plano separado por tabulador con una extensión .DAT. Su estructura es muy sencilla:

```
-------------------------------------------
Etiqueta1    dato1,1 dato1,2 ......... dato1,N
Etiqueta2    dato2,1 dato2,2 ......... dato1,N
.........    ..... ......  ......... ......
EtiquetaM    datoM,1 datoM.2 ......... datoM,N
-------------------------------------------
```

donde:

- M: es el número de muestras
- N: es el tamaño de las muestras
- <Etiqueta$_M$> es la cadena que identifica a la *m-ésima* muestra
- <dato$_{MN}$> es el *n-ésimo* componente del *m-ésimo* vector muestra

Los datos pueden ser números enteros o reales. En este último caso, es obligatorio que el separador de decimales sea el punto (**.**), no la coma (**,**). En ambos casos no debe de existir separador de miles.

Este formato también es muy fácil de interpretar con las librerías de Java, dando lugar a que se disponga de 2 diferentes entradas para las muestras.

A su vez, en lo que al formato de archivos de *red* se refiere, se ha determinado que el mismo debe tener, al menos, los siguientes parámetros:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Kohonen>
    <parametros>
        <capa_entrada>100</capa_entrada>
        <lado_Kohonen>4</lado_Kohonen>
        <normalizacion>E</normalizacion>
        <eta_inicial>0.25</eta_inicial>
        <eta_final>0.01</eta_final>
        <periodo>500</periodo>
        <curva>LINEAL</curva>
        <refuerzo>NO</refuerzo>
        <refuerzos>0</refuerzos>
        <compresion_eta>0.0</compresion_eta>
        <extension_periodo>0</extension_periodo>
        <vecindario_inicial>4</vecindario_inicial>
        <presentacion_muestras>SECUENCIAL</presentacion_muestras>
    </parametros>
```

Se puede ver claramente que los parámetros hacen referencia a todos los posibles parámetros que puede tener una red de Kohonen. Es importante destacar que el parámetro de *normalización* puede tener los valores de 'E' (euclídea) o 'N' (ninguna), el parámetro *curva* puede tener los valores LINEAL o EXPONENCIAL, el parámetro *refuerzos* puede tener los valores SI o NO, y la *presentación de muestras* puede ser SECUENCIAL o ALEATORIA. El resto de parámetros deben ser enteros o reales en base al tipo de variable al que hacen referencia que se ha explicado anteriormente.

En el caso de que la red haya sido entrenada, se guardará también un objeto en el mismo fichero que contendrá la estructura de la red:

```
<pesos>
        <peso e="0" nx="0" ny="0">0.07650840963841876</peso>
              .
          .
          .
</pesos>
</Kohonen>
```

Cada uno de los pesos tiene 3 atributos: *e* que hace referencia a la neurona de entrada (habrán tantas como dimensiones de las muestras), *nx* que hace referencia a la columna de la matriz de la capa de Kohonen en la que se encuentra la neurona de Kohonen que se conecta a *e*, y *ny* que hace referencia a la fila de la matriz de Kohonen en la que se encuentra la neurona de Kohonen que se conecta a *e*.

Finalmente se ha determinado necesario que se puedan guardar los resultados de la clasificación en un fichero, para lo cual se ha ideado el siguiente formato:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<clasificacion>
    <datos_red>
        <lado_Kohonen>7</lado_Kohonen>
    </datos_red>
    <resultado>
        <muestra>
            <etiqueta>001FL1B1-1</etiqueta>
            <fila>0</fila>
            <columna>4</columna>
        </muestra>
        <muestra>
          .
          .
          .
          </muestra>
    </resultado>
</clasificacion>
```

Este es un fichero generado por la herramienta de Kohonen que muestra los resultados de la clasificación especificando la etiqueta de la muestra, y la neurona activada indicando la fila y la columna de la matriz de la capa de Kohonen.

Una vez definida la estructura entre los ficheros, es importante que exista una relación entre los mismos. De esta forma, mediante una muestra se puede entrenar una red, y con esa red entrenada se puede generar una clasificación. Se ha determinado que una muestra tiene un nombre, una red tiene otro nombre, y la clasificación se genera a partir de ambos nombres:

- *Muestra:* MUESTRA.XML
- *Red entrenada:* MUESTRA_RED.XML (la red "RED" fue entrenada con la muestra "MUESTRA").
- *Clasificación:* MUESTRA_RED_CLS.XML (la muestra "MUESTRA" fue clasificada con la red "RED").

En definitiva, la relación entre los ficheros se realiza a través de sus nombres.

Tras haber definido la estructura de los ficheros y la relación de los mismos, se ha pasado a la implementación de las funciones que leen y escriben en los mismos. La mayoría de estas funciones se encuentran en la clase *XML_PARSER.java* (que hace referencia al módulo de Parser de Archivos), pero también hay algunas funciones de escritura que han sido incluidas directamente en la Herramienta de Kohonen (*KOHONEN.java)*.

XML_PARSER.java

A continuación se definen las funciones que han sido utilizadas en la clase *XML_PARSER.java* para la lectura y escritura de ficheros:

- **`public List<MUESTRA> lectura_muestras (String nombre_fichero):`**

  Recibe el nombre y ubicación del fichero de entrada de muestras y pasa a su lectura. Devuelve una lista con todas las muestras que ha encontrado.

- **`public int LeerMuestrasTXT(String nombre_fichero, String directorio):`**

  Recibe el nombre de un fichero .DAT de muestras y la ubicación del mismo. Devuelve un código de *OK* en caso de que no haya ningún problema y guarda en la ubicación dada el mismo archivo con formato XML. En caso de haber un error en la estructura del fichero, devuelve el código correspondiente. En resumen, transforma un fichero .DAT a uno .XML de muestras.

- **`public void Salvar_Kohonen(String nombre_fichero, String directorio, KOHONEN red):`**

  Recibe el nombre de fichero y la ubicación en donde se desea guardar una red así como la red en cuestión a guardar. Permite guardar dicha red con todos sus parámetros siguiendo la estructura de fichero de red .XML

- **`public void Salvar_Clasificacion(String nombre_fichero, List<RESULTADOS_CLASIFICACION> resultados, int lado_Kohonen):`**

  Permite guardar en un fichero los resultados de una clasificación. Recibe el nombre del fichero con el cual se desea guardar, una lista con los resultados, y el lado de Kohonen utilizado.

KOHONEN.java

A continuación se definen las funciones que han sido utilizadas en la clase *KOHONEN.java* para la lectura y escritura de ficheros:

- **`public KOHONEN(String nombre_fichero, String directorio):`**

  Recibe como parámetros un fichero de red XML y el directorio del mismo. De esta forma, el constructor de la clase Kohonen construye el objeto en base a los parámetros encontrados en el fichero. En caso de la red estar entrenada, se cargan los pesos del fichero.

- **`public void cargar_pesos(String nombre_fichero, String directorio):`**

  Recibe como parámetros un fichero de red XML y el directorio del mismo. Busca en ese fichero pesos de la red, y en caso de existir, carga dichos pesos en la red ya construida.

6.4.2.4 PLANIFICACIÓN, PROBLEMAS Y PRUEBAS

Una vez conseguido que se cargasen todos los ficheros correctamente, y que se guardase la información en los mismos, se ha procedido a comprobar la exactitud de la clasificación de la herramienta con diferentes parámetros que se cargaban desde ficheros. Se han utilizado muestras y resultados esperados que ha proporcionado el tutor, y la herramienta ha podido clasificar dichos datos correctamente.

También se ha comprobado que una red pudiera cargar los pesos de un fichero y utilizarse dichos pesos como iniciales para el entrenamiento y posterior clasificación de las muestras.

6.5 DESARROLLO DE LA INTERFAZ GRAFICA Y LA HERRAMIENTA DE SUBIDA Y DESCARGA

Una vez terminada la Herramienta de Kohonen, se ha procedido al diseño e implementación de la interfaz gráfica y a la posibilidad de subir los ficheros desde el explorador.

6.5.1 TERCER INCREMENTO

Este incremento ha consistido en el diseño de la interfaz gráfica que se va a utilizar. Para ello se han tenido en cuenta los requisitos que hacían referencia a la misma, en la que se han pedido 3 ventanas distintas en función de si se quería entrenar, clasificar o visualizar la red de Kohonen.

6.5.1.1 OBEJTIVOS PRINCIPALES

- Diseñar una interfaz amigable para el usuario que permita el entrenamiento, clasificación y visualización.

6.5.1.2 ANÁLISIS DE RIESGOS

- Posibilidad de que el diseño no sea el más adecuado para la herramienta, para lo cual se han considerado diferentes posibilidades en el mismos y se ha llegado al que más usabilidad ofrecía.

6.5.1.3 EJECUCIÓN

En este incremento lo primero que se ha hecho es un boceto de la idea que se buscaría implementar más adelante. El resultado inicial ha sido el siguiente:

Figura 40: Diseño inicial de la interfaz gráfica

Una vez definida la estructura, se ha pasado a la implementación de dicha interfaz gráfica en HTML, CSS, y Javascript. Los resultados de dicha implementación han sido los siguientes:

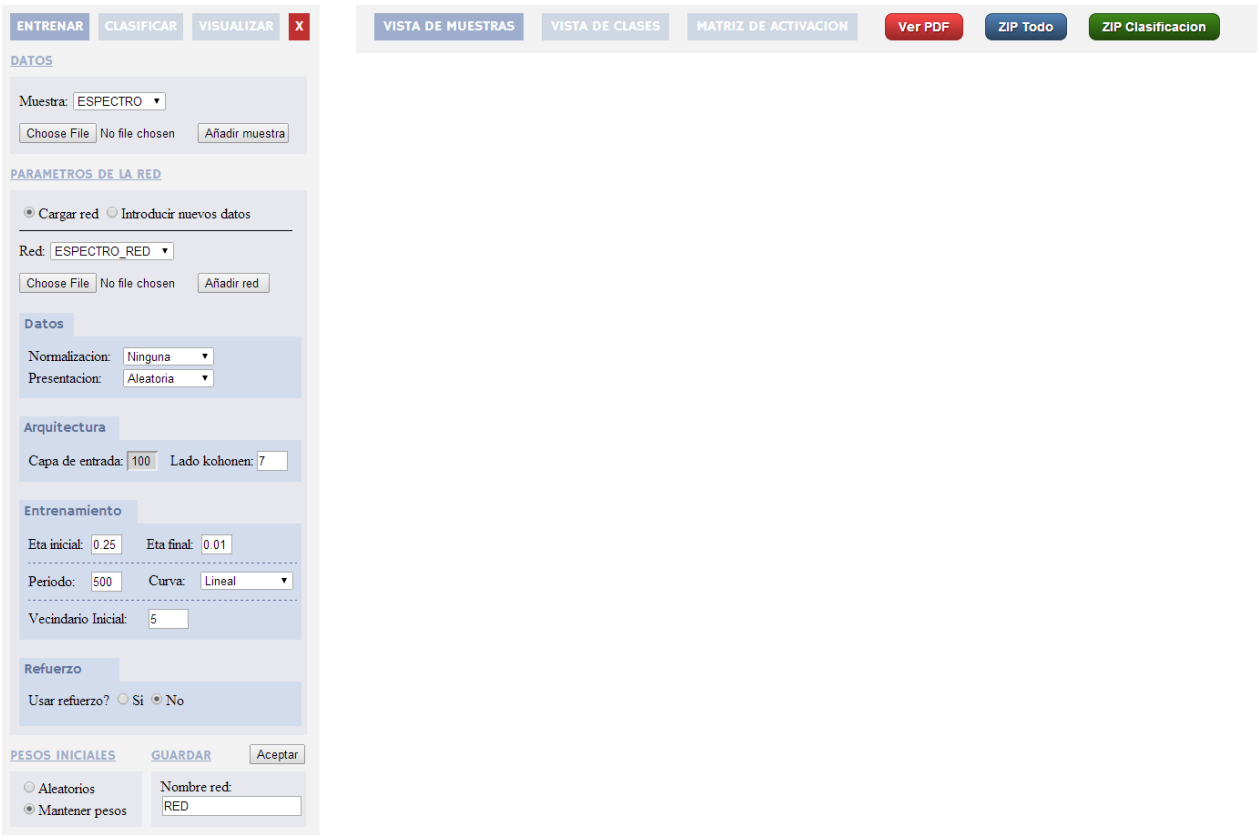

Figura 41: Diseño final de la pestaña Entrenar

Se puede observar en la figura 44 la pestaña de Entrenar, en la que el botón de "Entrenar" se encuentra más oscuro indicando la pestaña seleccionada, luego se ofrece subir datos de muestra, y tras ello se permite o "Cargar red" para subir un fichero de red, o "Introducir nuevos datos" para construir una nueva red desde cero. Independientemente de la selección, el usuario puede seleccionar los parámetros de la red y modificarlos, tras lo cual le dará un nombre a la red el cual será usado como identificador de dicha red.

A su vez, se puede observar el menú de visualización, en el cual se permiten ver 3 formas distintas de visualización, y el menú de descarga, en el cual se le permite al usuario descargar un informe en PDF de los resultados de la clasificación, un fichero ZIP con todos los ficheros de la carpeta de sesión del usuario, y un fichero ZIP con sólo los ficheros correspondientes a la clasificación actual.

Cabe destacar que actualmente esto es una interfaz gráfica independiente de la aplicación, que posteriormente habrá de ser conectada a la misma.

Los resultados de la interfaz gráfica para las pestañas "Clasificar" y "Visualizar" han sido muy parecidos, sólo que no se dispone de la modificación de parámetros de la red.

6.5.1.4 PLANIFICACIÓN, PROBLEMAS Y PRUEBAS

Una vez finalizado el diseño en HTML, CSS y Javascript de la interfaz gráfica, se ha de proceder en los siguientes incrementos a la implementación de la misma en la herramienta, de forma que sea posible acceder a la Herramienta de Kohonen desde un explorador web.

No ha habido problemas a destacar en el diseño y programación de esta interfaz.

6.5.2 CUARTO INCREMENTO

Tras el diseño de la interfaz gráfica, se ha procedido a conectarla con la herramienta y a generar el módulo de subida y descarga de archivos.

6.5.2.1 OBEJTIVOS PRINCIPALES

- Implementar la interfaz gráfica en JSP.
- Hacer el proyecto una Java Web Application mediante la plataforma Struts.
- Implementar el uso de sesiones en la aplicación y la creación automática de la carpeta de usuario.
- Permitir la carga de archivos desde el explorador.

6.5.2.2 ANÁLISIS DE RIESGOS

- Dificultad a la hora de tratar los datos de la Herramienta de Kohonen con Struts en la interfaz JSP. Frente a esto, se han realizado varias pruebas de Struts para garantizar dicha posibilidad.
- Complicaciones en la carga y descarga de archivos desde el explorador. Frente a esto se ha buscado documentación extensiva al respecto y se ha dado un margen de tiempo aceptable para su implementación.

6.5.2.3 EJECUCIÓN

El proceso para la realización de los objetivos de este incremento ha sido el siguiente:

1. Se han cargado las librerías de Struts en la aplicación.
2. Se ha creado las páginas JSP necesarias para el uso de la herramienta. Estas páginas incluyen *index.jsp* (que sirve de mensaje de bienvenida para el usuario y que ha sido utilizada para crear una carpeta de sesión, cuyo nombre será el ID aleatorio asignado al usuario), *entrenar.jsp* (que muestra la pestaña *Entrenar*), *clasificar.jsp* (que muestra la pestaña *Clasificar* y *visualizar.jsp* (que muestra la pestaña visualizar).
3. Se han personalizado dichas páginas con los resultados de CSS del incremento.
4. Se ha incluido en cada una de las páginas el apartado de subir los ficheros correspondientes, y se ha comprobado que se guarden en la carpeta de sesión del usuario. Para ello, se han utilizado formularios que permiten la subida de ficheros. Cada formulario está vinculado a una "acción" (basado en el diseño modelo-vista-controlador, la acción sería el controlador), que es una clase java encargada de realizar todas las acciones asociadas a dicho formulario. En ese sentido, las 3 acciones implementadas han sido:

- **UploadMuestraAction.java:** permite la subida de un fichero muestra a la carpeta de sesión del usuario y guarda el nombre de dicha muestra en una estructura de datos para poder mostrar el desplegable de muestras en la interfaz gráfica. En caso de haber algún error (formato de fichero, etc) envía un mensaje de error que será mostrado al usuario.
- **UploadRedAction.java**: permite la subida de un fichero red a la carpeta de sesión del usuario y guarda el nombre de la red en una estructura de datos para su manipulación y para mostrarlo al usuario. En caso de haber algún error en el fichero, envía un mensaje que será mostrado al usuario.
- **UploadViewAction.java:** permite la subida de un fichero de visualización de resultados a la carpeta de sesión del usuario. Aún no se ha desarrollado la estructura de este tipo de ficheros, por lo que se ha detenido por ahora la implementación de esta acción.

5. Para la gestión de la sesión de usuarios, se han creado las siguientes acciones:
   - **CreateSession.java:** comprueba si hay alguna sesión activa y en caso de no ser así, se crea una sesión y una carpeta asociada a la misma.
   - **CloseSession.java:** esta función es llamada tras un período de inactividad en la sesión de un usuario o cuando el usuario desea terminar su sesión. Se encarga de cerrar la sesión y destruir la carpeta asociada a la misma.

6.5.2.4 PLANIFICACIÓN, PROBLEMAS Y PRUEBAS

Debido a que aún no han sido implementadas las acciones encargadas de entrenar una red o clasificar una muestra desde la interfaz, todavía no se ha implementado la parte de la herramienta de descargar los archivos de la carpeta de sesión.

Dentro de lo que sí se ha implementado, se han realizado todas las pruebas correspondientes a la subida de ficheros y correcto funcionamiento de las sesiones de usuario.

6.6 PERMITIR LA MODIFICACIÓN DE PARÁMETROS DE RED Y EJECUCIÓN DE LA HERRAMIENTA DE KOHONEN DESDE EL EXPLORADOR

6.6.1 QUINTO INCREMENTO

Una vez que se ha conseguido una interacción entre la interfaz y la aplicación Java (mediante la subida de ficheros desde la página JSP), se ha de continuar dicha interacción consiguiendo la ejecución de la herramienta de Kohonen.

6.6.1.1 OBEJTIVOS PRINCIPALES

- Permitir cargar los parámetros de una red en la pestaña de *Entrenar* una vez subido un fichero de red. En el caso de seleccionar la introducción de parámetros nuevos, se han de cargar por defecto los valores predeterminados en la clase *VALORESDEFINIDOS.java*.
- Poder realizar la ejecución de la herramienta para entrenar o clasificar desde las páginas JSP una vez validados los parámetros introducidos en dicha página.
- Guardar los ficheros resultantes del entrenamiento y la clasificación en la carpeta de sesión destinada a ello.

6.6.1.2 ANÁLISIS DE RIESGOS

- Dificultad a la hora de pasar los valores desde la página JSP a la Herramienta de Kohonen. Para ello se ha conseguido documentación extensiva al respecto y se ha dejado el margen de tiempo suficiente para resolver cualquier problema que pueda surgir.

6.6.1.3 EJECUCIÓN

En primer lugar, tras subir un fichero de red desde la pestaña *Entrenar*, ha sido necesario mostrar los parámetros de la red contenida en ese fichero en el explorador. Para ello, una vez cargado el fichero mediante la acción *UploadRedAction.java*, se ha generado una lista de redes Kohonen que se pasa como atributo a la página JSP, cada red siendo una de las subidas mediante fichero o creadas por el usuario. A su vez, dichas redes guardan todas las propiedades del fichero mediante la cual fueron creadas, por lo que simplemente habría que seleccionar dichos atributos e imprimirlos en la página JSP cada vez que se sube un fichero.

