UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCOLA POLITECNICA SUPERIOR DE GANDIA

Grado en Ing. Sist. de Telecom., Sonido e Imagen

“Sistema multimodal para la evaluación del riesgo de cáncer de mama desde el enfoque de la minería de datos”

TRABAJO FINAL DE GRADO

Autor/a:
Jordi Moreno Claver

Tutor/a:
José Carlos Periñán Pascual

GANDIA, 2018

Resumen

A través de la minería de datos, se pueden desarrollar sistemas de recomendación que guíen las decisiones de los usuarios. El objetivo del trabajo es el diseño y desarrollo de una aplicación informática que, a partir de la información sobre un determinado paciente, pueda predecir si el tumor de mama en cuestión es benigno o maligno. Más concretamente, la información más relevante que se extrae del input a clasificar proviene de descriptores numéricos sobre el tumor, p.ej. radio, textura, área, etc. Este tipo de asistente médico realiza la predicción basándose en métodos supervisados de minería de datos. La entrada de datos del sistema es vía voz o texto escrito en español, tras lo cual se aplica un preprocesamiento del input con el fin de que el sistema pueda trabajar con datos estructurados. En la siguiente fase, se emplean métodos como Naïve Bayes, Support Vector Machines (SVM) y aprendizaje profundo con redes neuronales sobre datos de entrenamiento con el fin de que se detecten patrones que permitan la clasificación del input. Este sistema de predicción también es capaz de determinar qué método es más efectivo tras un proceso de autoevaluación. El sistema se programa en C# dentro del entorno de Microsoft Visual Studio.

Palabras clave: Minería de Datos; Naïve Bayes; Support Vector Machines (SVM); Redes Neuronales

Abstract

Through the data mining, can be developed recognizing systems that guide the decisions of the users. The aim of this work is to design and develop an informatics app, which uses the information about a determinate patient, in order to predict if a specific breast tumor is benign or not. More concretely, the most relevant information extracted from the input to be classified, is provided by numeric descriptors about the tumor, for instance radius, texture, area, etc. Based on supervised data mining methods this type of medical assistant make the prediction. The input data of the system are either via voice or text on Spanish, after that are applied a preprocessing of the input, so that the system work with structured data. In the next stage, we use methods like Naïve Bayes, Support Vector Machines (SVM) and deep learning with neuronal networks about training data, for detecting patterns which allow to classify the input. This prediction's system is also able to determine the most effective method, once the self-assessment is done. The application is developed with C#, within the Microsoft Studio's environment.

Key words: Data Mining; NB (Naïve Bayes); SVM (Support Vector Machines); NN (Neural Networks)

Lista de Tablas

Lista de Figuras

Índice

1. Introducción

El tema del trabajo es la realización de una aplicación que funciona como asistente médico, que está basada en tres modelos computacionales de minería de datos para hacer sus predicciones y sea completamente funcional vía voz. En concreto la aplicación aborda el campo del cáncer de mama, ayudando a determinar si cierto tumor es benigno o maligno dependiendo de sus características registradas numéricamente.

La aplicación se ha realizado completamente utilizando el entorno de programación de C#, Visual Studio. Se utilizaron librerías importantes de este como las del reconocimiento de voz. Los modelos computacionales se han implementado como un servicio web para su fácil reutilización en otras aplicaciones. Los modelos elegidos son: Redes Neuronales, Maquinas de Soporte Vectorial y Naïve Bayes. Estos modelos utilizan un corpus gratuito sobre cáncer de mama para entrenarse y realizar una predicción lo más efectiva posible sobre los nuevos pacientes. Se evalúan y enfrentan los resultados obtenidos por estos modelos para ver cuál es más efectivo.

Entre los objetivos del trabajo han estado el crear una aplicación con una interfaz, y dotar a esta aplicación de ciertas funcionalidades. A través de la aplicación se debían poder introducir nuevos pacientes, así como visualizar estos pacientes utilizando una base de datos que los almacenaba, y que había que diseñar. Se debía poder interactuar con los pacientes almacenados en la base de datos y además se debía hacer uso de un servicio web, cuya programación era uno de los objetivos, para poder diagnosticarlos.

Otro de los objetivos era la programación de tres modelos computacionales (Redes Neuronales, Support Vector Machines (SVM) y Naïve Bayes), estos modelos computacionales serían acomodados en un servicio web y serían consumidos mediante la aplicación comentada en el apartado anterior exponiendo un caso práctico donde podrían ser útiles. Estos modelos antes de alojarse en un servicio web, se programarían y evaluarían como aplicaciones independientes de consola. Finalmente se debía programar un cliente en la aplicación general para que se consumiera el servicio web.

Como objetivo último se hizo que todas las funcionalidades, comentadas en los dos apartados anteriores, estuvieran disponibles vía voz. Es decir, la aplicación general se debía poder utilizar por completo utilizando simplemente la voz.

Respecto a la metodología empleada en el trabajo se han utilizado ampliamente los conocimientos aprendidos y descritos en el *"Marco Teórico"*. Destacar que aunque existía la posibilidad de utilizar librerías existentes[1], programadas en C# u otros lenguajes, para implementar los modelos computacionales, se decidió programar todos los modelos de predicción a partir de los conocimientos teóricos y ejemplos obtenidos de libros y artículos empezando por el libro escrito por Bramer (2007). Con el objetivo de tener pleno control sobre estos módulos, y comprender la base matemática y teórica de la cual hacen uso continuamente.

La metodología del trabajo se ha realizado siguiendo un orden regido por un flujograma de datos previamente diseñado. Se intentó ir programado los módulos y afrontar los retos conforme estos eran necesarios para el funcionamiento de la aplicación según el diseño inicial. Puede verse todo este proceso desarrollado en el apartado de *"Metodología"*. En primer lugar, se hace uso de la comunicación entre servicio web y cliente, este servicio web contiene tanto los métodos necesarios para *"entrenar los módulos"* como los métodos utilizados durante el diagnóstico de los pacientes. Para guardar los pacientes se utiliza una base de datos externa, una vez introducidos los pacientes existe la posibilidad de eliminarlos, mediante instrucciones "*Structured Query Language"* (SQL), o diagnosticarlos, haciendo uso de los módulos de predicción alojado en el servicio web. Una vez han terminado de hacer sus cálculos los módulos de predicción, el cliente recibe el resultado final y lo expone en la interfaz de la aplicación.

Para asegurarse de que los módulos funcionaban correctamente se les aisló y se les evaluó como se explica en el apartado de *"Evaluación"*. Para introducir un paciente en la base de datos se hace uso de instrucciones de inserción SQL. Hay que tener en cuenta que cada vez que se utiliza el servicio web se hace a través de un cliente que pertenece a la aplicación principal. Para todas las interacciones e interoperabilidades entre usuario y aplicación se ha dado la posibilidad de utilizar tanto la voz como el sistema estándar de acción en interfaces de usuario, además se puede utilizar la voz sin perder ninguna funcionalidad.

No se puede hablar de las etapas del trabajo sin hacer referencia a la interfaz de la aplicación. El flujo de ventanas de la interfaz, y funcionalidades asociadas, de la aplicación

[1] Redes Neuronales: http://accord-framework.net/docs/html/N_Accord_Neuro.htm
SVM: http://accord-framework.net/docs/html/N_Accord_MachineLearning_VectorMachines.htm
Naïve Bayes: http://accord-framework.net/docs/html/T_Accord_Neuro_Learning_LevenbergMarquardtLearning.htm
General: http://accord-framework.net/docs/html/N_Accord_MachineLearning.htm

final han marcado ampliamente las etapas del trabajo, dado que esta fue diseñada de antemano. El flujo de la interfaz puede verse desarrollado en el apartado *"Interfaz de la aplicación"*. Se fue programando en Visual Studio desde el esqueleto hasta las funcionalidades más específicas de la aplicación. Se empezó por la parte de introducir el paciente, siguiendo con la comunicación con la base de datos, seguidamente se trabajó el visualizado de pacientes y su manipulación, lo que llevó a configurar el "*Servicio Web"* y su comunicación con la aplicación general, finalizando con la programación del reconocimiento de voz.

En cuanto a los problemas encontrados se pueden remarcar varios, desarrollados con mucho más detalle dentro del apartado *"Metodología"* junto con la solución que se les dio, conforme fueron apareciendo durante las diferentes etapas del trabajo. Uno de los problemas fue la programación de los módulos a partir de la teoría y los ejemplos de libros y artículos en lugar de utilizar librerías existentes. También supuso un reto la adaptación de los datos de entrenamiento para cada uno de los módulos de predicción, sobre todo para Naïve Bayes. Otro problema que surgió fue, el entrenamiento y registro del entrenamiento en directorios utilizando el *Servicio Web*. Además de lo complicado que resultó montar un instalador sencillo de todo el proyecto, que funcione en cualquier ordenador. Finalmente, el reconocimiento de voz propuso muchísimos retos, como la carga del gran volumen de gramáticas necesarias, y los conflictos entre estas gramáticas que se originaban durante el flujo de la aplicación, por último, el reconocimiento de números por voz supuso también mucho trabajo y tiempo para su resolución.

2. Marco Teórico

2.1. Descripción General

La mayor carga teórica de este proyecto se encuentra en los modelos computacionales Redes Neuronales, SVM y Naïve Bayes. Estos forman parte de los muchos modelos de minería de datos basados en la actual filosofía de la Inteligencia Artificial (IA) que se encuentran hoy en día. En los últimos años la inteligencia artificial ha ido involucrándose en cada vez más sectores, esto es debido a la gran cantidad de información que es captada y a la imposibilidad del ser humano para procesarla sin la computación. Pueden ser encontradas estas grandes cantidades de información en casi cualquier ámbito de en lo que se ha convertido la vida diaria, ya sea en las redes sociales, los complejos dispositivos de examinación utilizados en campos como la medicina, los servicios web o toda información proporcionada por el campo del “internet de las cosas”.

La IA se basa en utilizar modelos de predicción como los que se ven a continuación. Estos modelos de predicción son entrenados con información obtenida mediante mediciones o análisis precisos. La información ha de ser precisa, y se debe tener mucha, para que finalmente los modelos de predicción sean precisos y efectivos.