Como existe un desplegable mediante el cual se pueden seleccionar redes cargadas anteriormente, era necesario que en el momento de seleccionar alguna, se actualizaran los campos habilitados para los parámetros de la red en la interfaz gráfica. Puesto que no tenía sentido que se tuviera que volver a cargar la página y enviar toda la información desde el servidor tras cada selección, se ha optado por guardar todos los valores de las redes cargadas hasta el momento mediante Javascript, de forma que existe un objeto red de Javascript en la página *entrenar.jsp* que contiene los mismos atributos que el objeto KOHONEN.java. Dichos objetos han sido guardados en una lista, de forma que a la hora de seleccionar un elemento del desplegable, se buscaran los parámetros en la lista y se mostrasen automáticamente. De esta manera, la lista de redes manejadas por Struts es la misma que se carga en la página JSP y que es usada mediante Javascript.

El único atributo extra existente en la versión del objeto red en Javascript es el de la etiqueta de la red, ya que es el nombre de la misma y se debe cargar también en el campo habilitado puesto que será necesario para la generación del nombre del fichero XML de red.

En el caso de no subir un fichero de red (sino que se selecciona la opción de introducir nuevos datos), se cargan también mediante Javascript los valores por defecto de la nueva red en los campos habilitados para ello. El nombre por defecto de una red es "RED".

Una vez completado todo esto, se ha procedido a la implementación de las acciones encargadas de llamar a los métodos de entrenar una red y clasificar una muestra. Acciones que serán llamadas dependiendo de la página JSP en la que el usuario se encuentre tras validarse los parámetros introducidos por el mismo. Estas acciones son:

- **EntrenarAction.java:** es llamada al hacer click en el botón *Aceptar* en la pestaña *Entrenar*. Utiliza los parámetros establecidos en la página para el entrenamiento y el nombre de la red para generar el archivo correspondiente de entrenamiento. Si el nombre de la red es el mismo que una red ya existente, se sobre escribe la misma (al igual que su fichero). En caso de haber indicado un nombre nuevo, se introduce una nueva red con dichos parámetros en la lista de redes.
- **ClasificarAction.java:** es llamada al hacer click en el botón *Clasificar* en la pestaña *Clasificar*. En base a la muestra y red seleccionada, llama a la herramienta de Kohonen para realizar la clasificación y generar el archivo de resultados de clasificación correspondiente. Dicho archivo de resultados se guarda en la carpeta del usuario.

6.6.1.4 PLANIFICACIÓN, PROBLEMAS Y PRUEBAS

No se ha encontrado ningún problema con la llamada a las acciones y la gestión de los mismos. Se ejecuta perfectamente la Herramienta de Kohonen desde el explorador en función a los parámetros indicados por los ficheros y el usuario.

En los siguientes incrementos ha de realizarse la visualización gráfica de los resultados y los enlaces de descarga.

6.7 VISUALIZACIÓN GRAFICA DE RESULTADOS, INFORMES PDF Y DESCARGA DE RESULTADOS

6.7.1 SEXTO INCREMENTO

Hasta ahora se ha conseguido el correcto funcionamiento de la Herramienta de Kohonen desde el explorador, la validación de los parámetros, la subida de ficheros al servidor, y la generación de ficheros de resultados desde la herramienta que también son guardados en el servidor.

Tras esto se ha llegado a la parte más significativa del proyecto: la visualización gráfica de los resultados del entrenamiento y clasificación. Todo el contenido mostrado a continuación se ha incluido en un sólo incremento más largo debido a lo relacionadas que estaban todas las acciones a realizar.

6.6.1.1 OBEJTIVOS PRINCIPALES

- Generar los 3 tipos de visualizaciones previstas con D3.JS: *vista de muestras, vista de clases y vista de matriz de activación*.
- Garantizar que la visualización sea representativa de los resultados.
- Hacer que la visualización se realice a partir de ficheros generados por la herramienta.

6.6.1.2 ANÁLISIS DE RIESGOS

- Complicaciones en la visualización de datos debido a la inexperiencia. Ante esto, se han buscado varios ejemplos sobre los que partir para generar los resultados.
- Posibilidad de que los resultados de la visualización no sean los esperados. Ante esto se han considerado otras posibles librerías Javascript para visualización de datos. Afortunadamente, no ha hecho falta utilizarlas.

6.6.1.3 EJECUCIÓN

El proceso de generar la visualización de los datos ha sido un poco complicado. A su vez, explicarlo detalladamente no es sencillo puesto que requeriría mostrar directamente el código fuente. Es por ello que a continuación se resumen las características principales del proceso, pero para más detalle se ruega ver las páginas *entrenar.jsp, clasificar.jsp y visualizar.jsp* que contienen el código fuente de la visualización.

Fichero de visualización

Para la visualización de los resultados se ha utilizado el módulo de *Cluster Force Layout de D3.JS*. Este consiste en una serie de nodos que se pueden conectar entre sí, y todos se ven

afectados por una serie de fuerzas entre las que destacan la gravedad y la fuerza de atracción entre los nodos.

Como la visualización ha de ser dinámica en función de los resultados de la clasificación, se ha idea un sistema mediante el cual tras hacerse una clasificación, se generase automáticamente un fichero con todo el contenido de la visualización. Debido a la compatibilidad de *D3.JS* con el formato de ficheros JSON, se ha escogido este formato.

En ese sentido, se ha mantenido la relación entre los ficheros mediante el nombre: tras hacerse una clasificación, automáticamente se ha de llamar a un módulo generador de ficheros JSON que crea el archivo JSON con todas las características que necesita *D3.JS* para generar la visualización. El nombre del fichero será *MUESTRA_RED_VIEW.json* en el que MUESTRA representa la muestra utilizada para ser clasificada, y RED representa la red ya entrenada utilizada para clasificar la muestra.

A continuación se resumen las características más importantes contenidas en el fichero para generar los 3 tipos de visualización:

- Se ha determinado la necesidad de incluir 2 tipos de nodos:
    - Un nodo *class*, el cual es un nodo grande que representa una neurona de la capa de Kohonen.
    - Un nodo *muestra*, que representa una de las muestras a ser clasificadas.

- Todo nodo muestra debe estar conectado a un nodo class (y sólo a uno), esto representa que esa muestra ha activado una neurona en particular. Todos los nodos independientemente de su tipo contienen las siguientes características:

    - *id:* es la etiqueta mediante la cual se identifica el nodo. Los nodos *class* tienen una *id* de la forma "Clase X" en donde la X representa un contador. Los nodos *muestra* contienen un *id* con la etiqueta que ha sido recogida del fichero de muestras.
    - *type*: sirve para identificar el tipo de nodo. Puede ser *class* o *muestra*.
    - *parent:* indica el nodo padre. Todos los nodos *class* con padres de sí mismos y los nodos *muestra* son hijos de un padre tipo *class.* Este valor es un entero que representa el número del nodo padre.
    - *fila:* representa la fila en la matriz de la capa de Kohonen del nodo padre.
    - *columna:* representa la columna en la matriz de la capa de Kohonen del nodo padre.

    A su vez, los nodos *class* tienen además las siguientes propiedades:

    - *distancia_media:* indica la distancia media de la clase.
    - *num_muestras:* indica el número de muestras conectadas a ese nodo.

    Finalmente, los nodos *muestra* tienen a su vez las siguientes propiedades:

    - *distancia:* indica la distancia de la muestra a la clase que la ha activado.

- Además de los nodos, ha sido necesario generar los enlaces que conectan dichos nodos, también llamados *links*. Estos enlaces son las líneas (y la distancia) que separa a un nodo de otro, y son necesarios ya que de lo contrario todos los nodos estarían unidos. Se ha determinado que estos enlaces deben tener las siguientes propiedades:

    - *source:* es el nodo origen. Su valor es un entero que representa el contador de nodos para el nodo en cuestión.

    - *target:* es el nodo destino. Su valor es un entero que representa el contador de nodos para el nodo en cuestión.

    - *distance:* indica la distancia entre el origen y el destino.

    - *value:* puede ser "visible" o "invisible". En caso de ser visible, se muestra una línea que conecta a un nodo y otro. En caso de ser invisible, simplemente se representa la distancia.

  Todos los nodos de clase se conectan entre sí mediante un enlace invisible. Esto permite que las distancias entre las clases sea representativa y equivalente a la distancia de las neuronas de la matriz de activación.

- Todas las propiedades mencionadas hasta ahora son utilizadas para generar la vista de clases y muestras. Por otro lado, la vista del mapa de activación de la red necesita de la siguientes propiedades:

    - *lado_Kohonen*: valor que indica el lado de Kohonen de la red utilizado para la clasificación.

    - *max_muestras:* el máximo número de muestras que ha activado una clase. Esto se ha usado entre otras cosas para el rango de colores que tienen las diferentes posiciones de la matriz.

    - *min_muestras:* el mínimo número de muestras que ha activado una clase. Al igual que el valor anterior, este valor es usado entre otras cosas para el rango de colores que tienen las diferentes posiciones de la matriz.

    - *distancia_mapa:* es la media de las distancias de las calses.

    - *nombre_final:* es el nombre del fichero de clasificación que acompaña a este fichero de visualización al clasificar una muestra.

    - *matriz:* es un array de objetos que representan cada una de las posiciones de la matriz de activación. Cada uno de estos objetos tiene las siguientes propiedades:

        - *fila:* indica la posición de la fila que representa dicho objeto en la matriz de activación.

        - *columna:* indica la posición de la columna que representa dicho objeto en la matriz de activación.

- *id:* es un identificador cuyo valor corresponde con el identificador de las clases en la vista de muestras y clases.
- *numero_clases:* indica el número de muestras que han sido activadas para esta clase.
- *distancia_media:* representa la distancia media de la clase.

A su vez, en función del número de muestras asociadas a cada posición de la matriz se tendrá también las siguientes propiedades:

- *etiquetaX:* representa la etiqueta de la muestra X asociada a dicha posición. X es un valor entero.
- *distanciaX:* representa la distancia de la muestra X a la clase asociada a dicha posición. X es un valor entero.

Esto concluye las características que debe tener el fichero de visualización. Una definida esta estructura se ha pasado al desarrollo del *Generador de JSONs*, el cual contiene la función encargada de generar este fichero.

Generador de JSONs

Debido a que el generador de JSONs es únicamente una función encargada de generar todo el fichero de visualización, esta función ha sido incluida en la clase *XML_PASER.java*. La función es:

- **`public void JSONInforme(List<CLASES_INFORME> informe, String nombre_archivo String directorio, int num_muestras):`**

  Es una función que recibe una lista con los resultados de la clasificación, el nombre del archivo a generar, el directorio donde se hade guardar (la carpeta del usuario), y el número de muestras que han sido clasificadas. Con todo esto, genera el archivo de visualización con la estructura explicada anteriormente.

Esta función es llamada cada vez que se clasifica una red y genera el fichero de visualización, cuyo nombre es enviado a la interfaz gráfica para que cargue los resultados de dicha clasificación y los muestre utilizando la herramienta D3.JS.

Vista de muestras y clases

La vista de clases y muestras es muy similar. En un principio se ha hecho la vista de clases y posteriormente se ha modificado para que sólo se mostrasen las muestras. La idea es la siguiente:

- Primero se mira si se ha recibido un parámetro con el nombre del fichero de visualización, y de ser así carga los resultados.
- Se crea el contenedor *svg* que contendrá todos los nodos.
- Se determinan los parámetros de anchura, altura, radio de los nodos (se aumentará o disminuirá dependiendo del tipo), gravedad, distancia, carga y escala a utilizar. Todos estos parámetros vienen por defecto y no hace falta modificarlos.

- En caso de haber una representación anterior, se eliminan todos los nodos.
- Se leen los nodos del array de nodos del archivo de visualización y se añaden al componente *force* de *D3.JS*
- Se leen los enlaces y también se añaden. Dependiendo del tipo de enlace (visible o invisble), se le añade una clase en la que se muestren o se oculten las líneas que unen los nodos.
- Se dibujan los nodos.
- Se añaden las características de los nodos como un *tooltip*, de forma que cuando se ponga el ratón sobre ellos se muestren.
- Se les añade el *id* a cada nodo y se crea un evento mediante el cual dependiendo de la posición del nodo dicha *id* aparezca en un cuadrante u otro.
- Se crean los bordes del *svg* y se impide que los nodos se salgan del mismo.
- Si se selecciona vista de muestras, los nodos visibles se hacen invisibles y se ocultan los nodos tipo *class*.

A continuación se muestran los resultados de ambos tipos de visualización para una clasificación de 4 clases:

Figura 42: Visualización de clasificación de muestras en 4 clases (vista de muestras)

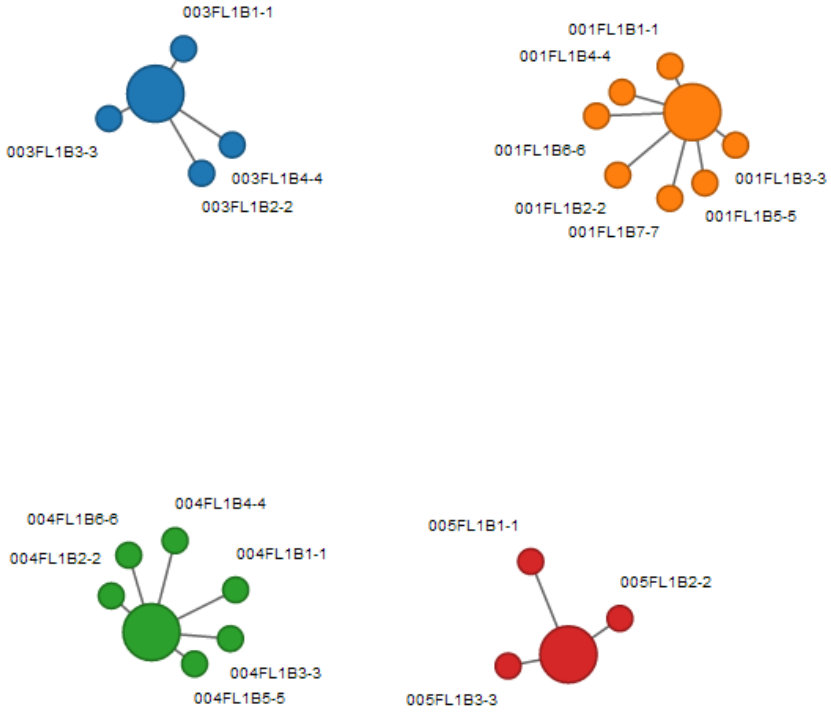

Figura 43: Visualización de clasificación de muestras en 4 clases (vista de clases)

Mapa de activación

El mapa de activación permite tener una perspectiva global de la clasificación y de las diferentes muestras. Muestra la matriz de activación con todas las muestras que han sido activadas en cada casilla. El proceso para realizar dicho mapa ha sido el siguiente:

- Se dibuja un *svg* con la anchura y altura predeterminadas. En este caso su tamaño cambiará en función del lado de Kohonen. La fórmula para calcularlo ha sido: *lado_Kohonen*28*(24/lado_Kohonen)*

- Se indica la anchura y altura de cada fila de la matriz. En este caso es dinámico dependiendo del lado de Kohonen. La fórmula para calcularlo ha sido *(anchura/lado_Kohonen)-10* y *(altura/lado_Kohonen)-10* respectivamente.

- Se asocia una posición del array matriz del fichero de visualización a cada elemento de la matriz que se ha creado.

- Se colorea el fondo de cada posición en función del número de muestras asociadas a la clase que representa.

- Se crea un *tooltip* con toda la información relevante a cada posición de la matriz.

A continuación se muestra el resultado del mapa de activación del resultado anterior y otra clasificación con una lado de Kohonen más grande:

Figura 44: Mapa de activación del ejemplo anterior (lado de Kohonen = 4)

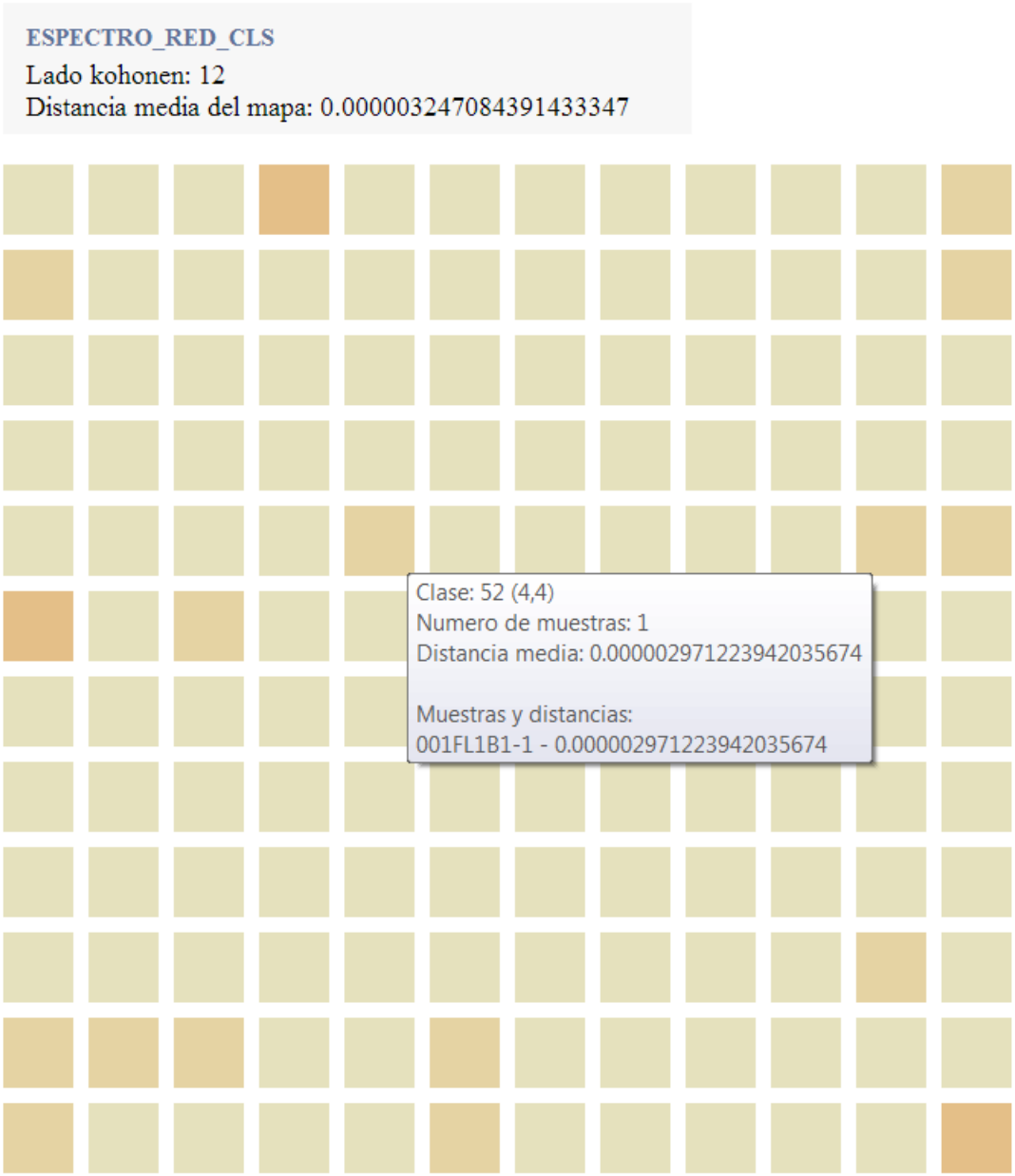

Figura 45: Mapa de activación para lado de Kohonen = 12

Una vez conseguida una correcta visualización de los resultados, se ha pasado a realizar el último módulo de la interfaz: *Visualizar*. El sistema ha sido sencillo ya que se había hecho casi todo. Simplemente ha consistido en una acción que permite subir ficheros JSON, validar su estructura y si todo es correcto, cargar los resultados de la visualización. La herramienta guarda una lista de ficheros de visualización utilizados para que sea fácil volver a cargarlos. Las acciones utilizada para esto son:

- **UploadViewAction.java:** permite subir un fichero de visualización JSON y validarlo para ver si cumple con los parámetros necesarios.
- **VisualizarAction.java:** envía el nombre del archivo de visualización cargado para ver los resultados de clasificación contenidos en el mismo. En el caso de no haberse cargado antes, se añade la visualización a la lista de ficheros.