En este trabajo se utiliza la IA en un contexto médico, pero puede verse cómo hoy en día se usa en mucho otros ámbitos, como, por ejemplo: el reconocimiento de imagen para coches inteligentes (LeCun, Bengio, & Hinton, 2015), el reconocimiento de voz en los dispositivos móviles (Hinton et al., 2012), corrección ortográfica (Montiel, 2018), traducción de texto (López & Roca, 2006; García Gutiérrez, 2016), clasificación de páginas web (AbdulHussien, 2017), detección de eventos potencialmente peligrosos a través de las redes sociales (Lin et al., 2014; Zhang, He, Gao, & Ni, 2018; Nguyen, Joty, Imran, Sajjad, & Mitra, 2016), estudios demográficos (Gebru et al., 2017), etc.

Los principios en que se basan los modelos de predicción que utilizo ya han sido utilizados anteriormente en otros trabajos, tanto de investigación como prácticos, sobre medicina. De hecho, el campo de la medicina es uno de los que más está siendo afectado de forma positiva por los desarrollos en IA (Litjens et al., 2017; Xu et al.,, 2014; Bar, 2015; Bar et al., 2015; Ravì et al., 2016).

2.2. Módulos de Predicción

En este trabajo se ha decidido programar los módulos de predicción, en lugar de utilizar librerías existentes. La razón es que al utilizar librerías creadas por otros no se tiene control total sobre estas, y no se puede llegar a comprender totalmente ni el código ni los fundamentos teóricos que se utilizan. El haber programado desde cero los módulos, da la posibilidad de conocer todas las partes del proceso poco a poco, y un conocimiento más profundo de las matemáticas en las que se basan, también da la posibilidad de modificarlos para hacer que sean lo más óptimo posible para este trabajo y base de datos en concreto.

2.2.1. Red neuronal

Una red neuronal artificial es un sistema de software que modela de manera poco precisa las neuronas y sinapsis biológicas. Esta red neuronal por sí sola no es capaz de predecir nada. Como cualquier módulo de análisis, se necesita una base de datos con muchos casos diferentes e información fidedigna, que puede llamarse “Datos de entrenamiento” o “Corpus”. Se ha utilizado una base de datos de pacientes con cáncer de mama, en concreto quinientos sesenta y nueve casos. De cada caso dispongo de treinta datos de entrada además del identificador y el diagnóstico final, es decir, cáncer benigno o maligno.

Las redes neurales son básicamente una complicada función matemática que entiende únicamente números, como se ve posteriormente. Los datos de entrada, por tanto, exigen cierto tipo de tratamiento antes de ser utilizados. Como solo entiende números se debe pasar por un proceso de codificación del input en caso de tener algún input que no sea un valor numérico. Para los diferentes tipos de valores de un atributo existen varias formas de codificarlos, pero la más utilizada es asignar a cada tipo de valor una “posición específica”, marcada con un “1” en un vector de tantos ceros como tipos de valores haya en cierto atributo. Se puede ver un ejemplo de cómo funciona esta codificación, en la *Tabla 1*.

Género (Input original)	Género (Input codificado)
Femenino	0 0
Masculino	0 1

Tabla 1: Ejemplo de Codificación

Una red neuronal típica dispone de tres niveles de nodos: los nodos de entrada de datos, los ocultos y finalmente los de salida. El número de nodos de entrada y salida está determinado por la estructura de los datos del problema, pero el número de nodos ocultos

puede variar y normalmente se encuentra a través de la prueba y el error. En la *Figura 1* se puede ver una red neuronal sencilla compuesta por 5-nodos de entrada, 4-nodos ocultos, y 3-nodos de salida. Cada una de estas líneas representa un valor numérico llamado peso, por tanto, al referirse a los “Pesos” de una red neuronal se hace referencia a los valores representados por las líneas que unen los diferentes nodos entre sí. También existe un peso especial para los nodos ocultos y de salida conocido como “Sesgo” o en inglés “Bias”, representado en la *Figura 1* por las líneas rojas.

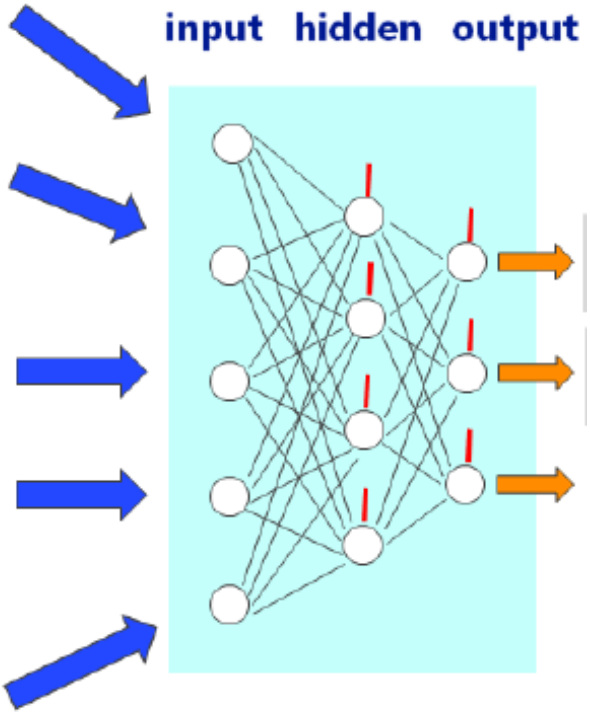


Figura 1: Red Neuronal (5·4·3)

Los valores de salida de una red neuronal están determinados por los valores de las entradas y los valores de los pesos y sesgos. Por lo tanto, la verdadera pregunta cuando se utiliza una red neuronal para hacer predicciones es cómo determinar los valores de las ponderaciones y los sesgos. Este proceso se llama entrenamiento (McCaffrey, 2014).

Dicho de otra forma, entrenar una red neuronal implica encontrar un conjunto de valores para los pesos y sesgos, de modo que cuando se presentan con los datos de entrenamiento, las salidas calculadas coinciden con los valores de salida conocidos y deseados.

Si se nombra a los valores de entrada valores-x y a los valores de salida valores-y, puede decirse que una vez que la red ha sido entrenada, se pueden presentar nuevos datos o nuevos valores-x con valores-y desconocidos, y se puede hacer una predicción. En este trabajo los valores-y son dicotómicos, es decir que la salida solo puede ser una de dos respuestas posibles, pero en otros casos se encuentran sistemas que deben dar una respuesta de una batería de respuestas posibles más grande que dos.

Además de aplicar una codificación en los valores de entrada que no sean numéricos, también se debe normalizar los valores numéricos. La normalización es un paso vital en la preparación de los datos para el posterior entrenamiento del sistema, sin normalizar se estarían comparando valores que no tiene sentido comparar entre sí. Por ejemplo, si en la base de datos existe un input (1) con valores muy altos y otro input (2) con valores bajos, cuando se apliquen las funciones matemáticas los valores del input (2) no está siendo tenidos en cuenta, prácticamente no influye en el resultado final.

Para la normalización, existen varias opciones. La más popular, simple y la que se utiliza finalmente en todo el proyecto es la "**Normalización Min-Max**". Otra de las opciones, también bastante popular, sería la "**Normalización Gaussiana**".

La normalización min-max consiste en aplicar la fórmula de la *fórmula (1)* a cada uno de los valores de un atributo o input concreto, este proceso se realiza para todos los inputs del corpus del trabajo. Siendo *Xinicial* el valor al que se aplica la fórmula y *Xfinal* el valor final después de las modificaciones. *Min* es el mínimo valor que se puede encontrar en uno de los inputs en concreto, y *max* sería el valor máximo de la misma "columna". Cuando se habla de "columna" siempre se hace referencia a cierto input y todos sus valores. En cambio, cuando se habla de una "fila" del corpus del trabajo, se alude a un objeto en concreto de toda nuestra base de datos, este objeto contiene un valor para cada uno de los inputs y un valor para output.

$$Xfinal = \frac{Xinicial - min}{\max - min} \quad (1)$$

La normalización Gaussiana se basa en el mismo procedimiento que la normalización min-max, solo que esta vez la función es diferente. Esta función se puede dividir en distintos pasos, como son:

- Primer paso, sacar la “media”. La media es igual a todos los valores de cierta columna sumados y divididos por el número de valores total. Se puede ver su representación matemática en la *fórmula (2)*.

$$media = \frac{\sum_{i=1}^{n} input_i}{n} \quad (2)$$

- Segundo paso, sacar la *desviación típica*, que consiste en hacer lo mismo que en la media, pero esta vez a cada valor se le resta la *media* al input y se pone al cuadrado, además de aplicársele la raíz cuadrada a todo. Se puede ver su representación matemática en la *fórmula (3)*.

$$desviacion\ tipica = \sqrt[2]{\frac{\sum_{i=1}^{n} (input_i - media)^2}{n}} \quad (3)$$

- Tercer y último paso, consiste en aplicar la fórmula de la *fórmula (4)* a cada uno de los valores de un atributo o input concreto, este proceso se realiza para todos los inputs del corpus del trabajo.

$$Xfinal = \frac{Xinicial - media}{desviacion\ tipica} \quad (4)$$

Para entender el comportamiento de una red neuronal es conveniente saber el comportamiento de uno de sus componentes más simples, el “**Perceptrón**”. Como destaca McCaffrey (2014, pág. 30) , el perceptrón es código que intenta modelar el comportamiento de una sola neurona:

> *A perceptron is software code that models the behavior of a single biological neuron. Perceptrons were one of the earliest forms of machine learning and can be thought of as the predecessors to neural networks. The types of neural networks described in this book are also known as multilayer perceptrons. Understanding exactly what perceptrons are and how they work is almost universal for anyone who works with machine learning. Additionally, although the types of problems that can be solved using perceptrons are quite limited, an understanding of perceptrons is very helpful when learning about neural networks, which are essentially collections of perceptrons.*

El mecanismo perceptrón de entrada-salida se ilustra en el diagrama de la *Figura 2*. El diagrama corresponde a la primera predicción de la *Figura 2,* donde las variables de entrada son x0 = 3.0 e x1 = 4.0, y los valores de pesos y sesgos determinados por el proceso de capacitación son w0 = 0.0065, w1 = 0.0123 y b= -0.0906 respectivamente. El primer paso para calcular la salida de un perceptrón es sumar el producto de cada entrada y el peso asociado de la entrada. Siguiendo la fórmula de la *fórmula (5).*

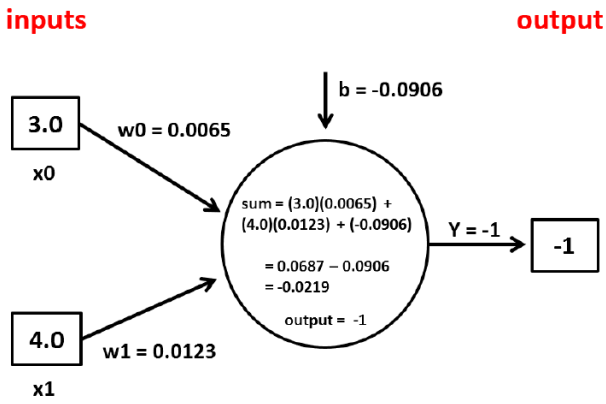


Figura 2: Mecanismo Input-Output del Perceptrón
(McCaffrey, 2014)

En el ejemplo de la *Figura 2*: $sum\ parcial = (3.0)(0.0065) + (4.0)(0.0123) = 0.0687$

$$sum\ parcial = x_0 \cdot w_0 + x_1 \cdot w_1 + \cdots + x_n \cdot w_n \quad (5)$$

El siguiente paso es sumar *sum parcial* y el sesgo, es decir:

$$sum = sum\ parcial + b \quad (6)$$

En el ejemplo de la *Figura 2*: $sum = 0.0687 + (-0.0906) = -0.0219$

El paso final es aplicar a la suma lo que se llama una "**Función de activación**". Las funciones de activación a veces se denominan funciones de transferencia. Existen varios tipos diferentes de funciones de activación, pero esta vez se utiliza una sencilla para poder entender otros conceptos. En esta función de activación, si sum es menor que 0, se dice que la salida es -1 y en cualquier otro caso es +1. Debido a que *sum* es -0,0219, la función de activación da -1 como salida del perceptrón. (McCaffrey, 2014)

El mecanismo de entrada-proceso-salida modela vagamente una sola neurona biológica. Cada valor de entrada representa una entrada sensorial o el valor de salida de alguna otra neurona. La activación de la función de pasos imita el comportamiento de ciertas neuronas biológicas que emiten o no emiten, dependiendo de si la suma ponderada de los valores de entrada supera algún umbral. En la *Figura 3* se puede ver la comparación entre una neurona y el sistema que se acaba de modelar, donde las “Ws” o “Weights” representan los pesos.

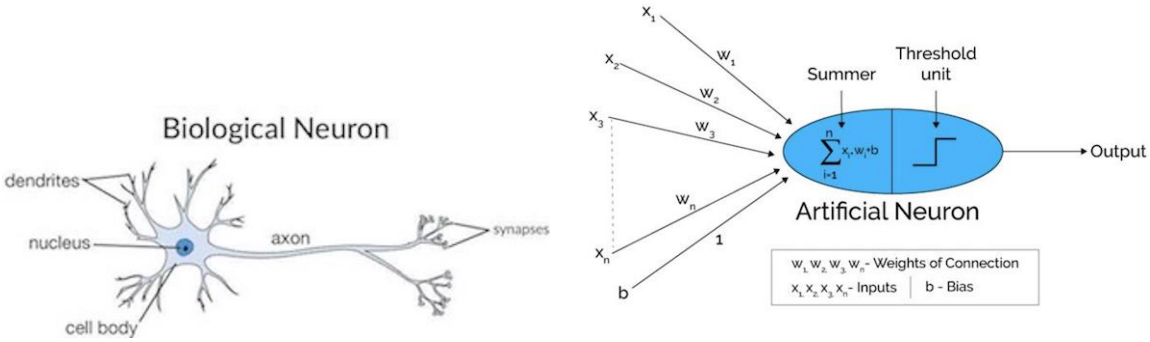


Figura 3: (a) Estructura de las neuronas en el cerebro (b) Analogía de la red neuronal artificial con la red neuronal biológica
(Chris, 2017)

Un valor de sesgo del perceptrón es sólo una constante que se añade a la suma de procesamiento antes de que se aplique la función de activación. En lugar de tratar el sesgo como una constante separada, muchas referencias tratan el sesgo como un tipo especial de ponderación o peso con un valor de entrada ficticio asociado de 1 (McCaffrey, 2014).

Uno de los pasos más importantes es el “**Entrenamiento**” del sistema. Para entender cómo se entrena el sistema por completo, es mejor entender primero cómo se entrena un perceptrón individualmente. El entrenamiento de un perceptrón es el proceso de ajustar iterativamente los valores de ponderación y sesgo para que las salidas calculadas para un conjunto dado de valores-x de datos de entrenamiento coincidan estrechamente con las salidas conocidas (McCaffrey, 2014).

El entrenamiento es bastante simple conceptualmente. El método de entrenamiento acepta como parámetros de entrada una matriz de datos de entrenamiento, un *"alfa"* de velocidad de aprendizaje y un límite de bucle que se llama *"maxEpochs"*. El parámetro *alfa* determina la velocidad a la que el sistema cambia el valor de los pesos durante el entrenamiento, es decir actúa como la derivada o pendiente de la curva que expone desde al valor inicial de los pesos, hasta el valor que adquiere finalmente. El parámetro *maxEpochs* determina el

número de iteraciones máximas que se realiza durante el entrenamiento. Además el error cuadrático medio se representa con el parámetro *"ECM"*, el *ECM* es una condición de parada del entrenamiento, y representa el error que se está cometiendo en las predicciones, si es inferior de un valor escogido se considera que el sistema ya está entrenado, el valor suele ser inferior o igual a 0,04 (Ephraim & Malah, 1984). El parámetro *maxEpochs* es importante ya que si el sistema estuviera mal modelado o se tuviese una mala base de datos podría no darse nunca la condición *ECM*, y por tanto el sistema seguiría entrenándose infinitamente sin que los pesos llegaran nunca a converger.

En muchas situaciones es preferible iterar a través de los elementos de datos de la formación utilizando un orden aleatorio cada vez a través del ciclo de procesamiento principal en lugar de utilizar un orden fijo. Para lograr esto, el método de entrenamiento usa un vector llamado secuencia. Cada valor en la secuencia de la matriz representa un índice en la fila de los datos de entrenamiento. Por ejemplo, si se tienen ocho elementos de entrenamiento y la secuencia de la matriz tuviera los valores {7, 1, 0, 6, 4, 3, 5, 2}, entonces la fila 7 de los datos de entrenamiento se procesaría en primer lugar, la fila 1 se procesaría en segundo lugar, y así sucesivamente.

Para alterar el orden de la secuencia que ha sido comentada anteriormente, se utiliza el "Algoritmo de Fisher-Yates" (Fisher & Yates, 1948). El cual se puede observar en la *Figura 4*.

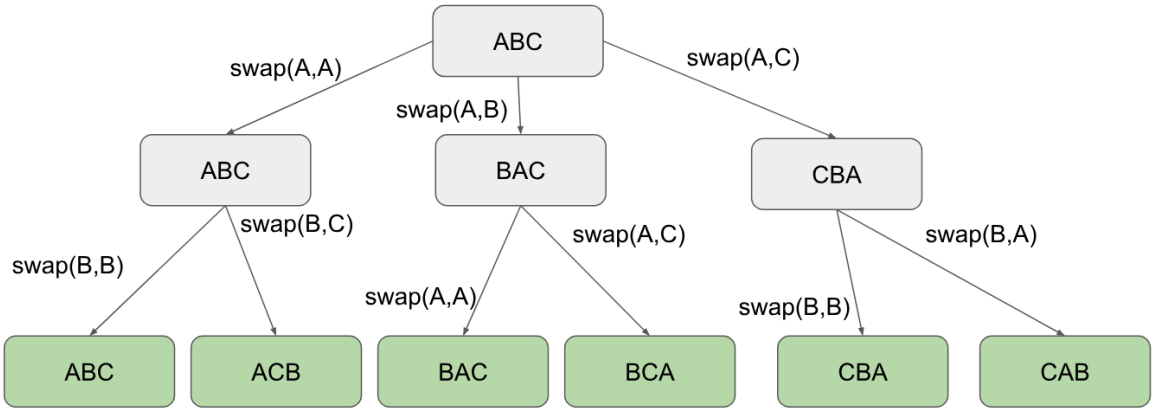


Figura 4: Ejemplo Implementación del "Algoritmo de Fisher-Yates".

Durante el entrenamiento, se van realizando una serie de pasos que terminan con la actualización de los pesos, los cuales dan forma al modelo final. En el método de actualización de pesos, se calcula la diferencia entre la salida calculada y la salida deseada, para cierto objeto/fila de la base de datos de entrenamiento, y se almatena en un parámetro llamado "Delta". *Delta* es positivo si la salida calculada es demasiado grande, o negativo si la salida calculada es demasiado pequeña. Para un perceptrón con salidas -1 y +1, el delta siempre es -2 (si se calcula = -1 y se desea = +1), o +2 (si se calcula = +1 y se desea = -1), o 0 (si se calcula igual).

Para cada $peso_i$, si la salida calculada es demasiado grande, el peso se reduce en la cantidad (alfa · delta · $entrada_i$). Si la $entrada_i$ es positiva, el término del producto también es positivo porque alfa y delta también son positivos, por lo que el término del producto se resta del $peso_i$. Si la $entrada_i$ es negativa, el término del producto es negativo, por lo que para reducir el $peso_i$ debe añadirse el término del producto. Para la actualización de los pesos se sigue la fórmula de la *fórmula (7)*:

$$Peso\ final = Peso\ inicial - (\text{alfa} \cdot \text{delta} \cdot \text{entrada}) \tag{7}$$

Se puede observar que el tamaño del cambio en un peso es proporcional tanto a la magnitud de delta como a la magnitud del valor de entrada asociado al peso. Así que un delta más grande produce un cambio más grande en el peso, y una entrada asociada más grande también produce un cambio más grande en el peso.

La velocidad de aprendizaje *alfa* escala la magnitud de un cambio de peso. Los valores más grandes de alfa generan cambios más grandes en el peso, lo que lleva a un aprendizaje más rápido, pero con el riesgo de sobrepasar un buen valor de peso. Los valores más pequeños de *alfa* evitan el rebasamiento, pero hacen que el entrenamiento sea más lento. El rebasamiento es el efecto provocado por valores altos de alfa, debido a este rebasamiento los pesos y los sesgos van oscilando en el entrenamiento entre demasiado y demasiado poco, por lo tanto, nunca llegan a converger en un valor determinado.

El parámetro sesgo no depende del input, y su actualización se rige por la fórmula de la *fórmula (8)*:

$$Bias\ final = Bias\ inicial - (\text{alfa} \cdot \text{delta} \cdot \text{entrada}) \tag{8}$$

Se puede observar que toda la lógica de actualización depende de la forma en que se calcula el valor de delta. En este caso se calcula arbitrariamente delta como (calculado - deseado). El proceso de entrenamiento consiste en iterar muchas veces este procedimiento hasta que finalmente delta sea más pequeña o igual al error que se esté dispuesto a cometer.