6.7.1.4 PLANIFICACIÓN, PROBLEMAS Y PRUEBAS

La realización de las pruebas en la visualización de datos ha sido exhaustiva: desde el correcto posicionamiento de las etiquetas, la correcta generación de documentos, las distancias entre los nodos (sobre todo cuando eran muchos), el mejor uso de colores para la representación, la correcta lectura de los documentos, etc.

Todos los problemas con dichos detalles han sido resueltos satisfactoriamente. En los próximos incrementos se ha de generar los informes PDF y los enlaces de descarga para poder descargar todos los documentos de la carpeta de sesión de usuario.

6.7.2 SÉPTIMO INCREMENTO

En este incremento se ha buscado generar los informes PDF y permitir las descargas especificadas en los requisitos de usuario. Dichas descargas incluyen los informes, un ZIP con los resultados de una clasificación en particular, y un ZIP con todos los ficheros de la carpeta de sesión.

6.6.2.1 OBEJTIVOS PRINCIPALES

- Generar un informe PDF tras una clasificación que recoja un resumen de los resultados.
- Generar la posibilidad de descargar un fichero ZIP con los resultados de una clasificación.
- Generar la posibilidad de descargar un fichero ZIP que contenga todos los ficheros de la carpeta de sesión.
- Generar los enlaces de descarga para el informe PDF y los ficheros ZIP.

6.6.2.2 ANÁLISIS DE RIESGOS

- Posibilidad de no encontrar una librería de fácil uso para la generación de archivos PDF y ZIP. Ante esto se han considerado varias alternativas y se ha leído documentación extensiva al respecto.

6.6.2.3 EJECUCIÓN

La clase encargada de generar el fichero PDF es *GeneraPDF.java*. No se ha explicado esta clase en más detalle debido a la complejidad técnica de sus funciones. Sin embargo, es importante destacar que utiliza la librería *itext* para generar los ficheros PDF y el contenido de los mismos. La única función importante de mencionar es:

- **`public static void Crear(String nombre_fichero_muestra, String nombre_fichero_red, String directorio, List<CLASES_INFORME> informe, KOHONEN red):`**

  Recibe como parámetros los nombres de los ficheros muestra y red utilizados para la clasificación, el directorio de la carpeta de sesión del usuario, la lista con el resultado de la clasificación y el objeto de la red de Kohonen utilizada. De esta forma utiliza toda esta información para generar el fichero PDF y guardarlo bajo el nombre: *MUESTRA_RED_RPT.pdf* en el cual MUESTRA se refiere a la muestra clasificada, RED a la red utilizada para la clasificación, y RPT indicando que es un "report".

Por otro lado, las funciones encargadas de generar los ficheros ZIP se encuentran en la clase *GeneraZIP.java*, siendo la importante a destacar:

- **`public GeneraZIP(String directorio, String muestra, String red, String id):`**

  Esta función es un constructor de la clase, que automáticamente crea los 2 ficheros ZIP (el de clasificación y el de todos los contenidos). Para ello recibe el directorio (la carpeta donde se encuentran las sesiones), el nombre de la muestra utilizada para la clasificación, el nombre de la red utilizada para la clasificación y el id del usuario.

  Esta función crea:

  - El fichero comprimido llamado *MUESTRA_RED_ZIP.zip* que contiene:
    - *MUESTRA.XML:* el fichero de datos con el nombre de muestra recibido por parámetros.
    - *MUESTRA_RED.XML:* el fichero de red con el nombre de la red y la muestra utilizada para entrenarla.
    - *MUESTRA_RED_CLS.XML:* el fichero con los resultados de la clasificación.
    - *MUESTRA_RED_VIEW.json:* el fichero que contiene la información para cargar de nuevo una visualización en la herramienta.
    - *MUESTRA_RED_RPT.pdf*: el fichero PDF con un resumen de la clasificación.
  - El fichero comprimido llamado *ID_USUARIO.zip* que contiene todos los contenidos de la carpeta del usuario.

Tanto la generación de PDF como la de los ficheros ZIP ocurre cada vez que se entrena, clasifica o visualiza en la herramienta. De esta manera, se garantiza que sean las versiones más actualizadas. En caso de que ya exista un fichero con el mismo nombre, el más antiguo es eliminado.

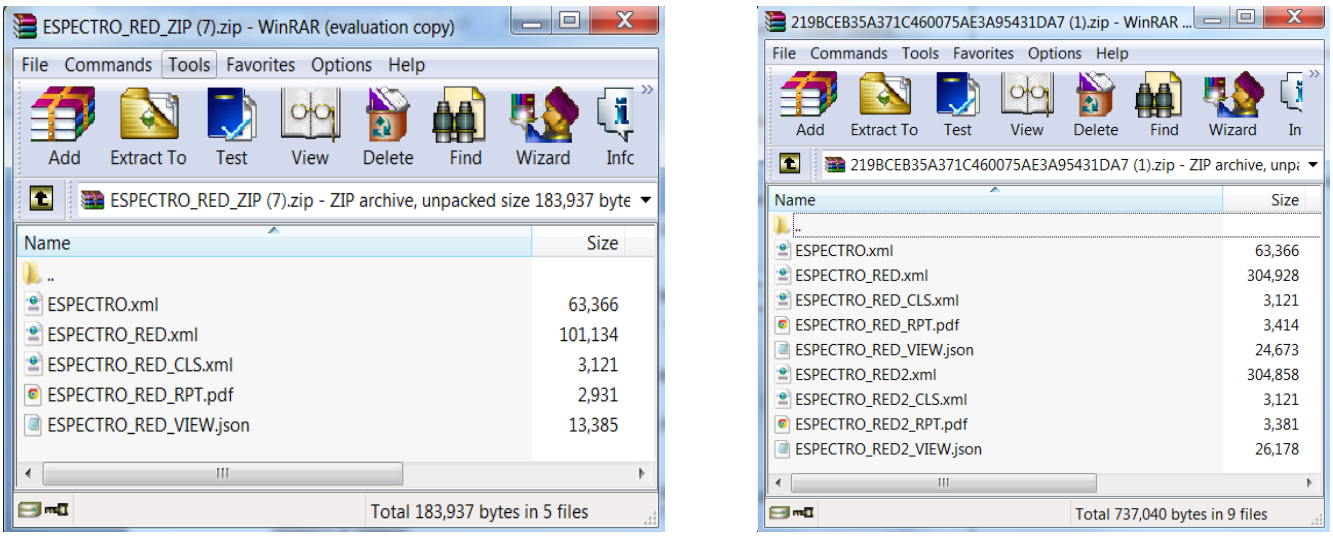

Figura 46: Contenido de los ficheros ZIP generados

Informe de Clasificación **21-06-2014**

Parametros de la clasificacion:

Ficheros relacionados:
- Fichero de DATOS: ESPECTRO
- Fichero de RED: ESPECTRO_RED
- Fichero de CLASES: ESPECTRO_RED_CLS
- Fichero de VISTA: ESPECTRO_RED_VIEW

Datos de entrada:
- Numero de ejemplos: 20
- Tipo de presentación de muestras: ALEATORIA
- Tipo de normalización de muestras: NINGUNA

Arquitectura de la red:
- Capa de entrada: 100
- Lado de kohonen: 7

Parámetros del entrenamiento:
- ETA inicial: 0.25
- ETA final: 0.01
- Periodo: 500
- Curva: 1
- Vecindario Inicial: 5

Parámetros del refuerzo:
- No se ha utilizado refuerzo

Estadísitcas:

- Numero de clases: 12
- Numero de Activaciones - Numero de Neuronas:
  - 0 - 37
  - 1 - 7
  - 2 - 3
  - 3 - 1
  - 4 - 1

División de clases:

Clase: 0. Neurona (0,0)
- 003FL1B1-1 Distancia: 0.011539274905959952
- 003FL1B4-4 Distancia: 1.2729119144496486E-4

Distancia media de la clase: 0.005833283048702459

Clase: 2. Neurona (0,2)
- 003FL1B3-3 Distancia: 7.134912651036814E-4

Distancia media de la clase: 7.134912651036814E-4

Clase: 5. Neurona (0,5)
- 005FL1B2-2 Distancia: 0.038252610496692846

Distancia media de la clase: 0.038252610496692846

Clase: 6. Neurona (0,6)
- 005FL1B1-1 Distancia: 0.0032703043874042757

Distancia media de la clase: 0.0032703043874042757

Clase: 8. Neurona (1,1)
- 003FL1B2-2 Distancia: 0.0062743592272486454

Distancia media de la clase: 0.0062743592272486454

Clase: 13. Neurona (1,6)
- 005FL1B3-3 Distancia: 0.0341561690216513

Distancia media de la clase: 0.0341561690216513

Clase: 22. Neurona (3,1)
- 004FL1B4-4 Distancia: 0.005308438151291338

Distancia media de la clase: 0.005308438151291338

Clase: 28. Neurona (4,0)
- 004FL1B1-1 Distancia: 8.637679045550533E-5
- 004FL1B6-6 Distancia: 0.0014699621779547358

Distancia media de la clase: 7.781694842051205E-4

Clase: 32. Neurona (4,4)
- 001FL1B1-1 Distancia: 7.955585895255095E-4
- 001FL1B2-2 Distancia: 0.007784150507003015
- 001FL1B3-3 Distancia: 0.004808836741106296
- 001FL1B4-4 Distancia: 0.0047943000150590595

Figura 47: Ejemplo de páginas del informe PDF

Finalmente para descargar los ficheros se ha generado una acción para cada enlace, que se puede encontrar en la barra superior de la interfaz gráfica:

Figura 48: Botones de descarga de la herramienta

Dichas acciones son:

- **LinkInforme.java:** genera un enlace de descarga para el informe PDF.
- **LinkZIP_CLS.java:** genera un enlace de descarga para el fichero ZIP de la clasificación.
- **LinkZIP_TODO.java:** genera un enlace de descarga para el fichero ZIP de la carpeta del usuario.

6.7.2.4 PLANIFICACIÓN, PROBLEMAS Y PRUEBAS

Se ha comprobado la correcta generación de los ficheros y que su contenido fuera el adecuado. A su vez, y siendo el último incremento, se ha aprovechado para realizar pruebas de todo el sistema y de implementar los mensajes de error que podían derivar de los ficheros subidos incorrectamente.

Tras un análisis completo del sistema y pruebas exhaustivas con los datos, se ha concluido que el sistema es funcional y cumple con los requisitos de usuario.

6.8 RESULTADOS OBTENIDOS - EJEMPLO DE EJECUCIÓN

Tras desarrollar todos los componentes necesarios del sistema, se ha conseguido integrarlo de forma satisfactoria. Los resultados obtenidos han sido los esperados: se ha desarrollado una herramienta de clasificación de datos mediante redes de Kohonen, y se permite ver sus resultados de una forma visual y amigable.

Para mostrar los resultados obtenidos, se procede a continuación a mostrar un ejemplo de ejecución de la herramienta, detallando la interfaz gráfica de la misma y el proceso seguido.

Una vez instalada la herramienta (ver Anexo I), se puede acceder a ella desde cualquier explorador web mediante la dirección IP del servidor en donde ha sido instalada. En este ejemplo, se accede desde el mismo servidor, por lo que bastaría con ir a: *http://localhost:8080/PFG/index.jsp*

Se llega a la ventana de bienvenida de la herramienta:

Figura 49: Ventana de bienvenida de la herramienta

En esta ventana se le explica al usuario el funcionamiento de la herramienta y se crea la sesión del mismo. Una vez leídas las instrucciones se debe hacer click en el botón *Entrar*, el cual llevará al usuario a la página principal de la herramienta. Se puede observar que se puede seleccionar el método de trabajo que se desea, disponiendo de las opciones *Entrenar, Clasificar, y Visualizar.*

ENTRENAMIENTO DE UNA RED

A continuación se puede observar la interfaz gráfica para el entrenamiento de la red:

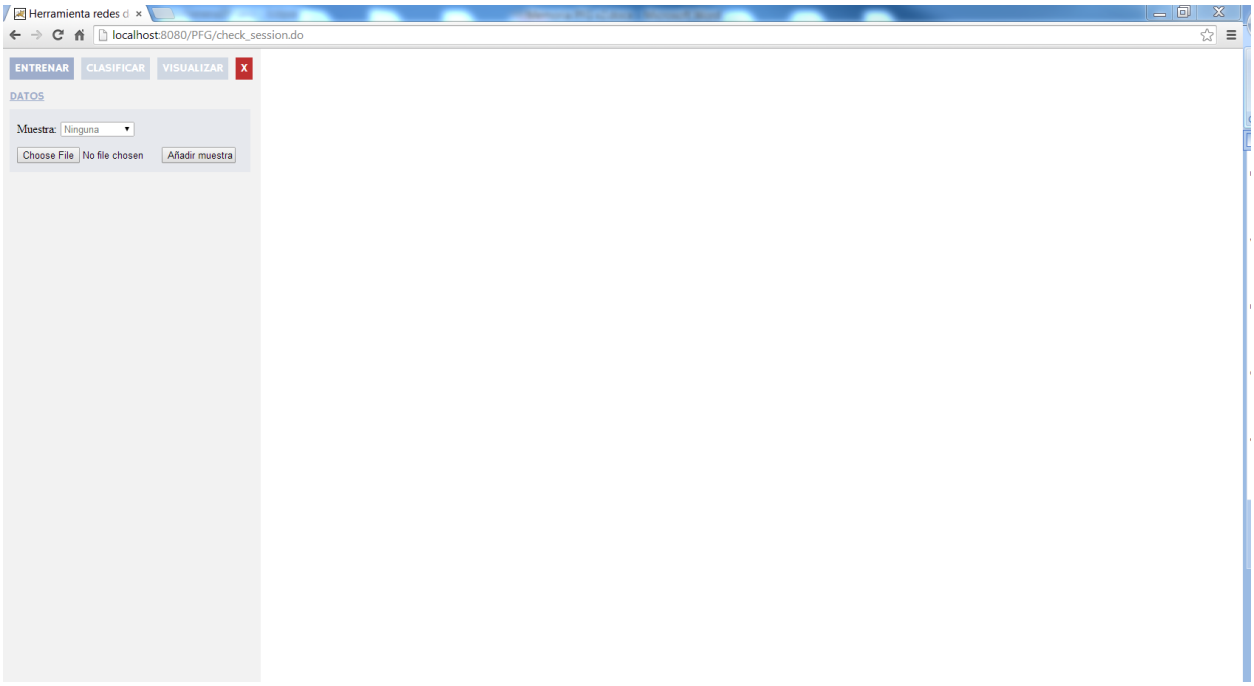

Figura 50: Interfaz gráfica inicial de la herramienta

La siguiente figura explica la barra izquierda de la herramienta, que contiene las opciones disponibles:

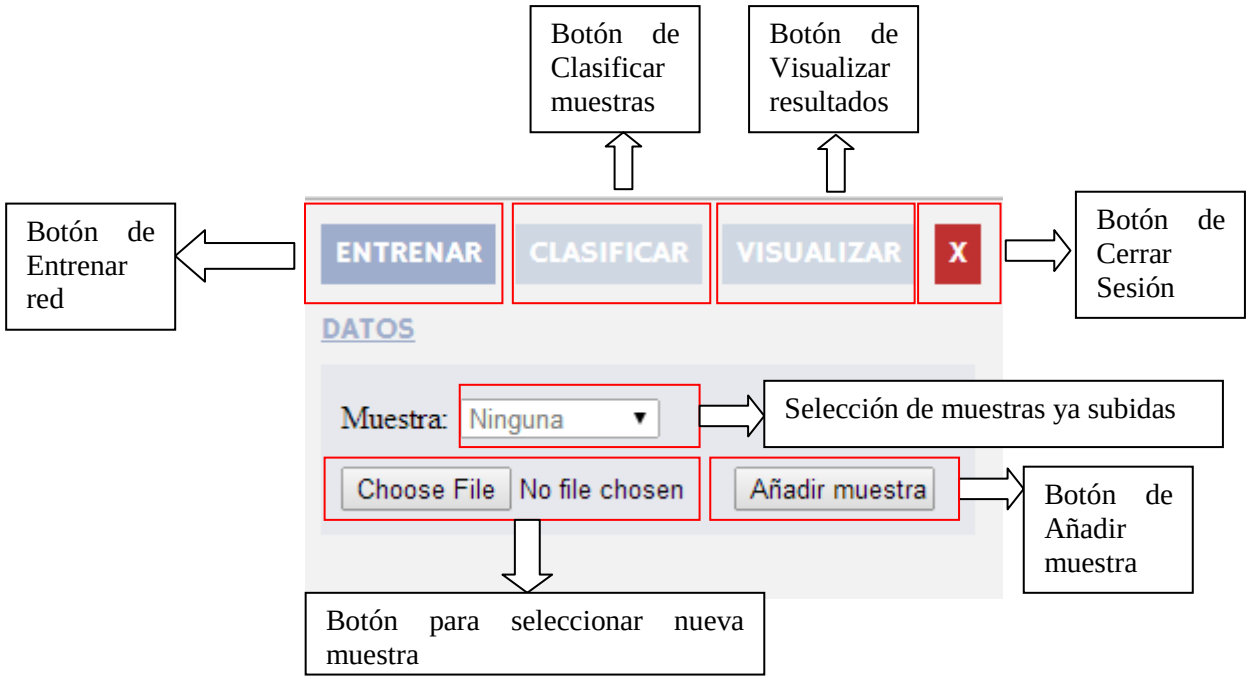

Figura 51: Interfaz Entrenar (1)

Lo primero que se debe hacer para entrenar una red, es seleccionar un fichero de muestras con el cual se desea entrenar la red. Para ello, se debe hacer click en el botón de seleccionar ficheros (*Botón para seleccionar nueva muestra*), y escoger un fichero de muestras. Es importante recordar que sólo pueden escogerse ficheros .DAT o .XML y que además tengan la estructura correcta explicada anteriormente. De lo contrario se mostrará un mensaje de error al usuario.

Una vez cargado el fichero, se actualizará automáticamente la lista de muestras y se mostrará además las opciones de la red a entrenar. Como se puede ver en la siguiente figura, se ha cargado una muestra con el nombre ESPECTRO del fichero ESPECTRO.DAT:

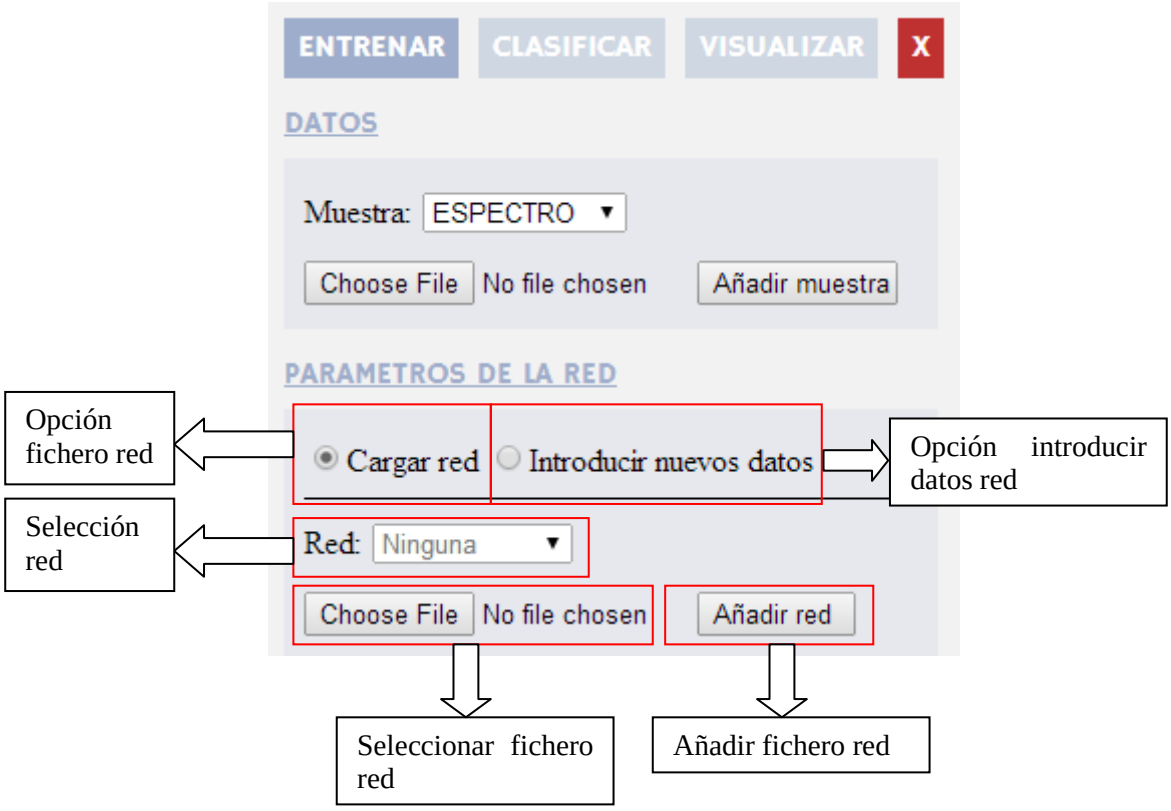

Figura 52: Interfaz Entrenar (2)

La opción de cargar red permite introducir los parámetros de una red desde un fichero mientras que la opción de introducir nuevos datos de red permite al usuario introducir dichos datos manualmente. Independientemente de la opción seleccionada, una vez cargado un fichero de red con la estructura correcta, o al introducir nuevos datos de red, se mostrarán los parámetros de la red (siendo en el caso de nuevos datos unos por defecto).