El gráfico de la *Figura 5* ilustra cómo funciona la predicción del perceptrón. Los puntos negro y rojo son los datos de entrenamiento. El punto azul es un objeto sin clasificar. Es bastante obvio que el nuevo objeto pertenece al grupo negro.

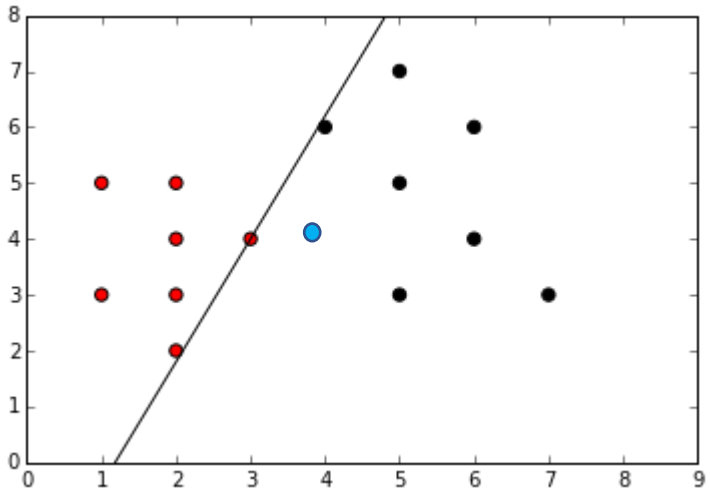


Figura 5: Funcionamiento de la Predicción de un Perceptrón
(Stamford, 2015)

El entrenamiento de un perceptrón encuentra esencialmente una línea recta de modo que todos los elementos de datos de entrenamiento en una estén a un lado de la línea y todos los elementos en la otra clase estén al otro lado de la línea. Esta característica de los perceptrones se llama separabilidad lineal. Se puede observar que hay muchas líneas de separación posibles.

Debido a que los perceptrones esencialmente encuentran una línea divisoria entre los datos de entrenamiento y las clases de valor, los perceptrones sólo pueden hacer buenas predicciones en situaciones en las que esto es posible. Dicho de otra manera, suponiendo que en la Figura 5 los datos de la capacitación se colocaron de tal manera que había una clara distinción entre rojo y negro, pero el espacio divisorio entre las clases era una línea curvada, en lugar de recta. Un simple perceptrón sería incapaz de manejar tal situación. En general, con los datos de la vida real no es posible saber de antemano si los datos de

entrenamiento son linealmente separables o no. Como destaca McCaffrey (2014, pág. 44), las redes neuronales tienen dos mejoras principales con respecto a los perceprones:

> *Neural networks have two main enhancements over perceptrons. First, a neural network has many processing nodes, instead of just one, which are organized into two layers. This is why neural networks are sometimes called multilayer perceptrons. Second, neural networks typically use activation functions which are more sophisticated than the step function typically used by perceptrons. Both enhancements increase the processing power of neural networks relative to perceptrons.*

A finales de la década de 1980, la investigación de la red neuronal demostró que, en términos generales, una red neuronal completamente conectada que utiliza una sofisticada función de activación puede aproximarse a cualquier función matemática continua. Esto se conoce como el teorema de la aproximación universal, o a veces el teorema de Cybenko (Cybenko, 1989). Lo que significa que las redes neuronales pueden clasificar con precisión los datos si es posible separar los datos en y utilizando cualquier línea o superficie lisa y curva.

Un perceptrón sólo puede clasificar correctamente cuando es posible dibujar una línea recta en el gráfico de modo que todos los puntos de datos negros estén a un lado de la línea y todos los puntos de datos rojos al otro lado, pero una red neuronal puede clasificar correctamente cuando es posible dibujar cualquier línea lisa y curva para separar las dos clases.

Una red neuronal es esencialmente una función matemática que acepta una o más entradas numéricas y produce una o más salidas numéricas. La computación básica de la red neural de “entrada-proceso-salida” se llama mecanismo de “**Retroalimentación**” o en inglés “**Feed-Forward**” (Hagan, 1994). Entender el mecanismo de retroalimentación es esencial para entender cómo crear redes neuronales que puedan hacer predicciones (McCaffrey, 2014).

En la *Figura 6* se puede ver una red neuronal con una sola capa de nodos ocultos, una de nodos de entrada y otra de nodos de salida. Cada nodo actúa como un perceptrón si se tienen en cuanta las líneas conectadas a este.

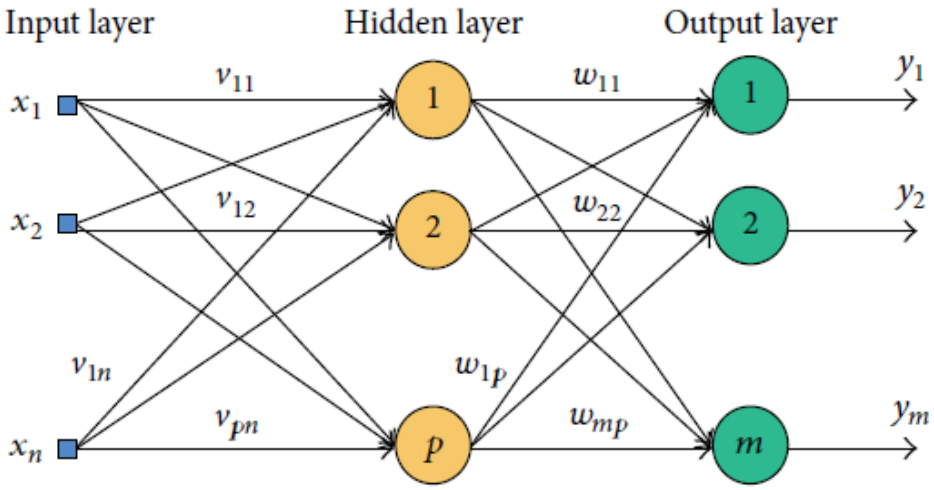


Figura 6: Red Neuronal (n·p·m)
(Lee Koo, Jing Liew, Mohamad, & Hakim Mohamed Salleh, 2013)

De acuerdo con la *Figura 6*, hay múltiples entradas, $x = (x1, x2, \cdots, xn)$ y salidas $y = (y1, y2, \cdots, yn)$. La suma ponderada de las aportaciones debe ser calculada por $u = (v \cdot x)$ y se define como se ve en la *fórmula (9)*:

$$\begin{pmatrix} u_1 \\ u_2 \\ u_n \end{pmatrix} = f\left(\begin{pmatrix} v_{01} & v_{02} & \cdots & v_{0n} \\ v_{11} & v_{12} & \cdots & v_{1p} \\ v_{p1} & v_{p2} & \cdots & v_{pn} \end{pmatrix} \cdot \begin{pmatrix} 1 \\ x_1 \\ x_n \end{pmatrix}\right) \tag{9}$$

A continuación, el resultado se utiliza para calcular la salida de la siguiente manera fórmula $y = f\ (w \cdot u)$ y se define como puede observarse en la *fórmula (10)*:

$$\begin{pmatrix} y_1 \\ y_2 \\ y_m \end{pmatrix} = f\left(\begin{pmatrix} w_{01} & w_{02} & \cdots & w_{0n} \\ w_{11} & w_{12} & \cdots & w_{1p} \\ w_{p1} & w_{p2} & \cdots & w_{mp} \end{pmatrix} \cdot \begin{pmatrix} 1 \\ u_1 \\ u_n \end{pmatrix}\right) \tag{10}$$

Véase que $v = (v1, v2, \cdots, vpn)$ y $w = (w1, w2, \cdots, wpn)$ son pesos que se utilizan para calcular y. Por lo tanto, el mapeo de entrada-salida puede resumirse de la siguiente manera $(x) = f\ (w \cdot y) = f\ (w \cdot (v \cdot x)))$ con una capa oculta (Lee Koo, Jing Liew, Mohamad, & Hakim Mohamed Salleh, 2013), o visto de otra forma $y = (w \cdot u) = (w \cdot (v \cdot x)))$.

Para tener en cuenta el parámetro sesgo o “pesos especial” en cada una de las operaciones realizadas, se considera un input adicional ficticio e igual a uno.

Cuando se habla la función **"$f(\cdot\cdot)$"** se hacer referencia a la función de activación, necesaria tanto en las salidas de la capa oculta como en la salida de la capa de salida. Aunque no suele ser la misma función para las dos capas.

> *Although there are some exceptions, in general the hyperbolic tangent function is the best choice for hidden layer activation. For output layer activation, if your neural network is performing classification where the dependent variable to be predicted has three or more values (for example, predicting a person's political inclination which can be "liberal", "moderate", or "conservative"), softmax activation is the best choice. If your neural network is performing classification where the dependent variable has exactly two possible values (for example, predicting a person's gender which can be "male" or "female"), the logistic sigmoid activation function is the best choice for output layer activation* (McCaffrey, 2014, pág. 62).

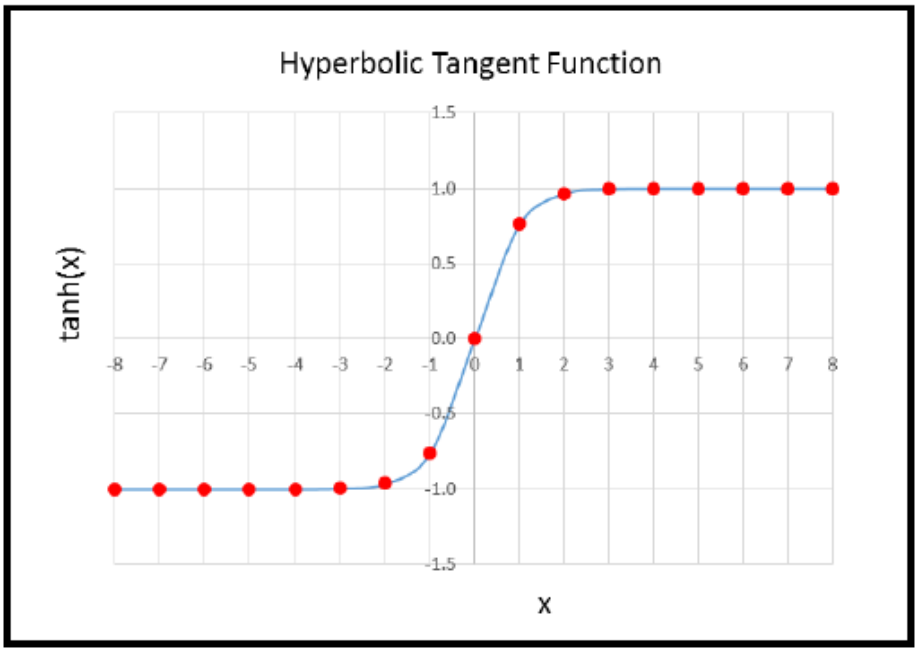


Figura 7: Función Tangente Hiperbólica

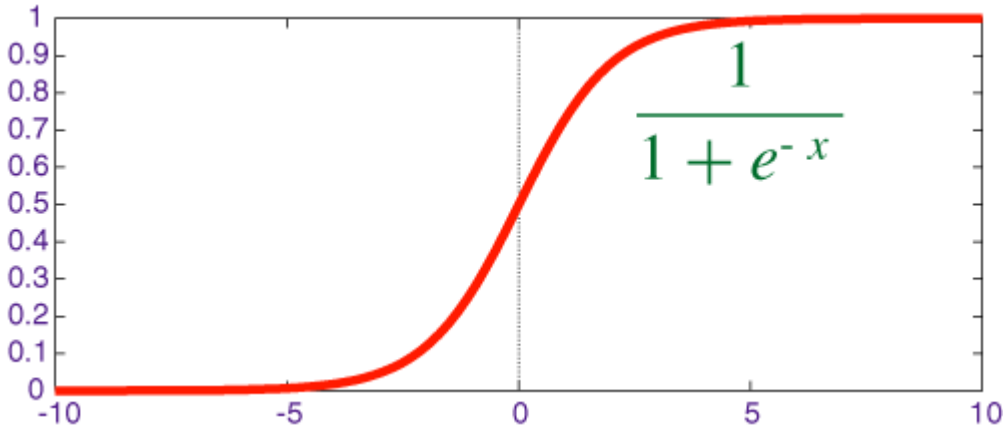


Figura 8: Función SoftMax, para salida dicotómica

Para más de dos tipos de salida, la función SoftMax, siendo *max* el mayor valor posible para la salida, es la *fórmula (11)*:

$$softmax(out_i) = \frac{e^{out_i - max}}{escala} \tag{11}$$

$$escala = e^{out_1} + e^{out_2} + \cdots + e^{out_n} \tag{12}$$

El entrenamiento de la red neuronal completa es muy parecido al entrenamiento de un solo perceptrón. El algoritmo más utilizado para el entrenamiento de redes neuronales es "**Retropropagación**" o en inglés "**Back-Propagation**".

La Retropropagación compara las salidas calculadas de la red neuronal (para un conjunto dado de entradas y valores de ponderación y sesgo) con los valores objetivo, determina la magnitud y dirección de la diferencia entre los valores reales y los valores objetivo y, a continuación, ajusta los valores de ponderación y sesgo de la red neuronal para que las nuevas salidas se acerquen a los valores objetivo. Este proceso se repite hasta que los valores de salida reales se aproximan lo suficiente a los valores objetivo, o hasta que se alcanza un número máximo de iteraciones.

El procedimiento a seguir en un algoritmo de retropropagación básico es el siguiente:

- Repetir los siguientes pasos hasta que se obtenga el error en las predicciones deseado:
  - calcular los valores de salida
  - calcular los gradientes de los nodos de salida
  - calcular los gradientes de los nodos de capa ocultos

- actualizar todos los valores de ponderación y sesgos

Para calcular los valores de salida se utiliza el algoritmo de retroalimentación visto anteriormente.

Los gradientes son valores que reflejan la diferencia entre los valores de salida calculados de una red neuronal y los valores deseados. Para el cálculo de los gradientes se utiliza la derivada del cálculo de la función de activación asociada a la capa a la que pertenezcan esos nodos. En el caso de este trabajo la función de la *fórmula (13)* para la capa de nodos ocultos y la función de la *fórmula (15)* para la capa de nodos de salida.

Se puede observar que los gradientes tienen una equivalencia directa con el parámetro *delta*, en el caso del sistema de un solo perceptrón. El gradiente de un nodo de salida es la diferencia entre el valor de salida calculado y el valor deseado multiplicado por el número derivado del cálculo de la función de activación utilizada por la capa de salida. Puede observarse en la *fórmula (13)*.

$$outGrad_{\ i} = f_{out}(calculado_{outi}) \cdot delta \tag{13}$$

$$delta = calculado - deseado. \tag{14}$$

El cálculo de los valores de los gradientes de los nodos ocultos utiliza los valores de los gradientes de los nodos de salida. El gradiente de un nodo oculto es el derivado de su función de activación, multiplicado por la suma de los productos de los gradientes de salida "descendente" y los pesos de salida asociados a los nodos ocultos. Puede observarse en la *fórmula (15)*.

$$hiddenGrad_i = f_{hidden}(calculado_{hidden}) * sum \tag{15}$$

$$sum = outGrad_1 \cdot w_{\ i1} + outGrad_2 \cdot w_{\ i2} + \cdots + outGrad_n \cdot w_{\ in} \tag{16}$$

Después de calcular todos los valores de gradiente de nodos ocultos y nodos de salida, estos valores se utilizan para calcular un pequeño valor delta (que puede ser positivo o negativo) para cada peso y sesgo. El valor delta se añade al valor de ponderación antiguo o al valor de sesgo. Las nuevas ponderaciones y los nuevos valores de sesgo producen, en teoría, valores de salida calculados que se aproximan más a los valores de salida deseados. Tal y como afirma McCaffrey (2014).

Se puede ver la representación matemática de la actualización del peso entre un nodo de entrada y otro oculto en la *fórmula (17)*.

$$v_{final} = (v_{inicial} + delta_{nueva}) + (v_{inicial} \cdot delta_{anterior}) \quad (17)$$

$$delta_{nueva} = alfa \cdot hiddenGrad \cdot \text{entrada} \quad (18)$$

Se puede ver la representación matemática de la actualización del peso entre un nodo oculto y otro de salida en la *fórmula (19)*.

$$w_{final} = (v_{inicial} + delta_{nueva}) + (w_{inicial} \cdot delta_{anterior}) \quad (19)$$

$$delta_{nueva} = alfa \cdot outGrad \cdot \text{entrada} \quad (20)$$

Con los pesos ya actualizados después de varias iteraciones hasta llegar al error deseado, ya puede realizarse una predicción de un objeto sin clasificar utilizando el algoritmo de retroalimentación con los nuevos inputs.

2.2.2. Support Vector Machine

Para “Support Vector Machine” (SVM), o “Máquinas de vectores de soporte”, se necesita hacer un tratamiento de la base de datos de entrenamiento idéntico al realizado para *Redes Neuronales*, tanto la parte de codificación, como para la parte de normalización. Para esta labor se utilizan los mismos algoritmos que ya se han visto anteriormente, en concreto en este trabajo se vuelve a hacer uso de la normalización min-max, después de numerosas pruebas que demostraban una mejora respecto a la normalización Gaussiana.

Los “vectores de soporte”, “pesos” y “sesgos” definen el modelo SVM entrenado. Se utiliza el modelo para generar valores de clase pronosticados para cada uno de los elementos de entrenamiento. Un valor de decisión negativo corresponde a una etiqueta de clase pronosticada de -1 y un valor de decisión positivo corresponde a una clase pronosticada de +1. El modelo entrenado debe predecir correctamente los elementos de entrenamiento del corpus del trabajo. Por ejemplo, siguiendo el ejemplo de McCaffrey (2019), *Figura 9*.

El objetivo cuando se entrena un sistema SVM es crear una regla que distinga entre los datos rojos y los azules de la *Figura 9*. El gráfico muestra un problema en el que los datos tienen sólo dos dimensiones (número de variables de predicción) sólo para poder visualizar el problema, pero los SVMs pueden trabajar con datos con tres o más dimensiones. (McCaffrey, 2019)

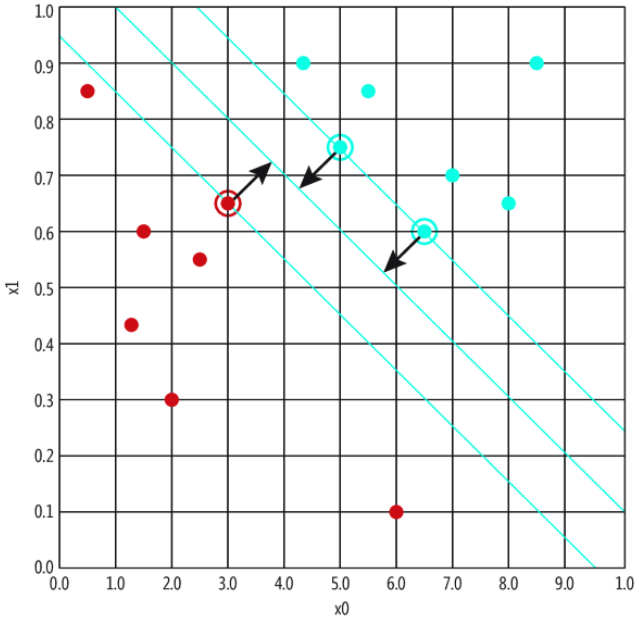


Figura 9: Concepto básico SVM encontrar el carril más ancho posible

En la *Figura 9* se puede ver como SVM trabaja buscando el carril más ancho posible que separa las dos clases y luego identifica uno o más puntos de cada clase que están más cerca del borde del *carril* de separación.

Para clasificar un nuevo punto de datos no visto anteriormente, lo que SVM hace es ver de qué lado del centro del carril cae el nuevo punto. En la *Figura 9*, el punto rojo marcado con un círculo en (0.3, 0.65) y los puntos azules marcados con un círculo en (0.5, 0.75) y (0.65, 0.6) se llaman “vectores de soporte”.

Hay tres grandes desafíos que deben ser resueltos para implementar un SVM utilizable. Primero, ¿qué hacer si los datos no son linealmente separables como en la *Figura 9*? Segundo, ¿cómo se encuentran los valores de los vectores de soporte, pesos y sesgos? Tercero, ¿cómo tratar los puntos de datos de entrenamiento que son anómalos y están en el lado equivocado del *carril*?

Hay muchos algoritmos que pueden ser usados para determinar los “vectores de soporte” para un problema de SVM. El algoritmo SMO (Sequential Minimal Optimization Algorithm) es el más común (Platt, 1998), y ha sido mejorado en cuestión de rapidez en los últimos años (Zeng, Yu, Xu, Xie, & Gao, 2008). Una explicación exhaustiva del funcionamiento del algoritmo básico SMO requeriría dedicarle un trabajo aparte. Como explicó Platt (1998), existen tres métodos básicos para la formación de los SVM, que se puede ver en la *Figura 10*.