Es importante mencionar varios aspectos respecto a los parámetros. En primer lugar, en caso de que la red seleccionada esté entrenada, se le dará la opción al usuario de mantener los pesos existentes como pesos iniciales de entrenamiento (siempre y cuando el lado de Kohonen de la red y el número de datos de entrada seleccionados sean los correspondientes a dichos pesos). Por otro lado, en el caso de no seleccionar refuerzos, no serán tomados en cuenta los valores del fichero, y directamente no se podrán introducir dichos valores manualmente. Por último, mencionar que se ha controlado el caso de que los valores introducidos sean erróneos mediante mensajes de error.

A continuación se muestran los parámetros seleccionables de la red por el usuario una vez seleccionado una red o la opción de introducir nuevos datos:

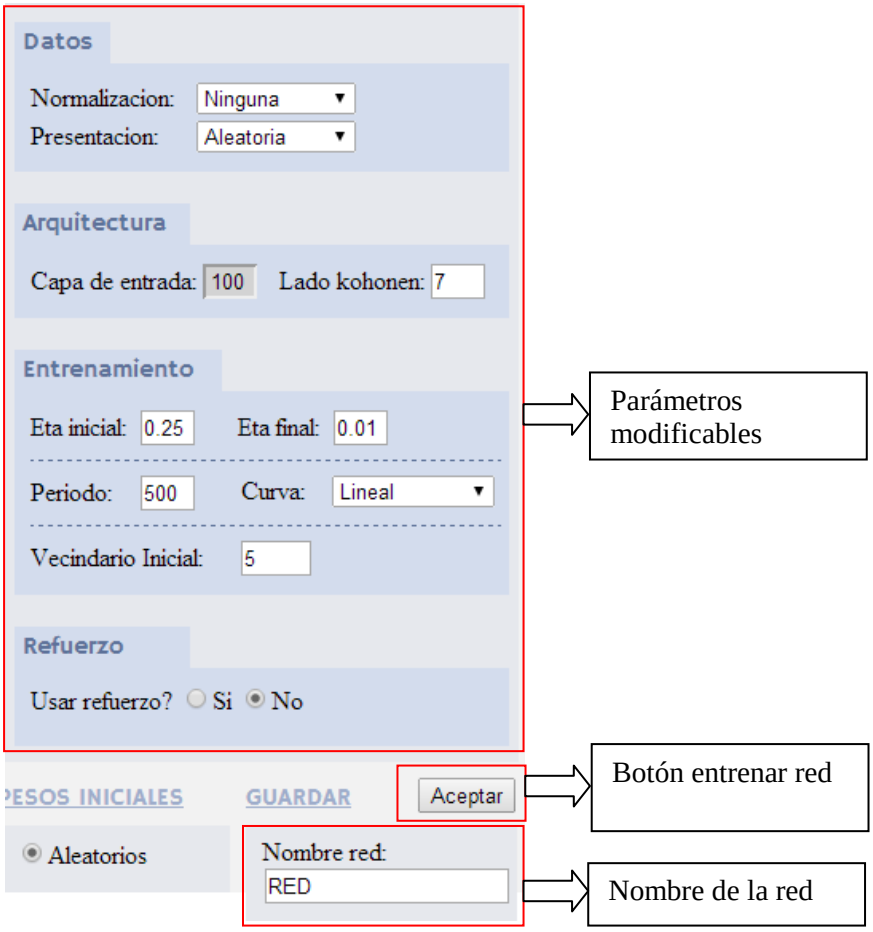

Figura 53: Interfaz Entrenar (3)

Se puede observar que el valor de capa de entrada no es un valor modificable, puesto que se calcula automáticamente en función de la muestra seleccionada. Por otro lado, seleccionar *Si* en la opción de usar refuerzo, el resultado es el siguiente:

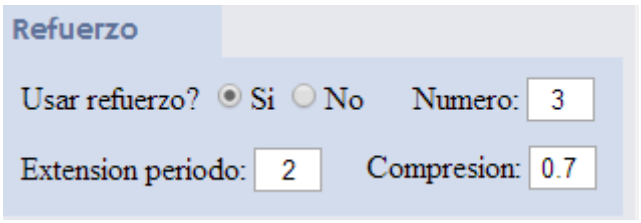

Figura 54: Interfaz Entrenar (4): Selección refuerzo

Al hacer click en el botón de entrenar red (*Aceptar*) se ejecutan todos los algoritmos correspondientes ya explicados y se generan los ficheros de salida en la carpeta de sesión del usuario. Tras esto se actualiza la interfaz gráfica y se muestran los resultados:

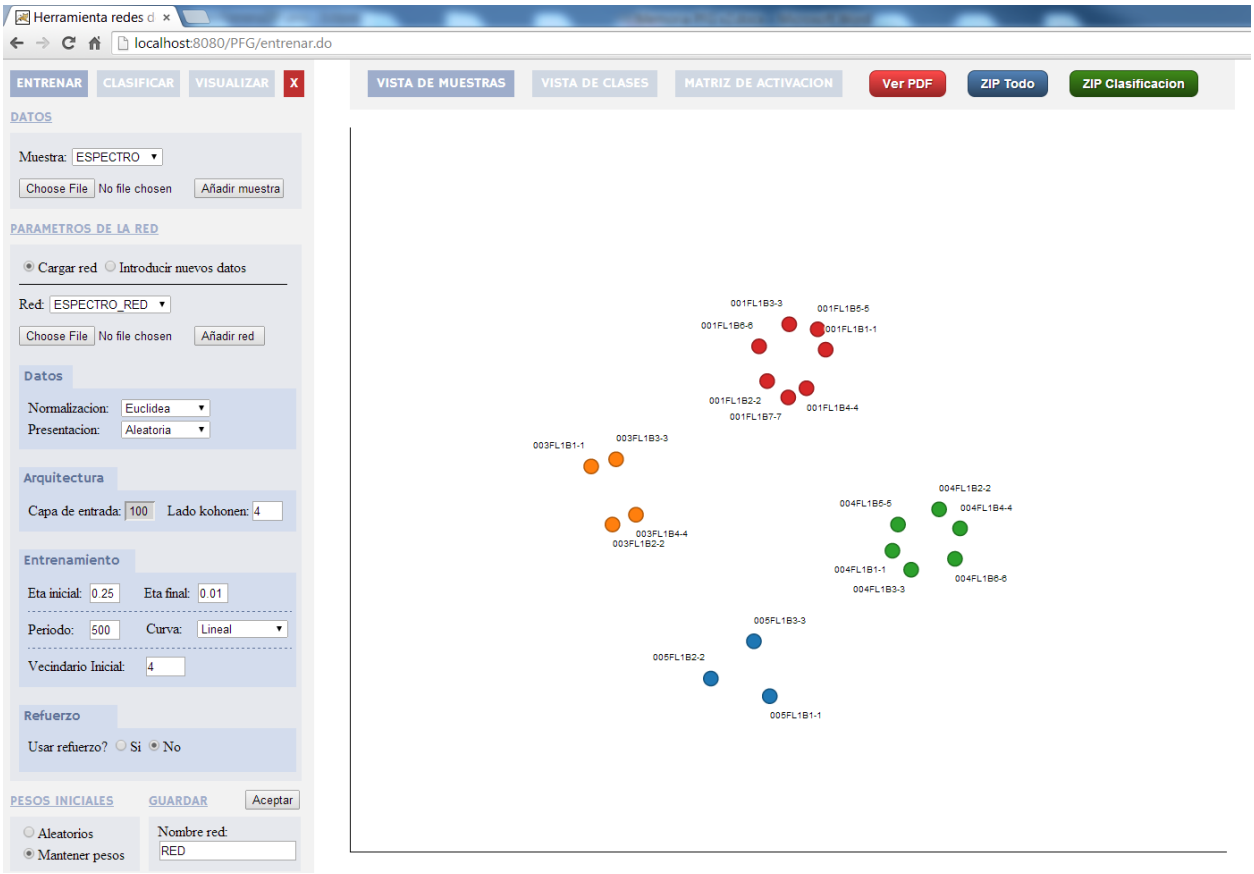

Figura 55: Resultados de entrenamiento y clasificación - Vista de muestras

Al hacer click en el botón de entrenar red, no sólo se entrena la red sino que automáticamente se clasifican las muestras utilizadas para el entrenamiento y se muestran los resultados de dicha clasificación. Como se puede observar, la interfaz gráfica ha sido actualizada con una barra superior con varias opciones:

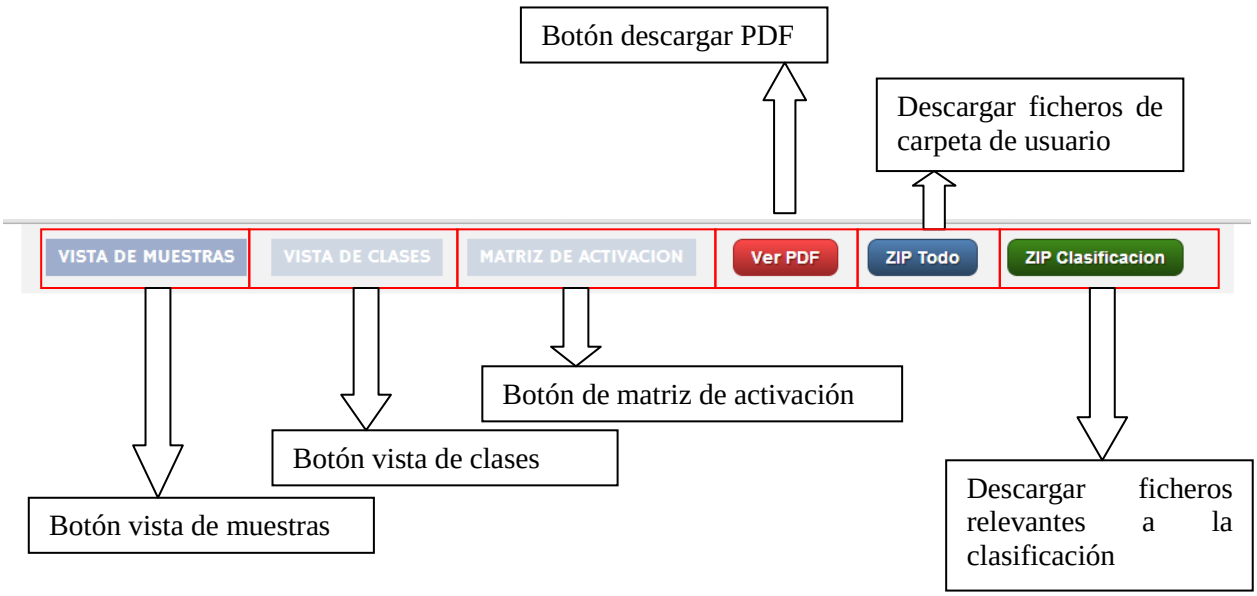

Figura 56: Interfaz de resultados

A su vez, se puede observar a continuación los resultados de las 3 vistas en más detalle:

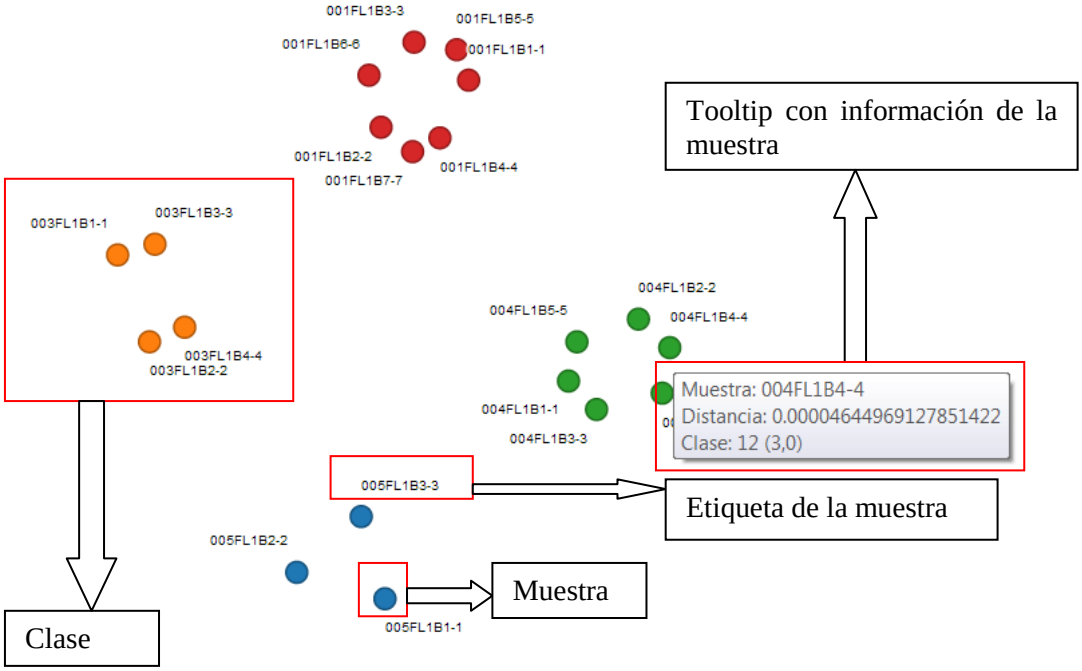

Figura 57: Vista de muestras

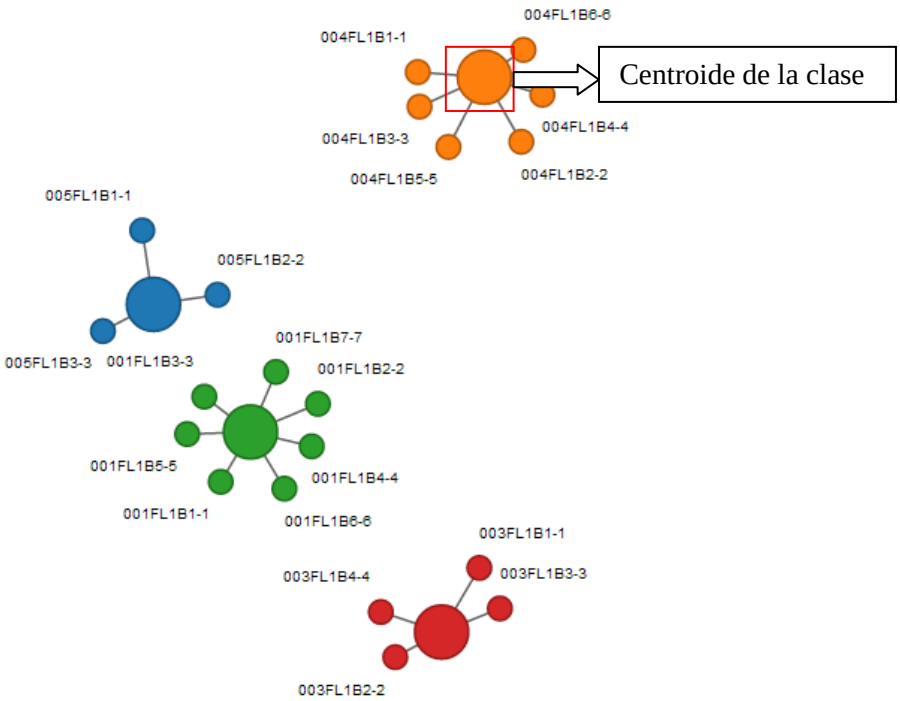

Figura 58: Vista de clases

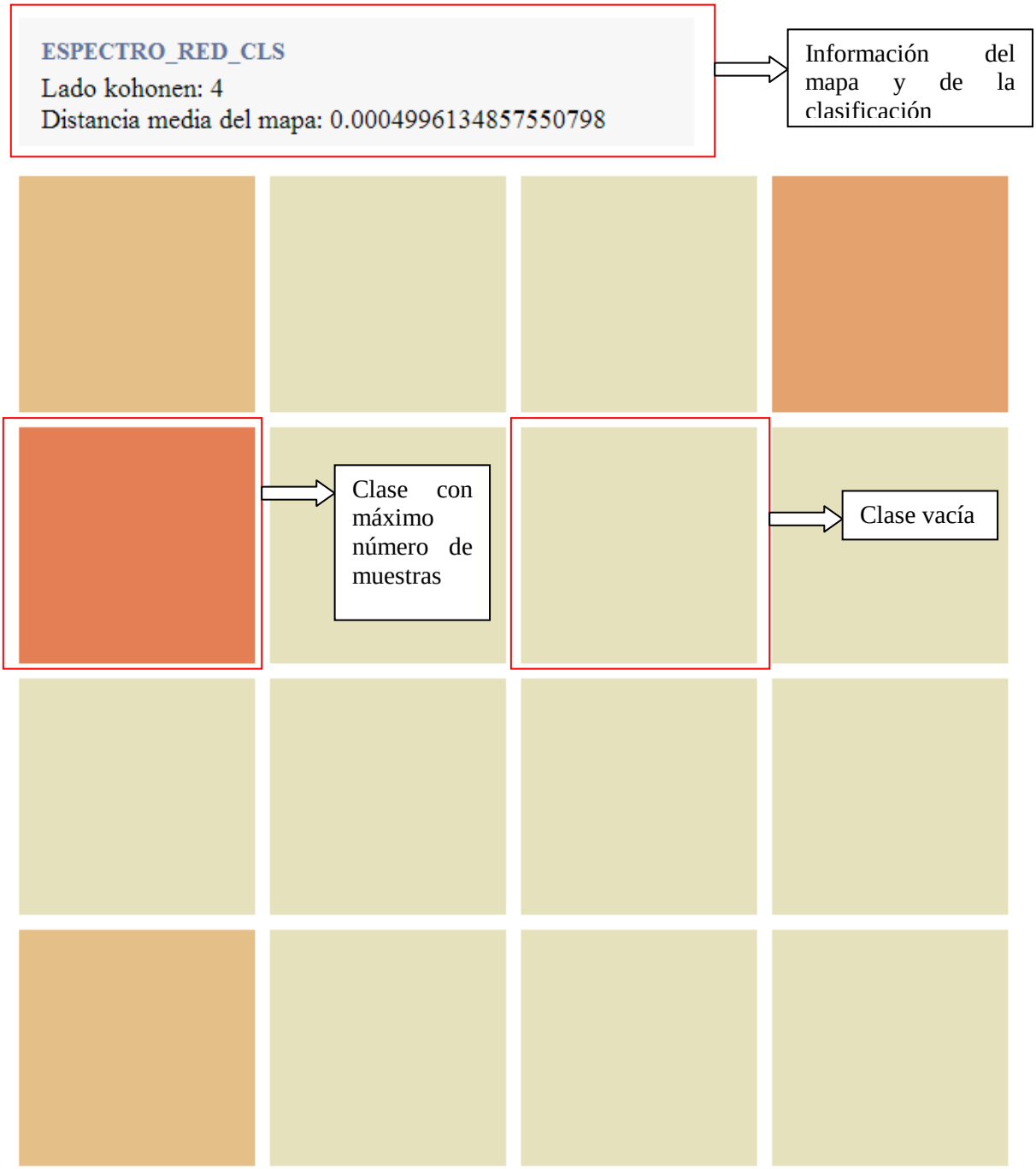

Figura 59: Matriz de Activación

Las últimas 3 figuras muestran los tipos de visualización de resultados ofrecidos por la herramienta. Cada uno tiene sus ventajas y desventajas, pero todos permiten al usuario comprender rápidamente la clasificación obtenida.