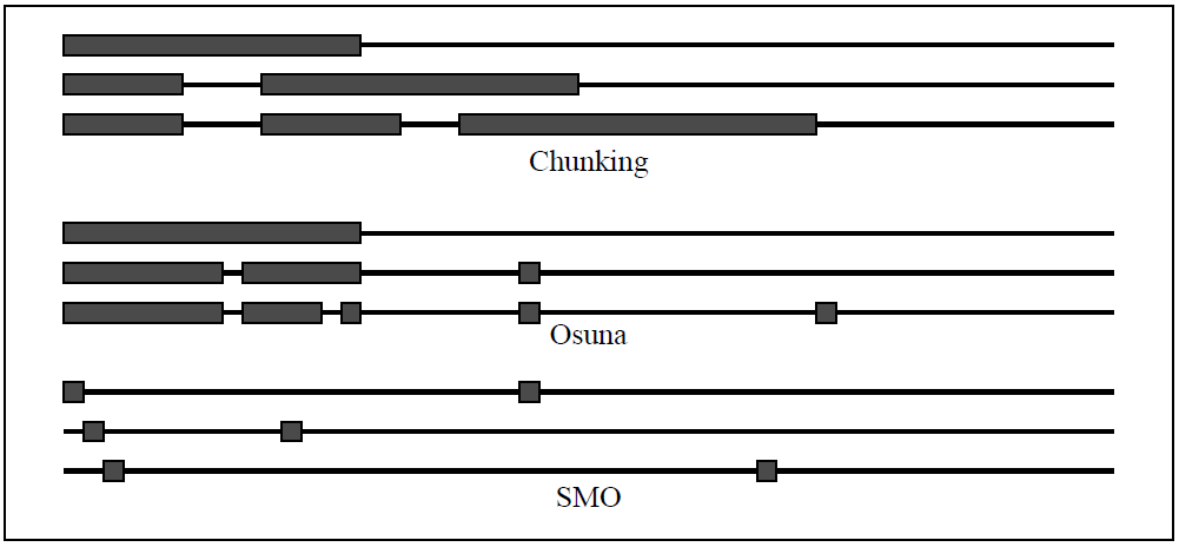


Figura 10: Tres métodos para la obtención de los Vectores de Soporte

Como se puede ver en la *Figura 10* tres métodos alternativos para la formación de los SVM: Chunking, el algoritmo de Osuna, y SMO. Para cada método, se ilustran tres pasos. La línea delgada horizontal en cada paso representa el conjunto de entrenamiento, mientras que las cajas gruesas representan los multiplicadores de Lagrange que se están optimizando en ese paso. Para el troceado, se agrega un número fijo de ejemplos en cada paso, mientras que los multiplicadores de Lagrange de cero son descartados a cada paso. Así, el número de ejemplos entrenados por paso tiende a crecer. Para el de Osuna, se optimiza un número fijo de ejemplos en cada paso: el mismo número de ejemplos es añadido al problema y descartado del mismo en cada paso. Para el SMO, sólo dos ejemplos son analíticamente optimizados en cada paso, de modo que cada paso es muy rápido (Platt, 1998).

En general, SMO tiene tres funciones claves, una de alto nivel que es la “función de entrenamiento”, una que es la “función de examinación” y finalmente otra llamada “función de mejora”. El algoritmo que constituye la función de mejora intenta encontrar una solución mejor para el sistema utilizando dos objetos cada vez de la base de datos de entrenamiento, obteniendo verdadero si se encuentra una mejora y falso de lo contrario. Para encontrar esta mejora se utiliza un método similar a redes neuronales, donde se compara el resultado obtenido de la computación del sistema y el resultado que se debería haber obtenido según la base de datos de entrenamiento. Obviamente esta comparación se realiza sobre un objeto o fila concreta de la misma base de datos, y según el nivel de error o de acierto resultante de esta comparación se considera que ha habido una mejora o no.

De la función de examinación, se obtiene el número de cambios que se han producido para que la función de mejora pueda determinar que se ha realizado un progreso.

Tanto la función de mejora como la de examinación utilizan una variable de alcance de clase llamada complejidad. Los valores más grandes de complejidad penalizan cada vez más los datos de entrenamiento atípicos y obligan al algoritmo SMO a intentar encontrar una solución que se ocupe de ellos, a expensas del ajuste excesivo de los modelos (McCaffrey, 2019).

En la función de entrenamiento se iteran las funciones de examinación y de mejora hasta que finalmente se cumple con los criterios de error admitido. Cuando finaliza el proceso se tienen los pesos, los sesgos y los vectores de soporte del sistema ya entrenado; SVM utiliza estos tres parámetros para predecir la clasificación nuevos objetos.

Se puede tratar con datos no linealmente separables usando lo que se llama una función de “núcleo” o en ¡¡nglés “kernel”. Con estas funciones, se pueden determinar los vectores de soporte, pesos y sesgos utilizando SMO. Y se puede tratar con datos de entrenamiento inconsistentes utilizando una idea conocida como “complejidad”, que penaliza los datos que no están “en el lado que debieran”. Una función de núcleo toma dos vectores y la combina para producir un único valor escalar. En principio, las funciones de núcleo simplemente sacan el escalar entre dos vectores, pero muchas veces se emplea un factor de escalado llamado “gamma”, con un factor de potencia llamado “grado” o con una constante “C”, como puede verse en la *fórmula (21)*.

$$f_{kernel}(v_1, v_2) = \left(gamma \cdot \left(v_1(1) \cdot v_2(1) + v_1(2) \cdot v_2(2) + \cdots + v_1(n) \cdot v_2(n)\right) + C\right)^{grado} \quad (21)$$

La elección de la función kernel que se va a utilizar y los valores de los parámetros que se aplican a la función kernel que se está utilizando son parámetros libres y deben determinarse mediante prueba y error.

2.2.3. Naïve Bayes

Para “Naïve Bayes Classifier” o “Clasificador Bayesiano” necesita hacerse un tratamiento de la base de datos idéntica a la realizada para Redes Neuronales, tanto la parte de codificación, como para la parte de normalización. Para esta labor se utilizan los mismos algoritmos que ya se han visto anteriormente, en concreto en este trabajo se vuelve a hacer uso de la normalización min-max, después de numerosas pruebas que demostraban una mejora respecto a la normalización Gaussiana. Pero en adición, y debido a que Naïve Bayes no puede trabajar con valores numéricos continuos, se deben “discretizar” todas las muestras de nuestra base de datos. Para discretizar, lo más óptimo es previamente normalizar todos los valores, y una vez normalizados elegir en cuántos rangos de valores se dividen todos los valores contenidos en el rango de 0 a 1. Se puede ver en el ejemplo de la *Tabla 2* cómo sería discretizar dividiendo el rango de valores original en 5. En este caso, se han elegido valores discretos para que sea más representativo del valor inicial, pero el clasificador Bayesiano podría funcionar también con cualquier otra cadena de texto, siempre y cuando sea cuantificable el número de veces que aparece en la base de datos.

Nº de Rango	Valores continuos	Valores Discretos
1	[0,0.2[	“1”
2	[0.2,0.4]	“2”
3	[0.4,0.6]	“3”
4	[0.6,0.8]	“4”
5	]0.8,1]	“5”

Tabla 2: Ejemplo de Discretización con 5 Cuartiles

Un clasificador de Bayes asume que la presencia o ausencia de una característica particular no está relacionada con la presencia o ausencia de cualquier otra característica, dada la clase variable. Por ejemplo, un cáncer de mama puede ser considerado como maligno si su radio es “2”, su textura “5” y su compacidad “3”, hablando en términos de valores discretos de la *Tabla 2*. Un clasificador de Bayes considera que cada una de estas características contribuye de manera independiente a la probabilidad de que este cáncer de mama sea maligno, independientemente de la presencia o ausencia de las otras características.

El modelo de clasificación Bayesiana se basa en probabilidades condicionadas y no condicionadas, siendo el modelo de probabilidad el que puede verse en la *fórmula (22)*, donde *F es* una de las características de un objeto y *C es* una de las clasificaciones posibles para cualquier objeto. En el ejemplo del cáncer de mama, *F* podría ser la textura, y *C* maligno. Además, $p(A|B)$ significa la probabilidad de tener *B* si ya se tenía *A*. Recordar que, $p(A|B) = \frac{p(A) \cdot p(B|A)}{p(B)}$.

$$p(C|F_1, F_2, \cdots, F_n) \tag{22}$$

$$p(C|F_1, F_2, \cdots, F_n) = \frac{p(C) \cdot\ p(F_1, F_2, \cdots, F_n|C)}{p(F_1, F_2, \cdots, F_n)} \tag{23}$$

Por tanto, el entrenamiento de un clasificador Bayesiano consiste simplemente en contar todos los casos posibles de todas las características de nuestra base de datos de entrenamiento. Una vez contado esto, se puede obtener la probabilidad para un input de una característica concreta, de ser clasificado finalmente como una cosa u otra. Sabiendo las veces que cierto tipo de input, de una característica concreta, ha sido encontrando en el corpus, condicionado a cierto valor de clase, es fácil obtener todas las probabilidades condicionadas o no, que se necesiten. Simplemente siguiendo la fórmula de la *fórmula (24)*, se puede saber la probabilidad de que C = maligno = 1, y también de que sea benigno, por deducción, si el sistema es dicotómico. Teniendo en cuenta que el ejemplo anterior se dice que sus atributos son radio “2”, textura “5” y compacidad “3”.

$$p(C = 1 \mid radio = \text{"2"} \,\&\, textura = \text{"5"} \,\&\, compacidad = \text{"3"}) \tag{24}$$

$$p(C = 1) = p(radio = \text{"2"} \mid C = 1) \cdot p(textura = \text{"5"} \mid C = 1) \cdot p(compacidad = \text{"3"} \mid = 1) \tag{25}$$

$$p(radio = 2 \mid C = 1) = \frac{n^{\underline{o}}\ (radio = 2\ \&\ C = 1)}{n^{\underline{o}}\ (radio = 2)} \tag{26}$$

Con la filosofía de la *fórmula (24)* se puede calcular cualquier caso, con un número cualquiera de atributos dados, si se han registrado todos los casos del tipo $n^{\underline{o}}\ (radio = 2\ \&\ C = 1)$ donde el *radio* sería una representación práctica de lo que significa *F* en la *fórmula (22)*.