Por último, destacar que al hacer click en los botones de *Ver PDF*, *ZIP Todo y ZIP Clasificación*, se descargan automáticamente los ficheros correspondientes a la selección.

Tras ver los resultados y descargarlos, para salir de la herramienta basta con darle al botón rojo *X* que permite cerrar la sesión del usuario y destruir su carpeta.

CLASIFICAR MUESTRAS

Al hacer click en el botón *Clasificar*, se carga la pestaña de clasificación con sus parámetros correspondientes. Es importante mencionar que a continuación sólo se explica la interfaz y no los resultados obtenidos de visualización, ya que son exactamente iguales a los del entrenamiento.

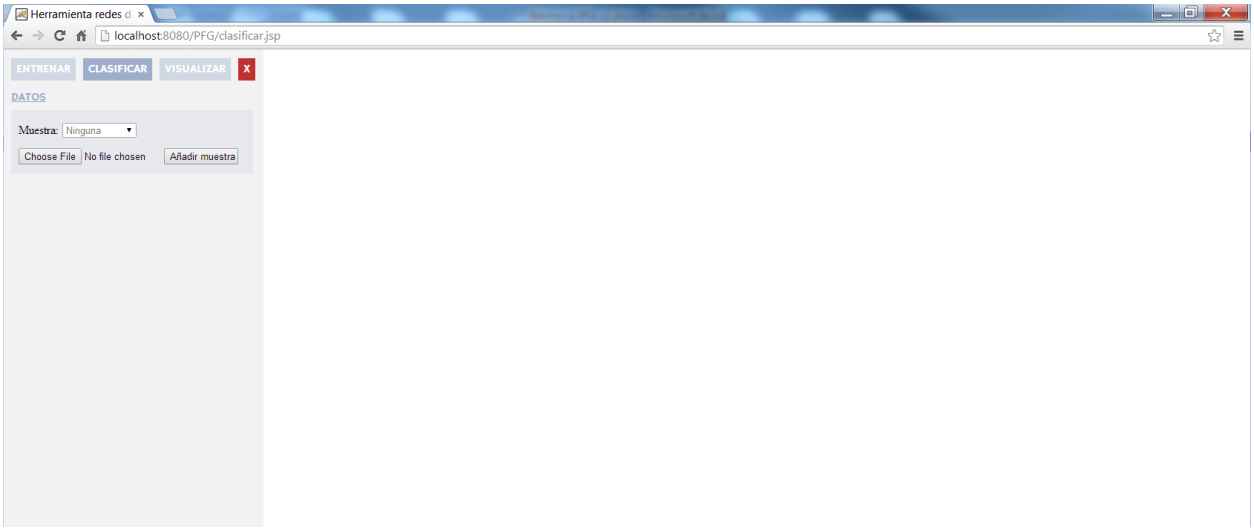

Figura 60: Interfaz Clasificar

La siguiente figura explica la barra izquierda de la herramienta, que contiene las opciones disponibles:

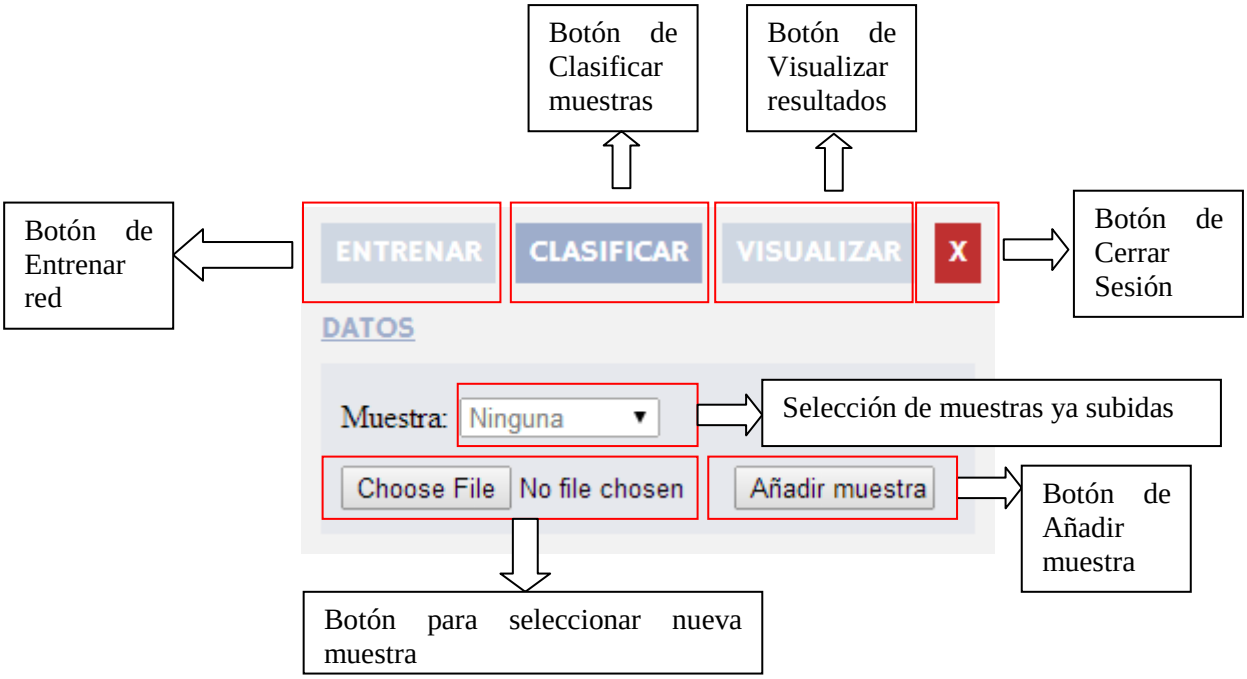

Figura 61: Interfaz Clasificar (1)

Es importante mencionar que las muestras son compartidas entre las pestañas de *Entrenar* y *Clasificar*, por lo que una muestra subida en una será visible en la otra.

Una vez cargada alguna muestra, se pasa a la selección de la red. En este caso, sólo es posible subir una red ya entrenada desde un fichero. En caso de que no esté entrenada, un mensaje de error se lo indicará al usuario:

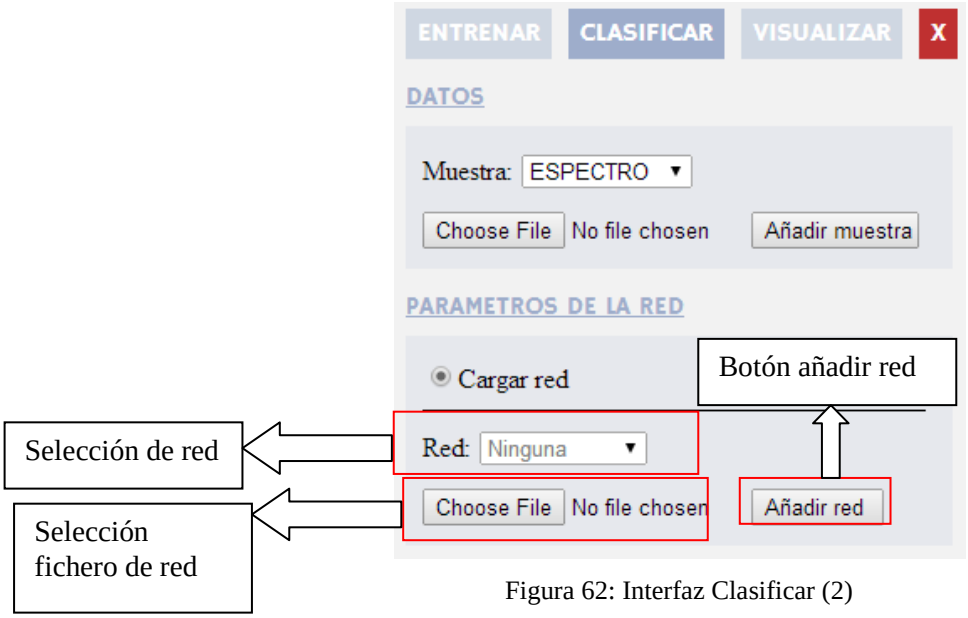

Figura 62: Interfaz Clasificar (2)

Figura 63: Interfaz Clasificar (3)

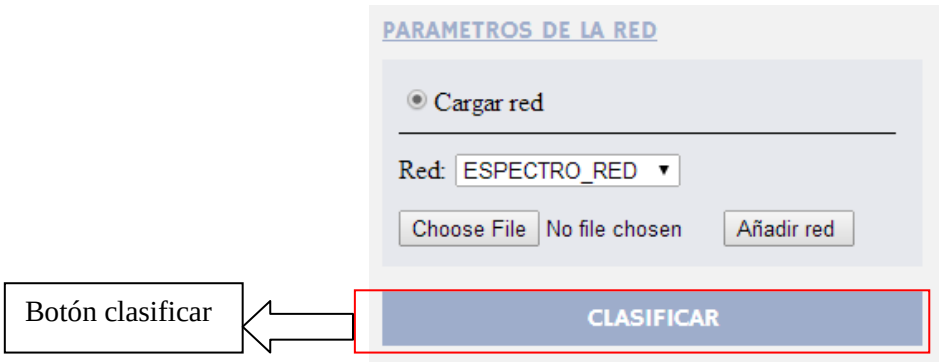

Figura 64: Interfaz Clasificar (4)

Tras seleccionar la muestra deseada y la red entrenada con la que se desea clasificar, se mostrarán los resultados ya explicados en el apartado de *Entrenar*. En este caso, sin embargo, se pasa directamente a la clasificación sin haber habido un entrenamiento.

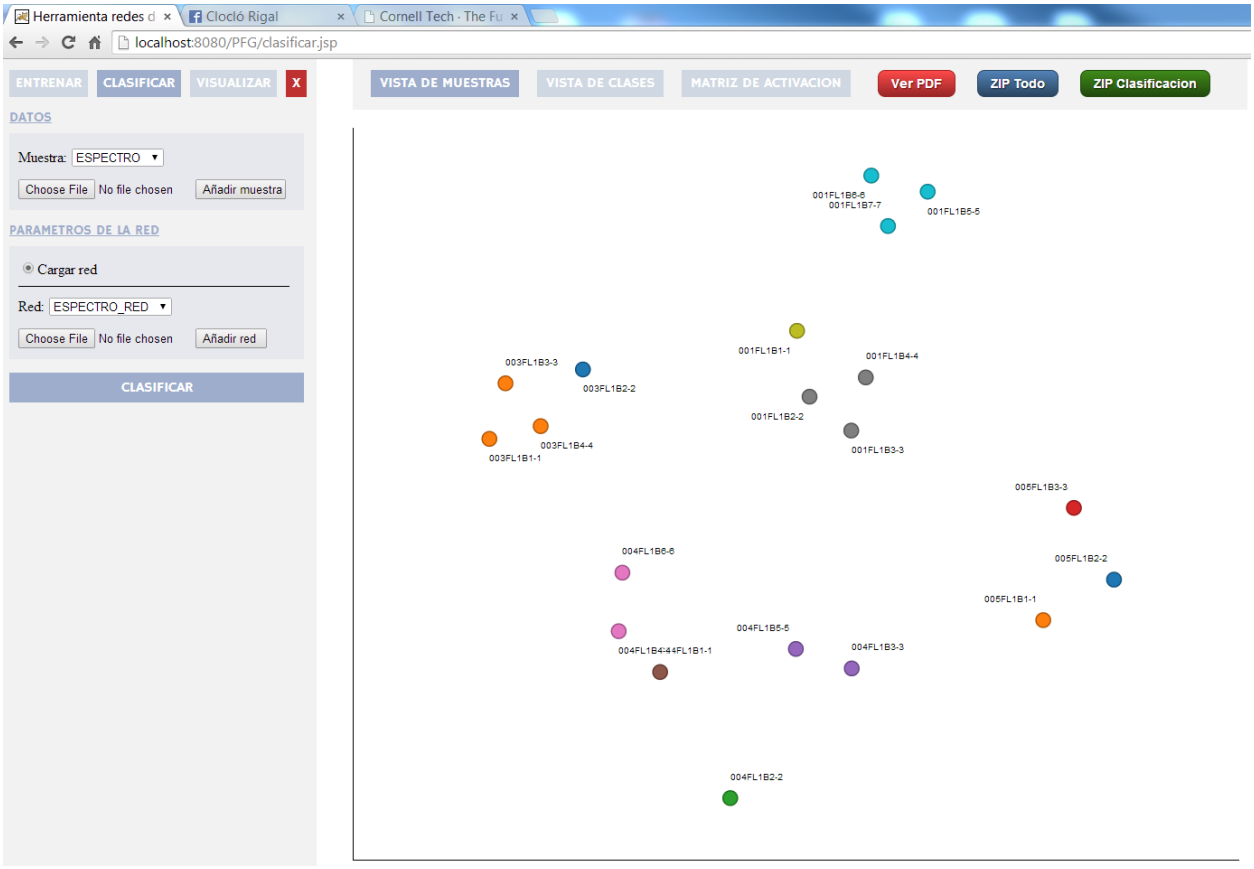

Figura 65: Interfaz Clasificar (5)

VISUALIZAR CLASIFICACIÓN

Al hacer click en el botón *Visualizar*, se activará la interfaz gráfica de visualización de resultados:

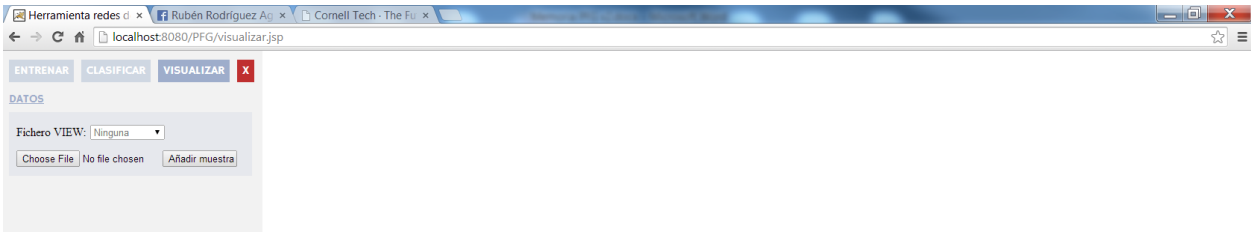

Figura 66: Interfaz Visualizar (1)

La siguiente figura explica la barra izquierda de la herramienta, que contiene las opciones disponibles:

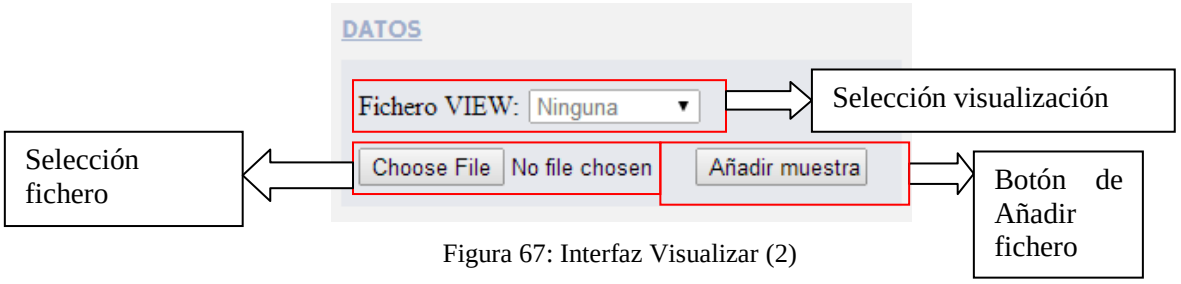

Figura 67: Interfaz Visualizar (2)

Tras subir un fichero de visualización y darle al botón *Visualizar*, se cargarán los resultados de la visualización. Cabe destacar que en este caso no se permite descargar los ficheros ZIP o PDF:

Figura 68: Interfaz Visualizar (3)

7. CONCLUSIONES Y PROPUESTAS

El proyecto presentado ha cumplido con todas las expectativas y los objetivos propuestos, hecho demostrado mediante la adecuación de la herramienta a los requisitos de usuario.

1. Se ha construido una aplicación web para representar un modelo de aprendizaje no supervisado mediante redes neuronales de Kohonen,
    a. La aplicación está basada en el patrón de diseño MVC (Modelo-Vista-Controlador) y es del tipo *Java Web Application*. Se ha utilizado el framework Struts para la comunicación entre la interfaz y las clases de Java pertenecientes a la Herramienta de Kohonen y generación de ficheros. Este framework es lo que permite utilizar el MVC, ya que permite el desarrollo de una vista (archivos web JSP), que llaman a una serie de acciones (clases Java), las cuales responden a un formulario que representa el modelo. Esto último se puede entender mejor mediante un ejemplo: un formulario en la página JSP contendrá una serie de campos a rellenar (en el caso de este proyecto dichos formularios eran para subir ficheros o para indicar los parámetros de la red). El modelo en Struts es representada por una clase que contiene un atributo fichero en el caso del primer tipo de formulario y un atributo por cada parámetro de la red en el caso del segundo. Esta clase era la encargada de guardar toda la información introducida por el usuario y es la clase activada por la acción para recoger dicha información. En ese sentido, si bien se puede afirmar que se ha seguido un patrón de MVC, no es un modelo de 3 capas.
    b. Esta herramienta puede ser utilizada con cualquier conjunto de datos, recogidos en ficheros con la estructura correspondiente explicada en esta memoria. De hecho, se ha desarrollado todo un módulo de subida y almacenamiento de archivos para cada usuario distinto que utilice la herramienta. Dichos ficheros permiten indicar desde los datos de entrada, hasta los parámetros y pesos iniciales de una red, e incluso la información necesaria para cargar una visualización ya vista anteriormente.
2. Por otro lado, en lo que a la interfaz gráfica se refiere, se ha desarrollado siempre teniendo en cuenta la experiencia final del usuario,
    a. Dando lugar a una GUI amigable y accesible sin complicaciones. Tanto es así, que se ha ordenado de la mejor manera posible todos los posibles parámetros de una red de Kohonen, cargando valores por defecto que suelen ser adecuados, y mostrando mensajes de error tanto en el establecimiento de parámetros imposibles como en la subida de archivos erróneos, y también para aquellos casos en los que la sesión del usuario hubiera caducado.
    b. A su vez, se ha realizado una herramienta de visualización de resultados que permite al usuario comprender la clasificación de forma sencilla. Esto último se ha conseguido mediante la librería Javascript D3.JS, la cual ofrece una de las mejores visualizaciones de datos disponibles para web de forma completamente gratuita. Si bien apenas se ha utilizado dos de las muchas herramientas de D3.JS para representar los resultados gráficamente (una de ellas para la visualización de muestras y clases y otra para mostrar el mapa de activación de la red), se han escogido las que más útiles resultaban para el proyecto en cuestión.