2.3. Evaluación

La evaluación de la precisión suele ser bastante similar para todos los modelos de aprendizaje automático. En este trabajo se realiza un estudio de **"Validación cruzada"** (k-fold cross validation) (Kohavi, 1995). Para entenderla, se tiene que describir primero cómo se hace una evaluación básica de un módulo de predicción.

En una evaluación básica se separa la base de datos de entrenamiento de en dos partes. Se suele coger un 20% de las muestras para una de ellas y un 80% para la otra, aunque esto es simplemente una convención. Una de las partes, que suele ser la más voluminosa, se utiliza para entrenar al sistema, y la otra parte se utiliza para hacer un test al sistema ya entrenado. Para hacer ese test, se compara la predicción de cada objeto de la parte del corpus destinada al test con los datos reales. Es decir, se comparan los datos calculados con los deseados, y se obtiene lo que se llama "Parámetros de evaluación".

Lo único que cambia a la hora de hacer el estudio de validación cruzada es la forma en la que se divide el corpus para hacer el test. El factor clave de este estudio es el valor de "***k"***, el cual determina el número de divisiones llevadas a cabo sobre el corpus. Por ejemplo, suponiendo que $k = 3$, entonces se tienen tres subgrupos del grupo original que era nuestra base de datos de entrenamiento. En primer paso es coger el primer subgrupo y utilizarlo como datos para hacer el test, los otros dos grupos se utilizan como datos de entrenamiento, con esta configuración se hace el test y se sacan los *parámetros de evaluación* para obtener conclusiones. En segundo paso es repetir el primer paso, pero esta vez el segundo grupo es el que se utiliza para el test, con este procedimiento se podrían evaluar tantos subgrupos como objetos tenga el corpus. Una vez sacados los *parámetros de evaluación* de las tres iteraciones, se hacen las medias de estos.

Los *parámetros de evaluación* se basan, como ya se ha dicho, en la comparación de los datos calculados y los deseados. El caso que se analiza, utiliza clasificación dicotómica, siendo un "Positivo" cuando se obtiene de la predicción un "1" y un "Negativo" cuando se obtiene de la predicción un "0". Una vez comparado con los datos deseados según el subgrupo del corpus para hacer el test, se puede distinguir entre cuatro tipos de resultados en toda la casuística, como se ve en la *Tabla 3*.

Clasificación	Calculado	Deseado
“Verdadero Positivo"	“1”	“1”
“Falso Positivo”	“1”	“0”
“Verdadero Negativo”	“0”	“0”
“Falso Negativo”	“0”	“1”

Tabla 3: Clasificación Dicotómica (Falso/Verdadero) (Positivo/Negativo)

También se puede ver la forma más estándar de representación de la *Tabla 3* en la *Tabla 4,* que recibe el nombre de matriz de confusión.

		Predicción	
		Positivos	**Negativos**
Observación	**Positivos**	Verdaderos Positivos (VP)	Falsos Negativos (FN)
	Negativos	Falsos Positivos (FP)	Verdaderos Negativos (VN)

Tabla 4: Matriz de Confusión

Con este tipo de estudio, es bueno cerciorarse de que los resultados a la hora de hacer la predicción son fruto de un buen modelado y no de la casualidad, ya que esta casualidad podría jugar a favor en algunas ocasiones, dependiendo de qué objetos se cojan para el test y cuáles para el entrenamiento. La casualidad también podría favorecer o desfavorecer dependiendo del orden en que se cojan los objetos, por eso es preciso “barajar” los subgrupos antes de utilizarlos, con el algoritmo de Fisher-Yates, que ya ha sido visto en el apartado de redes neuronales.

Finalmente, los parámetros de evaluación se calculan a partir de la matriz de confusión de la *Tabla 4*. En este sentido, las medidas más características son las siguientes:

$$Accuracy = \frac{(VP + VN) \cdot 1}{(VP + VN) + (FP + FN)} \quad (27)$$

$$Precision = \frac{VP}{VP + FP} \quad (28)$$

$$Recall = \frac{VP}{VP + FN} \quad (29)$$

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \quad (30)$$

Cada uno de los parámetros de evaluación tiene diversas interpretaciones, pero en general cuanto más altos sean mejor. Aunque una buena apreciación es que sería bueno sacrificar la precisión (precisión) en aras de conseguir más cobertura (*recall*), sobre todo porque en muchas aplicaciones un *FN* suele tener más implicaciones críticas que un *FP*. Las aplicaciones médicas como las de este trabajo son un buen ejemplo.

3. Metodología

3.1. Flujo de Datos

En este apartado se describe como los datos fluyen por la aplicación, este flujo está determinado por las propias funcionalidades de la aplicación y como el propio usuario inicia ciertos procesos. Es importante saber que todos y cada uno de los procesos que se van a describir a continuación utilizando la *Figura 11* como apoyo han sido desarrollados desde cero para este trabajo, incluida la programación de los módulos de predicción.

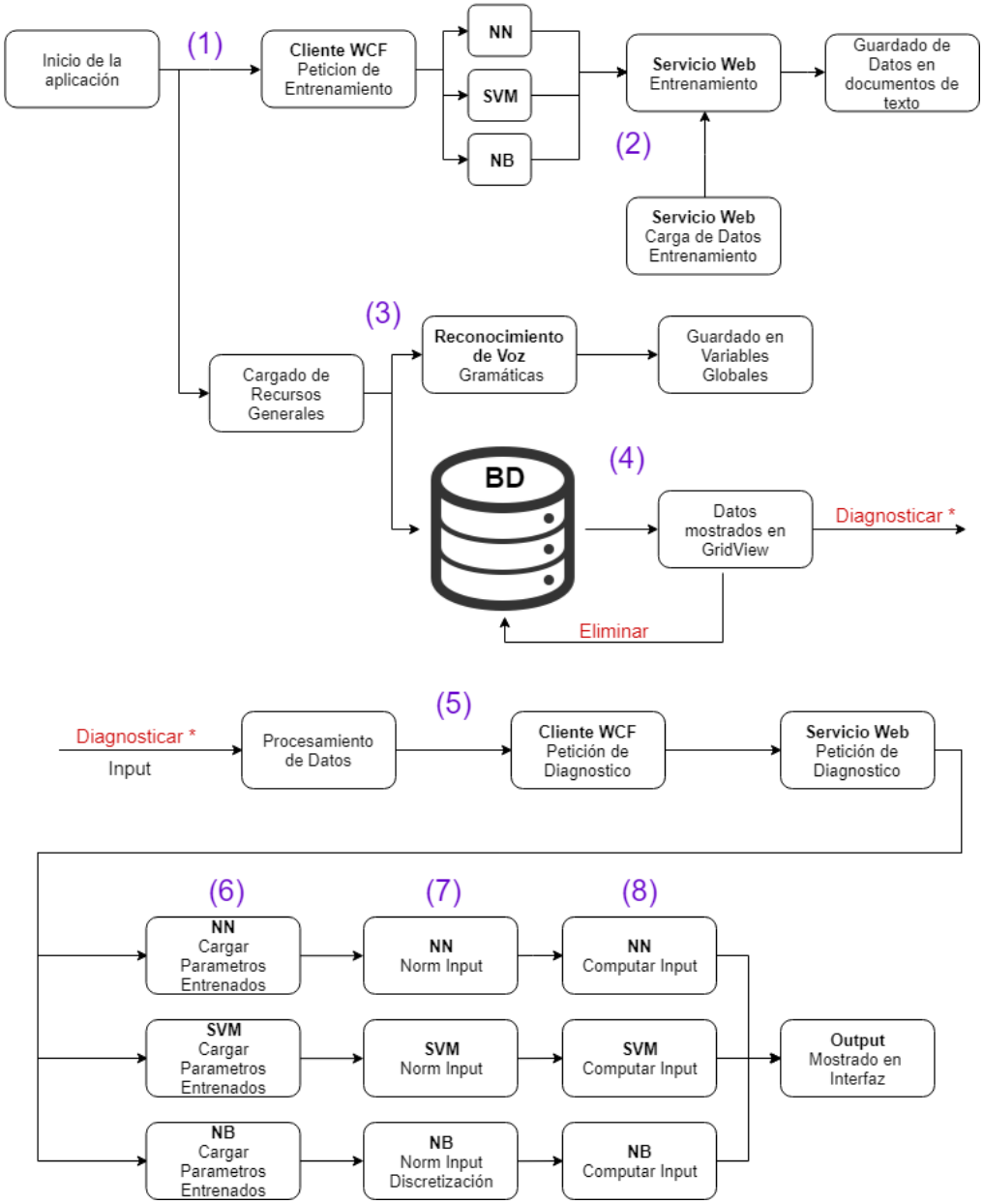


Figura 11: Flujograma de Datos

En el paso (1) del *Flujograma de Datos,* puede verse que al ejecutar la aplicación se hacen dos acciones, estas dos acciones ocurren de forma paralela.

En primer lugar, se hace una petición al *Servicio WCF* [2] *(Windows Communication Foundation)* a través de la clase *Cliente WCF*. Esta petición solicita al servicio web que inicie el entrenamiento de los tres módulos de predicción. El servicio web o *Servicio WCF* está ejecutado y esperando instrucciones hospedado localmente. La función de la clase *Cliente WCF* es básicamente servir de intercomunicador entre el servicio web y la aplicación general.

En el paso (2) del *Flujograma de Datos,* el *Servicio WCF* recibe la petición de entrenar los tres módulos de predicción disponibles. Para hacer esta labor necesita cargar la base de datos de entrenamiento, la cual se obtiene de un archivo de texto. Una vez programados los modelos de análisis utilizando los conocimientos explicados en el apartado *Marco Teórico*, tanto para su evaluación como para su uso en este *Flujo de Datos* surgió cierto problema. El problema era la modificación de la base de datos de entrenamiento para que los módulos de análisis fueran capaces de utilizarla con eficacia. La solución fue utilizar aplicaciones, como "note++" de lectura y edición de texto. Junto con estas aplicaciones, sus herramientas de búsqueda y expresiones regulares se consiguió abordar el problema. Haciendo cambios como "M" de maligno y "B" de benigno, de la base de datos original, por "1" para maligno y "0" para benigno, entre otros.