3. Se ha trabajado con librerías de Java para la generación de archivos PDF que generen un informe de cada clasificación, con librerías para la generación de ficheros ZIP los cuales permiten descargar toda la información y los archivos de la carpeta de sesión,
    a. En el caso de los ficheros PDF, se ha utilizado la librería externa *itext* para generarlos. Esta librería tiene un gran potencial ya que permite establecer todo tipo de parámetros en fichero: color y tamaño de fuente, ubicación del texto, generación de tablas, etc. En este proyecto se han utilizado las funciones más básicas de la librería.
    b. Para los ficheros ZIP, se ha utilizado la librería por defecto de Java habilitada para ello. En este proyecto dichos ficheros adquieren una gran importancia, puesto que permiten que en todo momento se guarde la versión más actualizada de todo el contenido de la carpeta de sesión del usuario y que mediante una sola descarga pueda acceder a dicha información.
4. Se ha llevado a cabo una aplicación práctica mediante la clasificación de datos facilitados por el tutor.
    a. Dichos corresponden a un caso real en la clasificación de espectros. Por ello se ha considerado que era una prueba adecuada para garantizar la correcta clasificación de la herramienta construida. Además, ya se conocían los resultados óptimos de esta clasificación, a los cuales se ha podido llegar en esta herramienta utilizando los parámetros correctos.
    b. Esto demuestra que los algoritmos fueron implementados correctamente, dando lugar a un herramienta web que no sólo ofrece una visualización atractiva de los resultados en una interfaz gráfica amigable, sino que además garantiza una ejecución correcta y una exactitud en los cálculos gracias en gran medida al backend (Java) utilizado.

De esta manera, se ha podido cumplir con el principal objetivo del proyecto, el cual hacía referencia a generar un predicador y clasificador de datos mediante una red de Kohonen que mantuviera la exactitud de los resultados necesarios para una clasificación correcta (sobre todo en lo referente a los pesos de la red y al cálculo de la neurona ganadora) pero que a su vez pudiera aprovechar del gran potencial de visualización de datos vía web que lleva cobrando fuerza en los últimos años.

Cabe destacar que el uso de D3.JS no ha sido particularmente fácil, y los resultados obtenidos han sido fruto de una gran cantidad de horas dedicadas a los detalles de la visualización. Entre otros aspectos, se ha experimentado con el color de los nodos, su tamaño para diferenciar muestras y centroides, las líneas que unen los nodos (se había considerado la posibilidad de que fuesen flechas), las distancias existentes entre los nodos (y la adecuación de dichas distancias al tamaño del lado de Kohonen de la red), la posición de las etiquetas de las muestras para que no se sobrepusieran unas sobre otras, los eventos de generación de *tooltips* al poner el ratón encima de los nodos, la carga de los archivos desde ficheros JSON, las relaciones existentes entre nodos de muestra y nodos de clase, y muchas otras funcionalidades que se han quedado fuera porque se concluyó que al final no eran convenientes o por la complejidad de su implementación.

En lo que a la arquitectura cliente-servidor utilizada se refiere, la decisión de dicha implementación tuvo lugar por varios motivos:

- La ya explicada necesidad de combinar la fuerza de cálculo de Java con la representación de datos vía web, para lo cual es necesario un servidor que realice los cálculos.

- La facilidad de acceso a la herramienta mediante una sola instalación. Basta con pegar el archivo WAR resultante del proyecto en la carpeta de Apache Tomcat y acceder a la dirección IP del servidor desde los terminales conectados a una red de área local para poder hacer uso de la herramienta.
- La separación de los cálculos entre el servidor y el cliente. Algunos detalles de la aplicación son ejecutados mediante Javascript, sin tener que realizar llamadas al servidor. Esto reduce la carga en el mismo para operaciones menores como puede ser el mostrar mensajes de error, ahorrando recursos importantes.

En lo que a propuestas de mejora se refiere, aún hay muchas formas de mejorar la herramienta desarrollada:

- *Mejoras sobre el algoritmo de Kohonen:* la herramienta actual no tiene en consideración otros tipos de normalización como puede ser el ZAxis. Por otro lado, no permite el uso de consciencia en el entrenamiento (más información sobre ambos se encuentra disponible en el punto referente al estado del arte del presente documento).
- *Mejoras en la visualización:* si bien la visualización ofrecida es bastante clara en los resultados, aún es posible incorporar ciertas mejoras entre las que destacan:
    - Utilizar una gama de colores distinta para los resultados. Actualmente la herramienta genera los colores de las clases de forma aleatoria, pero se podría mejorar si se utilizara un rango de colores mediante el cual mientras más cercanas se encuentren las clases y las muestras, más parecido será el color de su nodo.
    - Mejorar la representación de distancias entre los nodos de muestras y nodos de clases. Actualmente la distancia entre muestras y clases no es representada de forma exacta debido a las grandes diferencias de distancia que pueden existir entre 2 muestras que pertenecen a la misma clase. Esto da lugar a que la visualización sea confusa, por lo que se ha optado por que la distancia entre las muestras a las clases sea similar.
- *Mejoras en los ficheros de clasificación:* sería interesante incorporar la posibilidad de que el fichero de clasificación no sólo indique las etiquetas de las muestras que pertenecen a una clase, sino también los valores de sus atributos (su vector de valores).
- *Ampliación a otros modelos de aprendizaje no supervisado:* sería interesante implementar otros algoritmos de aprendizaje no supervisado y poder realizar comparaciones entre los resultados obtenidos.
- *Ampliación a modelos de aprendizaje supervisado:* tras el correspondiente rediseño de la herramienta, una ampliación a modelos de aprendizaje supervisado incluyendo mecanismos de validación cruzada para asegurar el correcto aprendizaje de la red, daría lugar a una herramienta potente y completa de clasificación de datos. Además si a todos los algoritmos implementados se les desarrolla una visualización como la utilizada actualmente (y mejorada), el resultado podría tener un potencial muy interesante para la comparativa de algoritmos.

8. LISTA DE REFERENCIAS, IMAGENES Y BIBLIOGRAFÍA

1. **TAN, STEINBACH, KUMAR.** *Introduction to Data Mining.* Londres : Pearson International, 2014.

2. **ROJAS, R.** *Neural Networks.* Berlin : s.n., 1994. pág. 408.

3. **SAVVA, KONG, Chhajta, FEI-FEI, AGRWALA, HEER.** *ReVision: Automated Classification, Analysis and Redesign of Chart Images.* s.l. : ACM User Interface Software and Technology (UIST), 2011.

4. **ABRIL, J.** *Modelos para el análisis de las series de tiempo.* s.l. : Consejo Nacional de Investigaciones Científicas y Técnicas de Argentina, 2006.

5. **MALLO, C.** *Predicción de la demanda eléctrica horaria mediante redes neuronales.* Oviedo : Departamento de Economía Cuantitativa, Universidad de Oviedo, 2008.

6. **JONES, M. TIM.** *Artificial Intelligence: A Systems Approach.* s.l. : Jones and Barlett, 2009.

7. **BISHOP, C.M.** *Neural Networks for Pattern Recognition.* Oxford : Oxford University Press, 1995.

8. **S. GEMAN, E. BEIENSTOCK, R. DOURSTAT.** *Neural Networks and the Bias/Variance Dilemma.* s.l. : Neural Computation, 1998.

9. **BLUM, A.** *Neural Networks in C++.* New York : Wiley, 1992.

10. **SWINGLER, K.** *Applying Neural Networks: A Practical Guide.* Londres : s.n., 1996.

11. **M.J.A BERRY, G. LINOFF.** *Data Mining Techniques.* New York : s.n., 1997.

12. **Z. BOGER, H. GUTERMAN.** *Knowledge extraction from artificial neural network models.* Orlando, FL : IEEE Systems, Man and Cybernetics Conference, 1997.

13. **KOHONEN, T.** *Self-organized formation of topogically correct feature maps.* s.l. : Biological Cybernetics, 1982.

14. **MALSBURG, C. VON DER.** *Self-organization of orientation sensitive cells in the striate cortex.* s.l. : Kyternetik, 1973.

15. **KOHONEN, T.** *The neural phonetic typewriter.* s.l. : IEEE Computer, 1988.

16. **CAUDILL, M.** *Kohonen Learning.* s.l. : Al Expert, 1990.

17. **HIOTIS, A.** *Inside a self-organizing map.* s.l. : Al Expert, 1993.

18. **R.C ELBERHART, R.W DOBBINS.** *Neural Networks PC Tools: A Practical Guide.* s.l. : Academic Press, 1990.

9. ANEXOS

9.1 ANEXO I: INSTRUCCIONES DE INSTALACIÓN

La herramienta ha sido diseñada para poder acceder a ella desde cual ordenador conectado a una red de área local. Por ello, basta con instalarla en uno de los ordenadores que actuará de servidor. Los pasos para su instalación son los siguientes:

1. Asegurar que se tiene instalado en el servidor la aplicación *Apache Tomcat 7.0*. De no ser así, descargarla en *http://tomcat.apache.org/download-70.cgi*
    - También se dispone del ejecutable para instalar *Apache Tomcat 7.0* en el CD que acompaña a la memoria. En la carpeta *Instalables*
2. Tras haber instalado *Apache Tomcat* en el servidor, insertar el CD que compaña a esta memoria y buscar la carpeta *Instalables*. Una vez ahí copiar el fichero *PFG.WAR*
3. Ubicar la carpeta de instalación de *Apache Tomcat*, por defecto es *C:\Program Files\Apache Software Foundation\Tomcat 7.0*. Una vez ahí, ubicar la carpeta *webapps* y entrar en ella.
4. Pegar el fichero *PFG.WAR* en la carpeta *webapps* correspondiente a la instalación de *Apache Tomcat 7.0*.
5. Abrir el configurador de *Apache Tomcat*, el cual se encuentra ubicado por defecto en *C:\Program Files\Apache Software Foundation\Tomcat 7.0\bin*, su nombre es *Tomcatw7.exe*

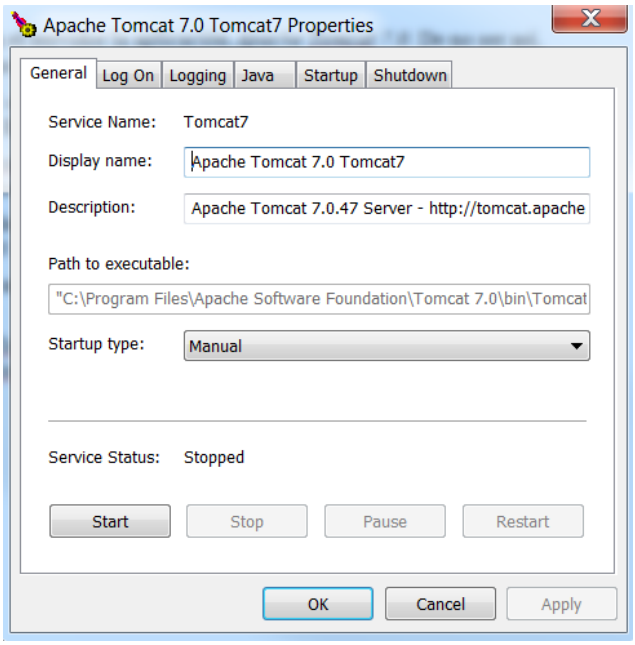

Figura 69: Configurador de Apache Tomcat

6. Hacer click en el botón *Start* para ejecutar el servidor y arrancar la herramienta.
7. Una vez ejecutado el servidor, automáticamente se creará una carpeta llamada *PFG* en la carpeta *webapps* ubicada anteriormente. Es recomendable comprobar que se ha creado sin ningún problema.

8. Comprobar que la herramienta corre correctamente en el servidor. Para ello, basta con abrir el explorador de internet (recomendado *Google Chrome*), e introducir la ruta: *http://localhost:8080/PFG/index.jsp*

   Si aparece la siguiente pantalla, entonces la aplicación se ha instalado en el servidor correctamente:

Figura 70: Página inicial de la herramienta

9. Para poder acceder a la herramienta desde el resto de ordenadores, es necesario conocer la dirección IP del servidor. Para ello hay que ir a inicio, ejecutar *cmd*, y escribir *ipconfig*, darle a *Enter*, y buscar la dirección IP. En la siguiente imagen sería 192.168.1.2:

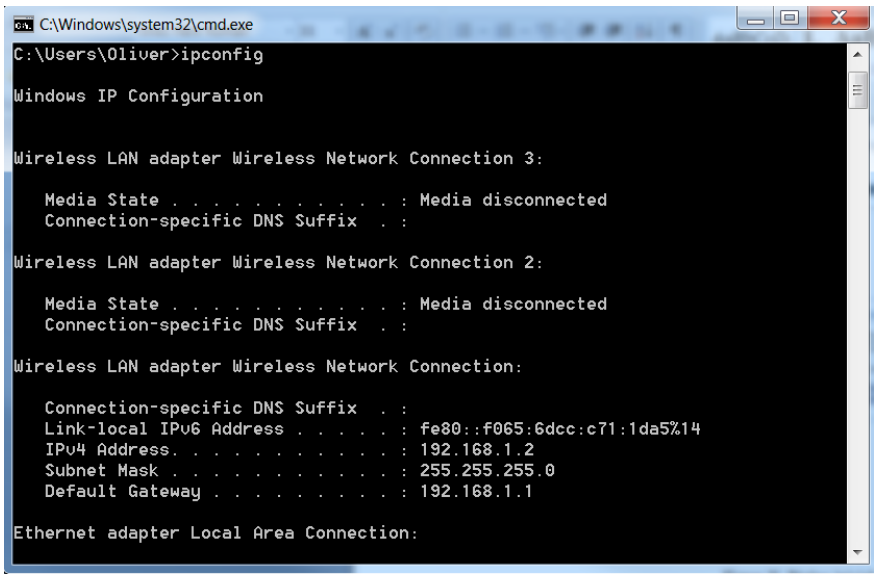

Figura 71: Dirección IP del servidor

10. Copiar/Recordar la dirección IP, y en cualquier otro ordenador de la red abrir el explorador de internet (recomendado *Google Chrome*), y escribir:

http://192.168.1.2:8080/PFG/index.jsp, sustituyendo 192.168.1.2 por la dirección IP del servidor en cuestión. Tras esto, se debería ejecutar la herramienta:

Figura 72: Página de inicio en el cliente

9.2 ANEXO II: INSTRUCCIONES DE USO

Antes de comenzar a utilizar la herramienta, es necesario disponer de un fichero de datos con la estructura correcta. En el CD de instalación en la carpeta Instalables, se dispone de un fichero de datos ejemplo llamado ESPECTRO.DAT y ESPECTRO.XML. Ambos pueden ser utilizados como ficheros de datos. Cualquier fichero que tenga la misma estructura también podrá ser utilizado como fichero de dato:

- La estructura de los ficheros con extensión .DAT es:

  $Etiqueta_1$ $dato_{1,1}$ $dato_{1,2}$ ..... $dato_{1,N}$

  $Etiqueta_2$ $dato_{2,1}$ $dato_{2,2}$ ..... $dato_{1,N}$

  ....

  $Etiqueta_M$ $dato_{M,1}$ $dato_{M.2}$ ..... $dato_{M,N}$

  ---

  donde:

  - M: es el número de muestras
  - N: es el tamaño de las muestras
  - <Etiqueta$_M$> es la cadena que identifica a la *m-ésima* muestra
  - <dato$_{MN}$> es el *n-ésimo* componente del *m-ésimo* vector muestra

  Los datos pueden ser números enteros o reales. En este último caso, es obligatorio que el separador de decimales sea el punto (**.**), no la coma (**,**). En ambos casos no debe de existir separador de miles.

- La estructura de los ficheros de muestra con extensión .XML es la siguiente:

```
<?xml        version="1.0"        encoding="UTF-8"
standalone="no"?>
<muestras>
    <dimension>DIMENSION_MUESTRAS</dimension>
    <muestra>
          <etiqueta>ETIQUETA_MUESTRA</etiqueta>
        <dato>0.548952</dato>
        <dato>0.52904</dato>
        <dato>0.519982</dato>
          .
        .
      </muestra>
      <muestra>
      .
      </muestra>
</muestras>
```

Una vez ubicado los ficheros de muestra, es posible utilizar la herramienta: se puede acceder a ella desde cualquier explorador web mediante la dirección IP del servidor en donde ha sido instalada. En este ejemplo, se accede desde el mismo servidor, por lo que bastaría con ir a: *http://localhost:8080/PFG/index.jsp*

Se llega a la ventana de bienvenida de la herramienta:

Figura 73: Ventana de bienvenida de la herramienta

En esta ventana se le explica al usuario el funcionamiento de la herramienta y se crea la sesión del mismo. Una vez leídas las instrucciones se debe hacer click en el botón *Entrar*, el cual llevará al usuario a la página principal de la herramienta. Se puede observar que se puede seleccionar el método de trabajo que se desea, disponiendo de las opciones *Entrenar*, *Clasificar*, *y Visualizar.*

ENTRENAMIENTO DE UNA RED

A continuación se puede observar la interfaz gráfica para el entrenamiento de la red:

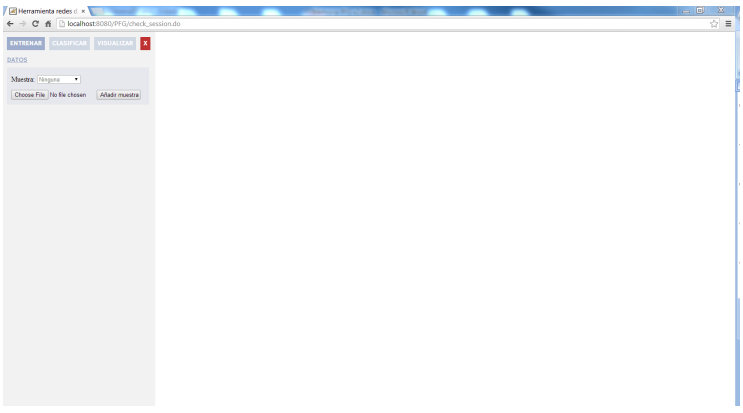

Figura 74: Interfaz gráfica inicial de la herramienta

La siguiente figura explica la barra izquierda de la herramienta, que contiene las opciones disponibles:

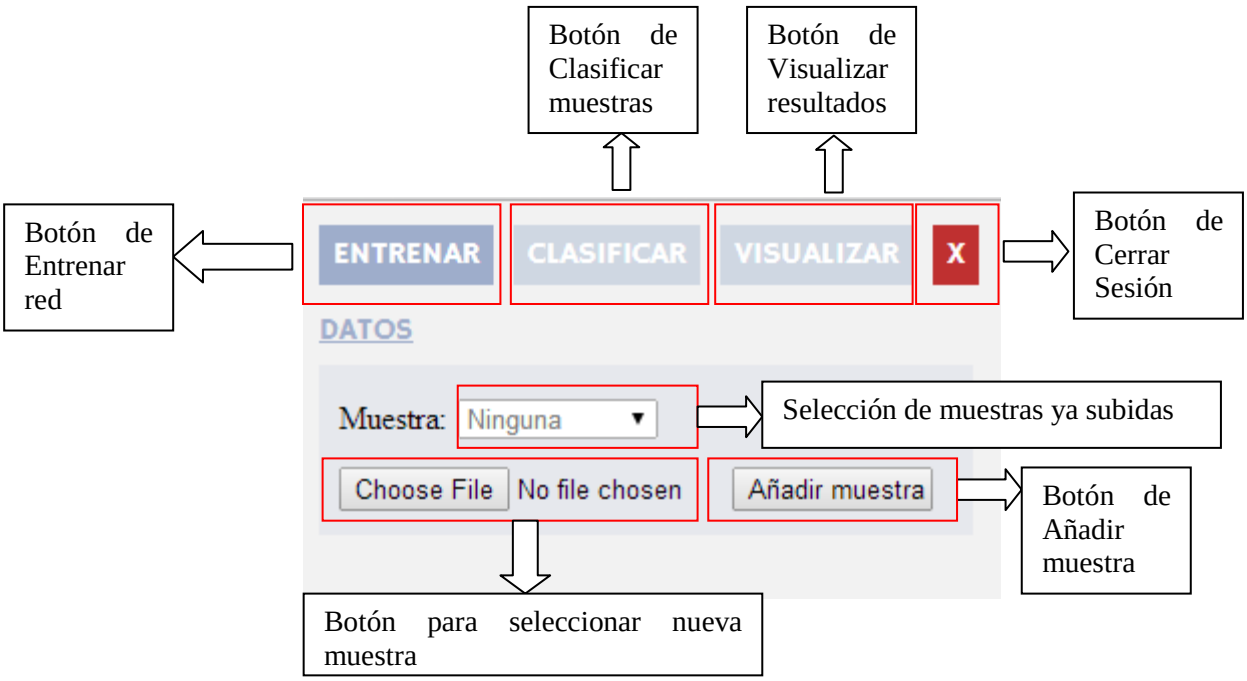

Figura 75: Interfaz Entrenar (1)

Lo primero que se debe hacer para entrenar una red, es seleccionar un fichero de muestras con el cual se desea entrenar la red. Para ello, se debe hacer click en el botón de seleccionar ficheros (*Botón para seleccionar nueva muestra*), y escoger un fichero de muestras. Es importante recordar que sólo pueden escogerse ficheros .DAT o .XML y que además tengan la estructura correcta explicada anteriormente. De lo contrario se mostrará un mensaje de error al usuario.

Una vez cargado el fichero, se actualizará automáticamente la lista de muestras y se mostrará además las opciones de la red a entrenar. Como se puede ver en la siguiente figura, se ha cargado una muestra con el nombre ESPECTRO del fichero ESPECTRO.DAT:

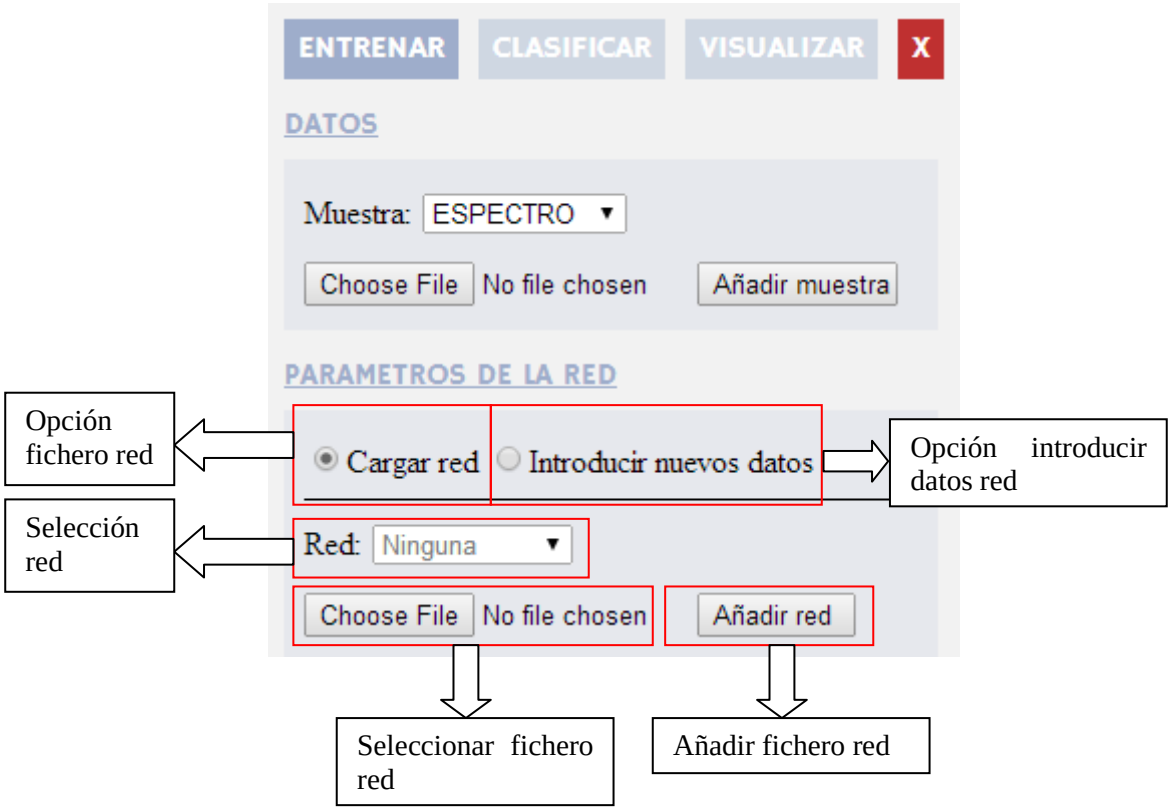

Figura 76: Interfaz Entrenar (2)

La opción de cargar red permite introducir los parámetros de una red desde un fichero mientras que la opción de introducir nuevos datos de red permite al usuario introducir dichos datos manualmente. Independientemente de la opción seleccionada, una vez cargado un fichero de red con la estructura correcta, o al introducir nuevos datos de red, se mostrarán los parámetros de la red (siendo en el caso de nuevos datos unos por defecto).

Es importante mencionar varios aspectos respecto a los parámetros. En primer lugar, en caso de que la red seleccionada esté entrenada, se le dará la opción al usuario de mantener los pesos existentes como pesos iniciales de entrenamiento (siempre y cuando el lado de Kohonen de la red y el número de datos de entrada seleccionados sean los correspondientes a dichos pesos). Por otro lado, en el caso de no seleccionar refuerzos, no serán tomados en cuenta los valores del fichero, y directamente no se podrán introducir dichos valores manualmente. Por último, mencionar que se ha controlado el caso de que los valores introducidos sean erróneos mediante mensajes de error.

A continuación se muestran los parámetros seleccionables de la red por el usuario una vez seleccionado una red o la opción de introducir nuevos datos:

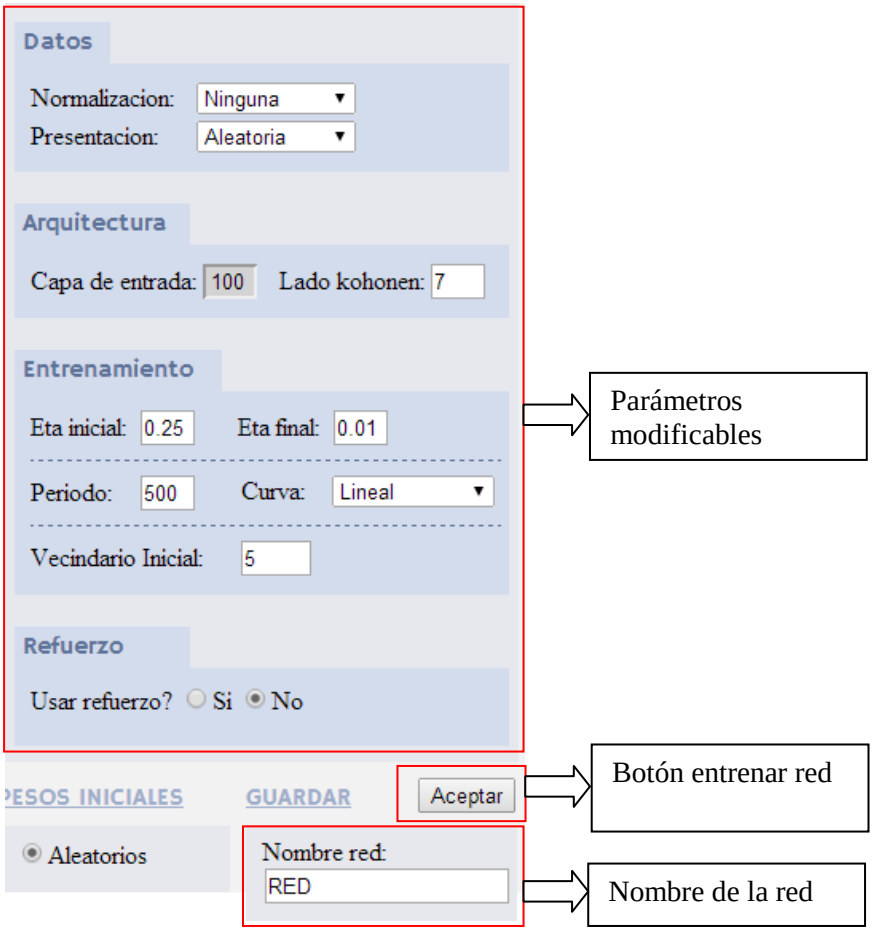

Figura 77: Interfaz Entrenar (3)

Se puede observar que el valor de capa de entrada no es un valor modificable, puesto que se calcula automáticamente en función de la muestra seleccionada. Por otro lado, seleccionar *Si* en la opción de usar refuerzo, el resultado es el siguiente:

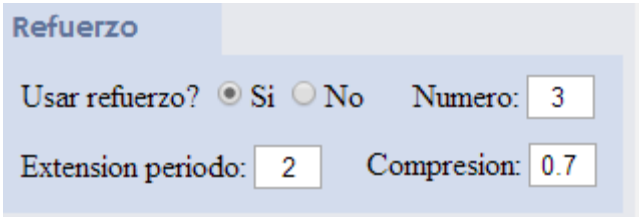

Figura 78: Interfaz Entrenar (4): Selección refuerzo

Al hacer click en el botón de entrenar red (*Aceptar*) se ejecutan todos los algoritmos correspondientes ya explicados y se generan los ficheros de salida en la carpeta de sesión del usuario. Tras esto se actualiza la interfaz gráfica y se muestran los resultados:

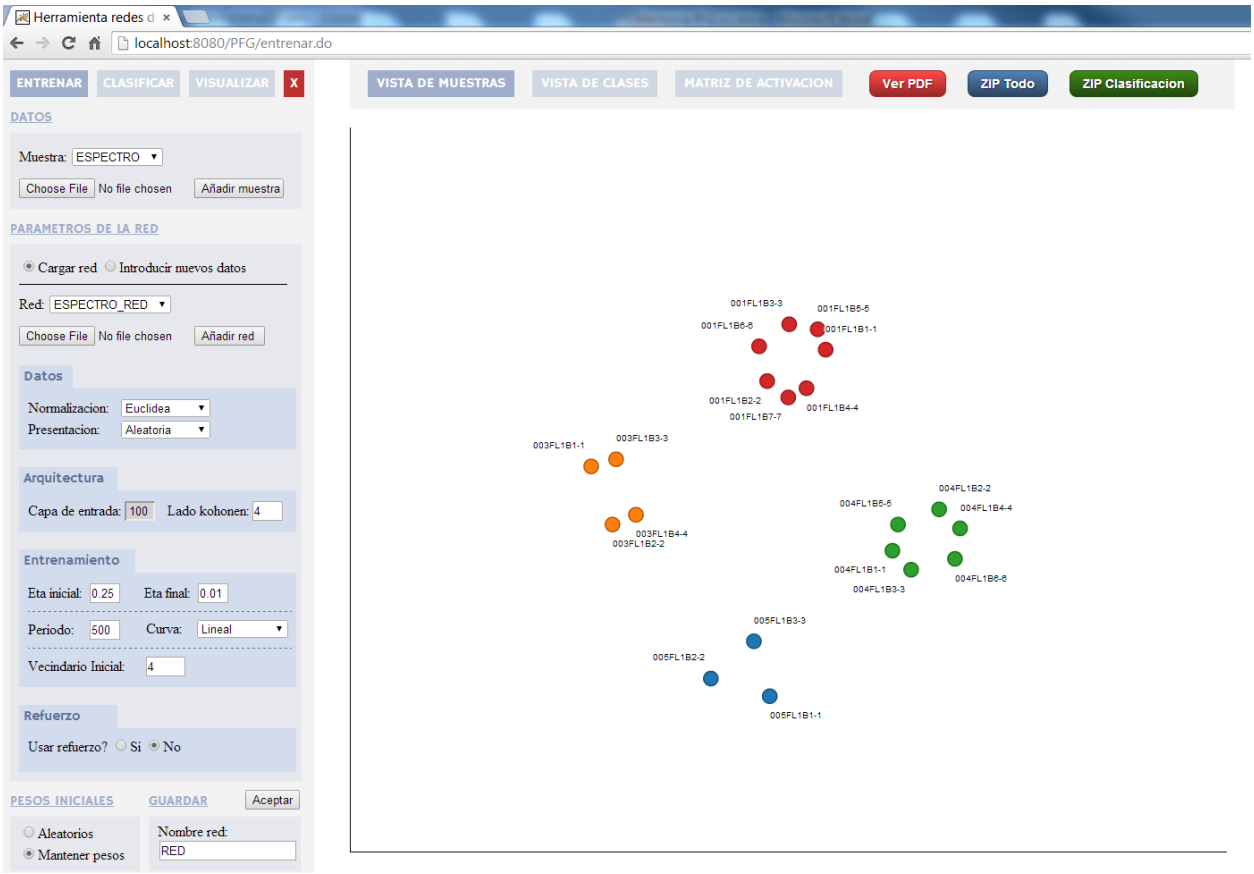

Figura 79: Resultados de entrenamiento y clasificación - Vista de muestras

Al hacer click en el botón de entrenar red, no sólo se entrena la red sino que automáticamente se clasifican las muestras utilizadas para el entrenamiento y se muestran los resultados de dicha clasificación. Como se puede observar, la interfaz gráfica ha sido actualizada con una barra superior con varias opciones:

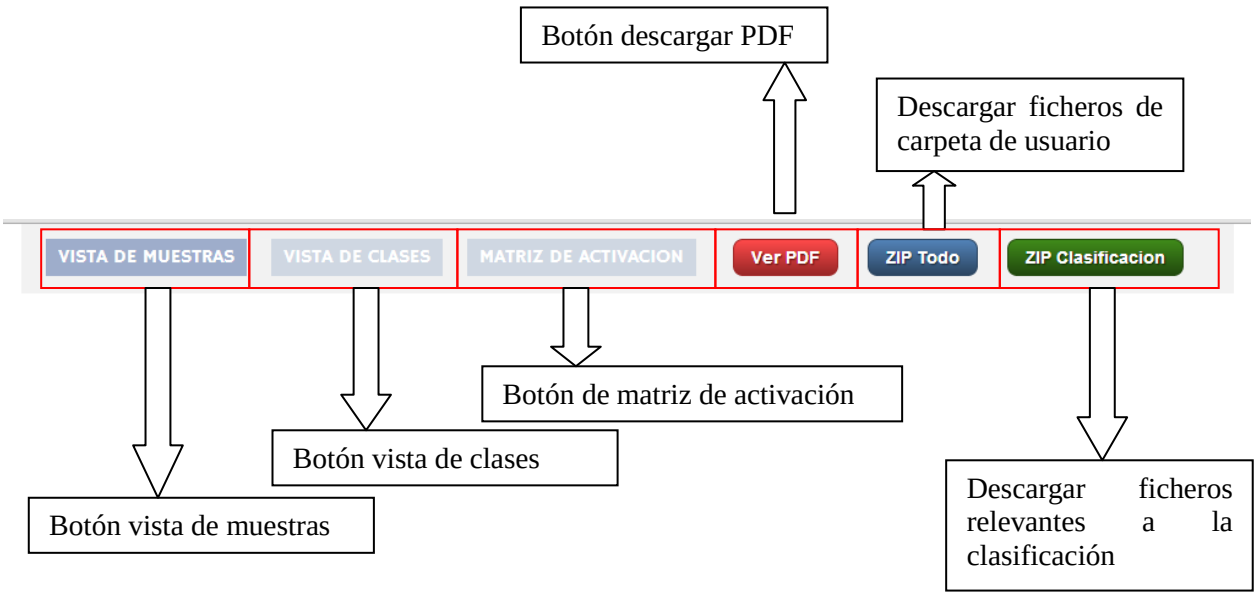

Figura 80: Interfaz de resultados

A su vez, se puede observar a continuación los resultados de las 3 vistas en más detalle:

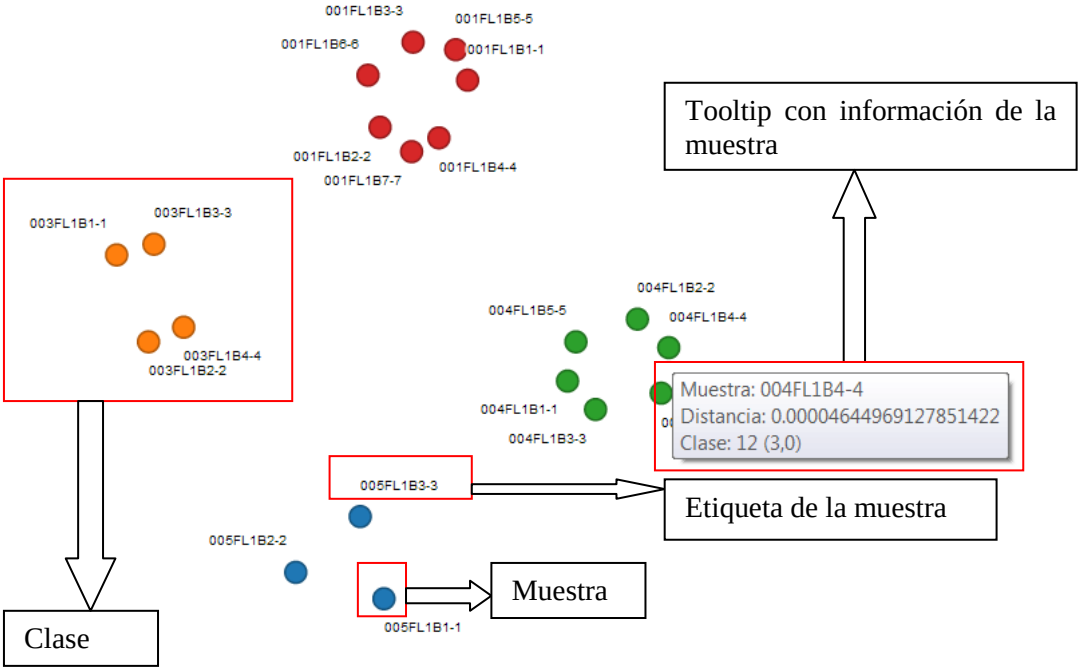

Figura 81: Vista de muestras

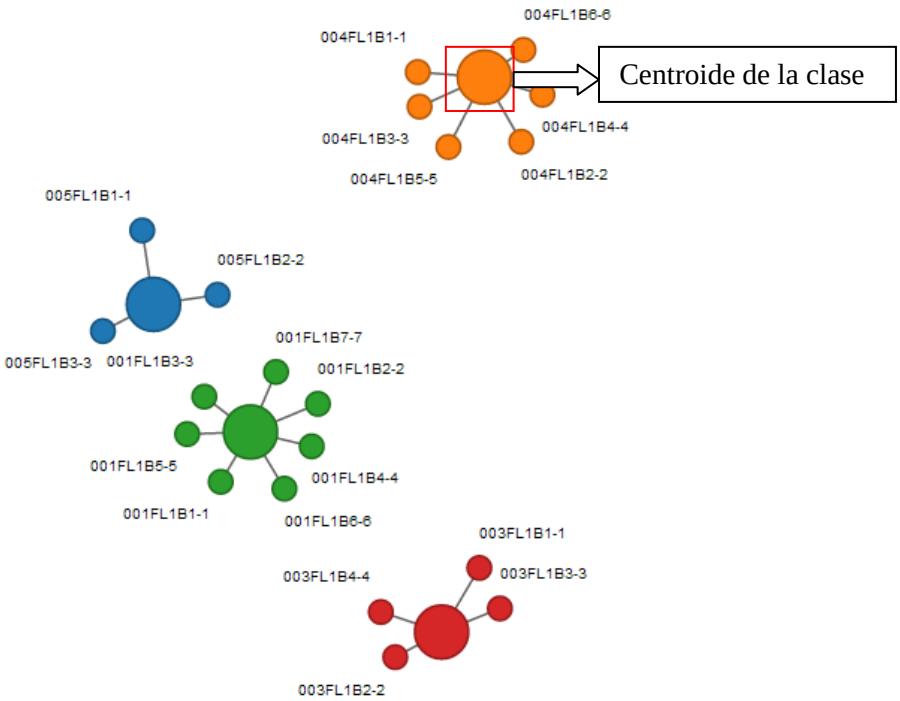

Figura 82: Vista de clases

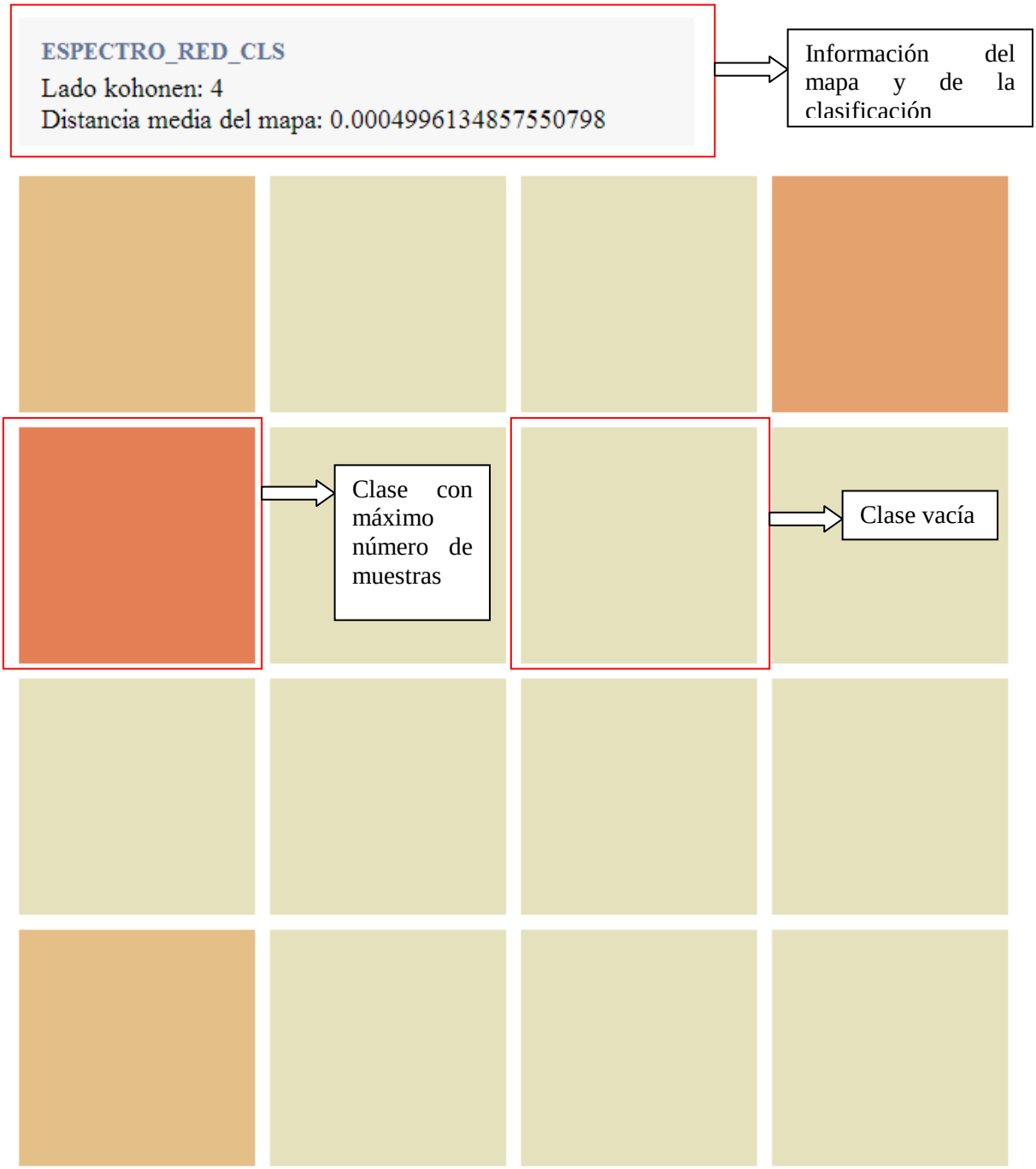

Figura 83: Matriz de Activación

Las últimas 3 figuras muestran los tipos de visualización de resultados ofrecidos por la herramienta. Cada uno tiene sus ventajas y desventajas, pero todos permiten al usuario comprender rápidamente la clasificación obtenida.

Por último, destacar que al hacer click en los botones de *Ver PDF, ZIP Todo y ZIP Clasificación,* se descargan automáticamente los ficheros correspondientes a la selección.

Tras ver los resultados y descargarlos, para salir de la herramienta basta con darle al botón rojo *X* que permite cerrar la sesión del usuario y destruir su carpeta.

CLASIFICAR MUESTRAS

Al hacer click en el botón *Clasificar*, se carga la pestaña de clasificación con sus parámetros correspondientes. Es importante mencionar que a continuación sólo se explica la interfaz y no los resultados obtenidos de visualización, ya que son exactamente iguales a los del entrenamiento.

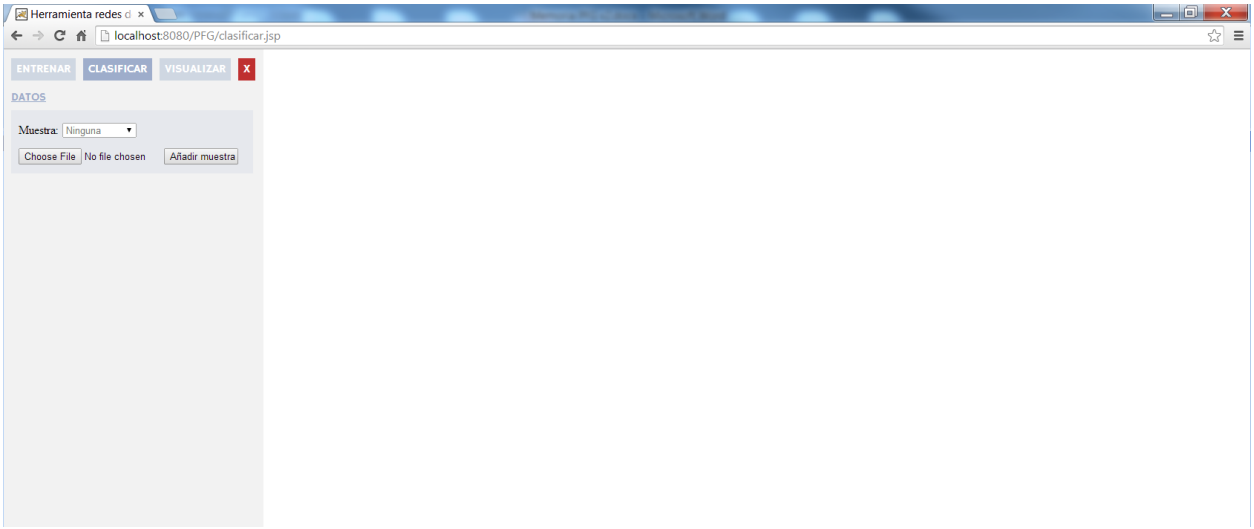

Figura 84: Interfaz Clasificar

La siguiente figura explica la barra izquierda de la herramienta, que contiene las opciones disponibles:

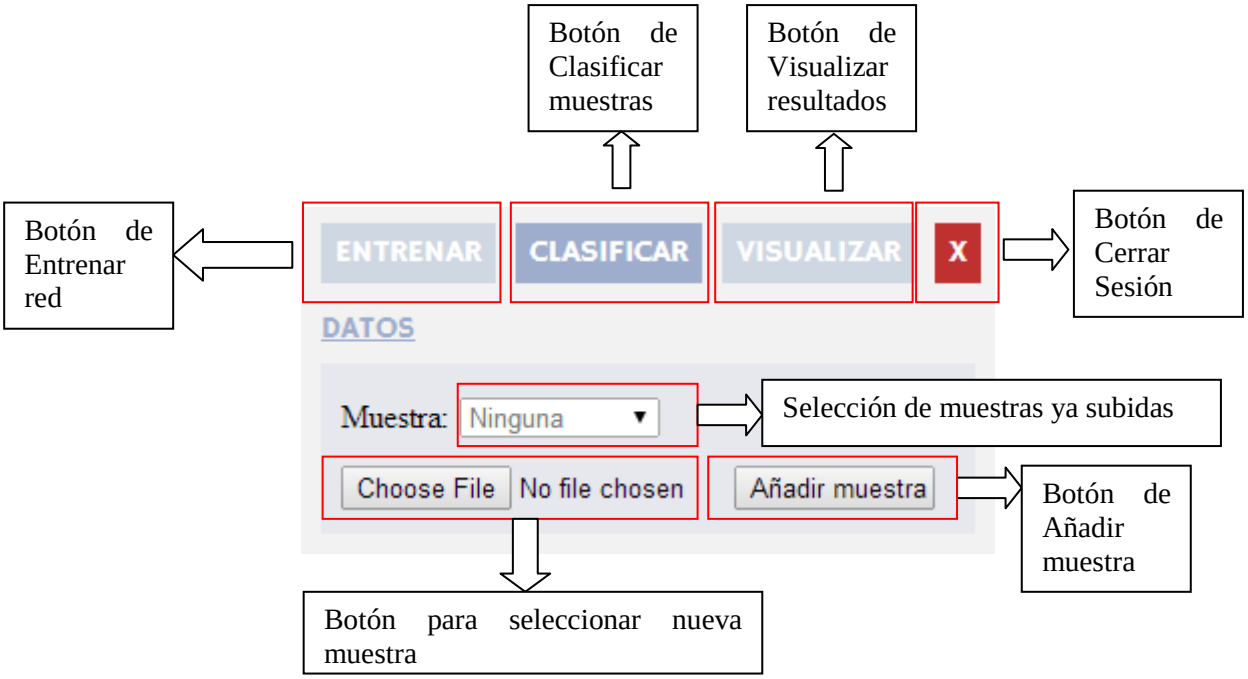

Figura 85: Interfaz Clasificar (1)

Es importante mencionar que las muestras son compartidas entre las pestañas de *Entrenar* y *Clasificar*, por lo que una muestra subida en una será visible en la otra.

Una vez cargada alguna muestra, se pasa a la selección de la red. En este caso, sólo es posible subir una red ya entrenada desde un fichero. En caso de que no esté entrenada, un mensaje de error se lo indicará al usuario:

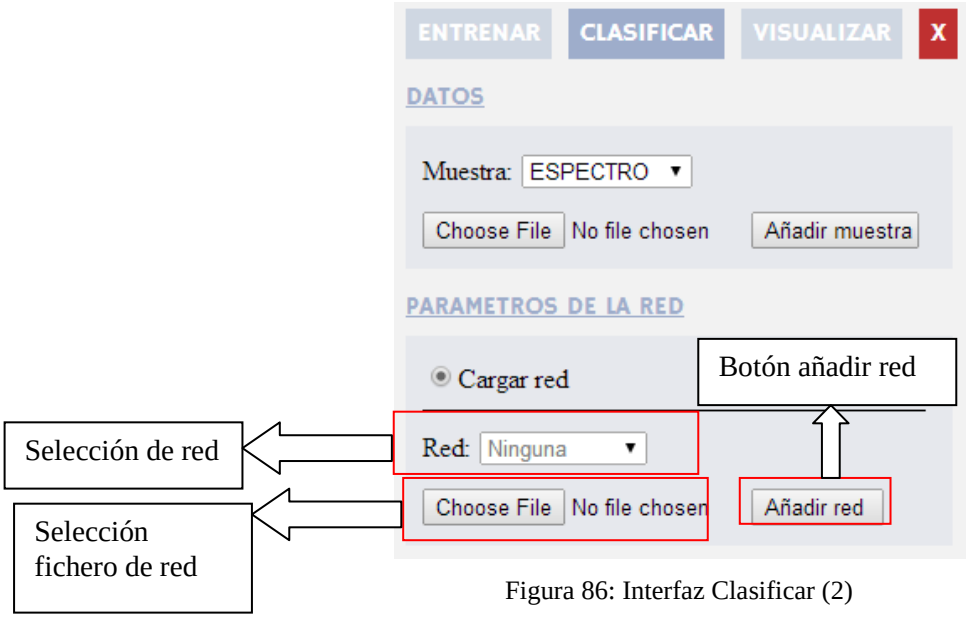

Figura 86: Interfaz Clasificar (2)

Figura 87: Interfaz Clasificar (3)

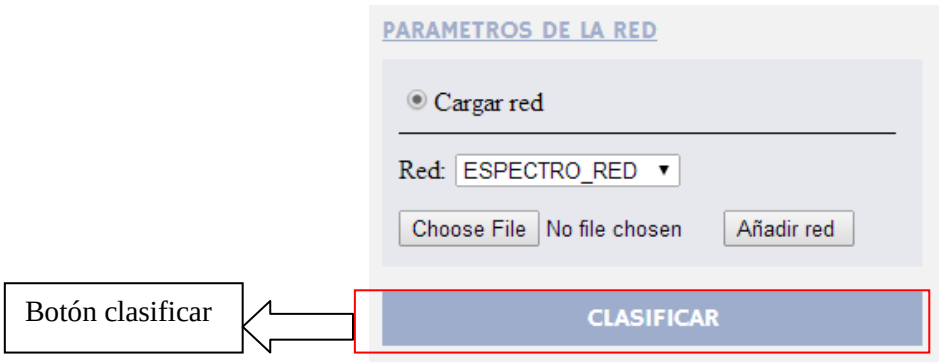

Figura 88: Interfaz Clasificar (4)

Tras seleccionar la muestra deseada y la red entrenada con la que se desea clasificar, se mostrarán los resultados ya explicados en el apartado de *Entrenar*. En este caso, sin embargo, se pasa directamente a la clasificación sin haber habido un entrenamiento.

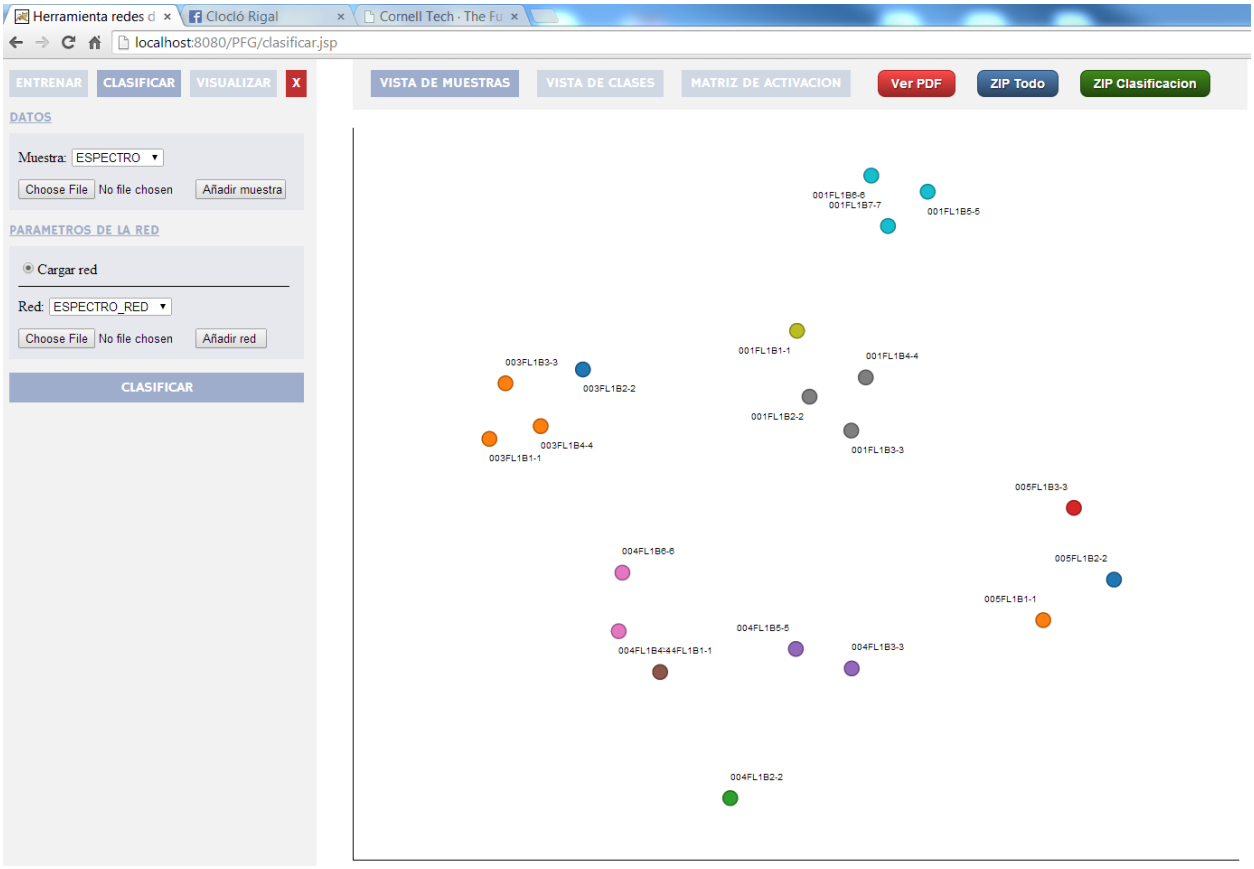

Figura 89: Interfaz Clasificar (5)

VISUALIZAR CLASIFICACIÓN

Al hacer click en el botón *Visualizar*, se activará la interfaz gráfica de visualización de resultados:

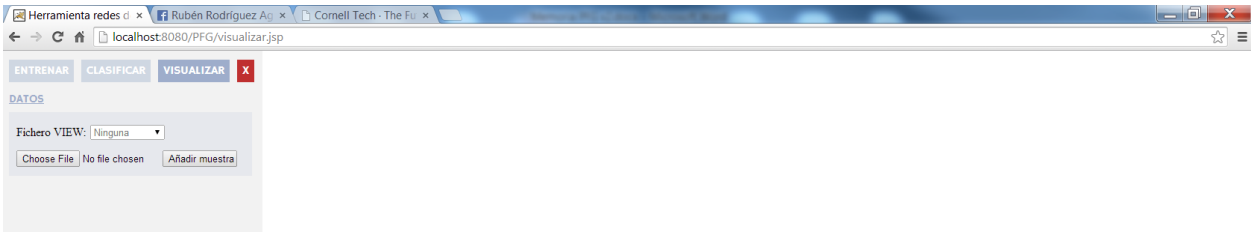

Figura 90: Interfaz Visualizar (1)

La siguiente figura explica la barra izquierda de la herramienta, que contiene las opciones disponibles:

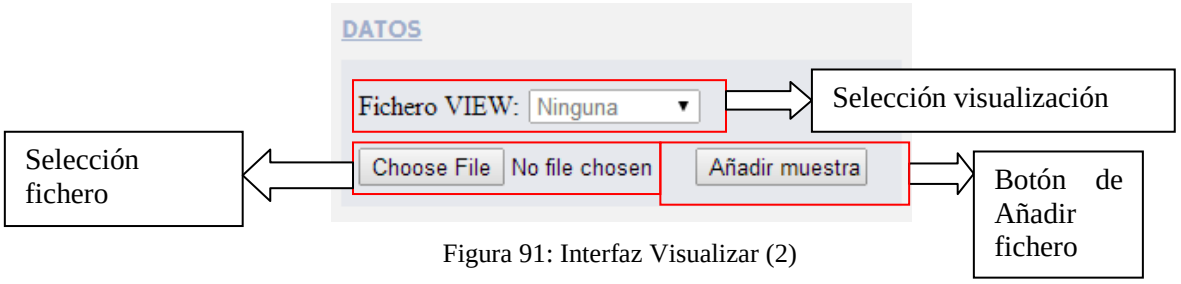

Figura 91: Interfaz Visualizar (2)

Tras subir un fichero de visualización y darle al botón *Visualizar*, se cargarán los resultados de la visualización. Cabe destacar que en este caso no se permite descargar los ficheros ZIP o PDF:

Figura 92: Interfaz Visualizar (3)

9.3 ANEXOS DIGITALES

- El código fuente mediante el cual la herramienta ha sido desarrollada se encuentra en la carpeta *Código* del CD que acompaña a esta memoria.
- La presentación Powerpoint utilizada para presentar el proyecto se encuentra en la carpeta Presentación del CD que acompaña a esta memoria.
- Ficheros *ESPECTRO.DAT* y *ESPECTRO.XML* para disponer de un ejemplo de muestras para probar el programa.