La base de datos de entrenamiento utilizada se ha descargado de la siguiente URL *"https://www.kaggle.com/uciml/breast-cancer-wisconsin-data/downloads/data.csv/2"*. Está compuesta por 32 columnas de las cuales treinta son atributos, las otras dos son los Identificadores (ID) de los pacientes y el diagnóstico maligno/benigno. Además, contiene 569 filas representando a los pacientes. Esta base de datos ha sido utilizada por diversos trabajos[3]. Podemos ver una pequeña muestra de cómo es la base de datos en la *Tabla 5*, en esta muestra hay tan solo un registro con sus 32 columnas, incluidas el *ID* del paciente y *"diagnosis"* que representa el diagnostico con los valores "malign" (M) y "benign" (B).

[2] Un Servicio WCF es un marco para crear aplicaciones orientadas a servicios. Con WCF, puede enviar datos como mensajes asincrónicos de un extremo de servicio a otro.

[3] https://www.kaggle.com/kanncaa1/feature-selection-and-data-visualization
https://www.kaggle.com/bibhuti93/cancer-detection/data
https://www.kaggle.com/mirichoi0218/classification-breast-cancer-or-not-with-15-ml

id	diagnosis	radius_mean	texture_mean
842302	M	17.99	10.38
perimeter_mean	area_mean	smoothness_mean	compactness_mean
122.8	1001	0.1184	0.2776
concavity_mean	concave_points_mean	symmetry_mean	fractal_dimension_mean
0.3001	0.1471	0.2419	0.07871
radius_se	texture_se	perimeter_se	area_se
1.095	0.9053	8.589	153.4
smoothness_se	compactness_se	concavity_se	concave_points_se
0.006399	0.04904	0.05373	0.01587
symmetry_se	fractal_dimension_se	radius_worst	texture_worst
0.03003	0.006193	25.38	17.33
perimeter_worst	area_worst	smoothness_worst	compactness_worst
184.6	2019	0.1622	0.6656
concavity_worst	concave_points_worst	symmetry_worst	fractal_dimension_worst
0.7119	0.2654	0.4601	0.1189

Tabla 5: Primer Paciente de la Base de Datos

Para el entrenamiento de los módulos de predicción como ya se ha dicho en el apartado *Marco Teórico* era importante normalizar los valores de la base de datos de entrenamiento previamente. Además, para el caso de Naïve Bayes se requeria un paso más, este paso era la discretización de datos para el módulo de Naïve Bayes en un número *"x"* de cuartiles *y* supuso un reto que había que solventar. Se solventó con un método basado en la cantidad de rangos deseada. Este método depende de que los datos hayan sido previamente normalizados. Aunque después de la implantación completa se ha leído en algunos artículos la posibilidad de utilizar Naïve Bayes con un corpus con datos no discreto (Yang & Webb, 2002; Metsis, Androutsopoulos, & Paliouras, 2006; Perez, Larranaga, & Inza, 2006), basado en probabilidad y en una función exponencial que define una campana de Gauss. También sería una posibilidad muy a tener en cuenta para futuras mejoras de la aplicación.

Una vez se han entrenado los módulos de predicción, se guardan en archivos de texto temporales todos los valores de todos los "*parámetros de entrenamiento",* cuyo valor da significado a que cierto modelo de predicción esté entrenado o no. Cada modelo tiene unos parámetros de entrenamiento totalmente diferentes, por lo tanto, se decidió asignar a cada

modelo un espacio reservado dedicado en forma de directorio, donde se guardan los archivos temporales que genere.

El registro de los valores de los parámetros de entrenamiento, hechos por el Servicio Web, en documentos de texto. Fue un reto debido a que, durante este registro de datos, se tuvo que hacer frente a problemas como la creación de métodos para guardar matrices de tres dimensiones en un documento de texto para luego recomponerlas con otro método leyendo el mismo texto que se acababa de crear.

En el paso (3) del *Flujograma de Datos,* que ocurre paralelamente a *(2)*, se puede observar cómo se cargan los archivos necesarios para que la aplicación general funcione. Por una parte, se carga desde archivos de texto diferenciados todo el vocabulario necesario durante el uso de la aplicación para que el sistema de reconocimiento de voz tenga sus "*gramáticas*" completas, y todos los comandos de voz sean reconocibles y funcionales. Por otra parte, se carga la base de datos de pacientes al completo, tanto para mostrarlos en la interfaz interactiva como para manipularlos, como se observa en el paso (4) del *Flujograma de Datos.*

A la hora de reconocer la voz, fue un gran reto entender cómo funcionaban las cargas, descargas y construcción de las gramáticas, la adición de vocabulario nuevo, la limitación de palabras clave que reconocer, establecer criterios de análisis de texto reconocido, o analizar y depurar el texto reconocido, entre otras tareas. Y el último gran reto que planteó el reconocimiento de voz de Visual Studio fue la interpretación del texto reconocido de una forma u otra, utilizando unas gramáticas u otras según el contexto de la aplicación en un determinado momento. Una de las soluciones fue establecer variables de contexto en toda la aplicación para poder depurar el texto condicionando esa depuración por el valor de las variables contextuales. La solución para la interferencia de las gramáticas generales y las de introducir un paciente mediante voz fue programar métodos de descarga de gramáticas, fruto de una comprensión profunda de cómo estas eran cargadas. Finalmente, la detección de números por voz se solucionó parcialmente utilizando unas gramáticas con mucha carga de vocabulario y unos diccionarios que interpretaban el número escrito y lo transformaban en el valor pertinente. Fue una solución parcial debido a que solo se pueden detectar por voz números que estén cierto rango (del 0 al 999.999,999.999).

En el paso (4) del *Flujograma de Datos,* ya se han cargado los pacientes de la *base de datos* y se procede a mostrarlos por pantalla. Una vez mostrados por pantalla, el usuario

tiene la posibilidad de manipularlos de distintas formas. Una forma es "Eliminar" individual y completamente de la base de datos, otra forma de actuar sobre ellos es iniciar el proceso de "Diagnosticar *". Cuando se habla de diagnosticarlos, se hace referencia a iniciar el proceso que empieza por procesar el paciente en cuestión para posteriormente enviarlo al paso *(5)* del flujograma de datos. Este procesado es básicamente la transformación de los datos que pueden verse en la interfaz, de cadenas de texto a un objeto *Paciente,* el cual puede entender y manipular libremente tanto el servicio web como la aplicación general.

En el paso (5) del *Flujograma de Datos,* el paciente ya está transformado y listo para iniciar el proceso de diagnóstico. En primer lugar, el *Paciente* se envía al *Cliente WCF* para que haga la petición de diagnóstico al *Servicio WCF*. Una vez el *Servicio WCF* ha procesado la petición de diagnóstico a través de todos los módulos de predicción, se inicia el proceso de diagnóstico de cada uno de los modelos para finalmente obtener tres predicciones "diferentes", aunque en realidad en el código existe la posibilidad de únicamente invocar tanto el entrenamiento como el diagnóstico de cada uno de los módulos por separado.

En el paso (6) del *Flujograma de Datos,* cada uno de los módulos inicia el proceso de carga de datos necesarios, o "*parámetros de entrenamiento*" para la predicción, almacenados en archivos de texto. Estos datos ya han sido generados previamente al inicio del programa en *Flujograma de Datos (2)*. [4]

En el paso (7) del *Flujograma de Datos,* se inicia la normalización del paciente introducido teniendo en cuanta cómo se había normalizado previamente la base de datos de entrenamiento. Para esta normalización utiliza la *fórmula (1)* y básicamente serán necesarios *min* y *max*, teniendo en cuenta todos los valores de todos los atributos del corpus por separado, para cada uno de los atributos del paciente. Las variables *min* y *max* han sido guardadas para cada uno de los atributos en *(2)* y ahora se cargan para ser utilizados en la normalización del nuevo paciente. Como acción adicional en *Flujograma de Datos (7)* se discretizan los datos ya normalizados en el módulo de predicción ***NB***, en el caso de esta aplicación se discretiza utilizando dos divisiones del rango original [0,1] con la misma filosofía utilizada en la *Tabla 2.*

[4] Tanto los módulos de predicción, como sus métodos para entrenarlos y los métodos para hacer un diagnóstico de un paciente nuevo están alojados en el Servicio WCF.

Finalmente, en el paso (8) *Flujograma de Datos* se utilizan los *parámetros de entrenamiento* cargados en *Flujograma de Datos (6)*. Con estos parámetros ya cargados en el módulo de análisis pertinente, se computa el paciente introducido y se envía el resultado al *Cliente WCF* para que la aplicación general pueda mostrárselo al usuario a través de la interfaz.

Con este último paso del flujograma de datos se llegó también al final de la programación de la aplicación y de todas las funcionalidades deseadas, aunque se describen con mayor profundidad algunas de las etapas de desarrollo en el apartado *Interfaz de la Aplicación*. Cuando la aplicación estuvo completada, se prosiguió a crear un instalador sencillo de la aplicación que funcionara en cualquier ordenador, y que no requiriera por parte del usuario ningún conocimiento en programación, dominio de directorios, bases de datos o alojamiento de Servicios WCF. Cada uno de los conocimientos que se intentaba evitar al usuario final era un pequeño calvario a la hora de la creación del instalador. Finalmente, se decidió abordar la solución con la aplicación *"Advanced Installer 16.0"*, programa que puede descargarse desde que se puede descargar desde ***"***https://www.advancedinstaller.com/download.html***"***, y se modificó la aplicación final para que lo que hacía el *Servicio WCF* estuviera integrado en un directorio ajeno, en lugar de en un alojamiento local respecto a la aplicación general, sin perder la metodología y filosofía de un Servicio Web ni la nomenclatura original de las clases.

4. Evaluación

Para la evaluación de los módulos de predicción, se han realizado tres aplicaciones consola a parte de la aplicación principal, una por cada módulo. Los métodos utilizados para la evaluación de los módulos de predicción se basan en el estudio de la *validación cruzada*, que ha sido explicado en el subapartado de *Evaluación* del apartado *Marco Teórico*. Para esta evaluación, se ha utilizado obviamente el mismo corpus y el mismo código de los módulos alojados en el *Servicio WCF*. Una vez ejecutado el programa, se pueden ver los siguientes resultados:

(A) Para Redes Neuronales:

```
------------------------------------------------------------
Final measures with test data, means of measures of all k = 3 iterations:
 - ACCURACY = 0,9543
 - PRECISION = 0,9905
 - RECALL = 0,8885
 - F1 = 0,9367
------------------------------------------------------------
------------------------------------------------------------
Final measures with test data, means of measures of all k = 5 iterations:
  - ACCURACY = 0,9702
 - PRECISION = 0,9628
 - RECALL = 0,9589
 - F1 = 0,9607
------------------------------------------------------------
------------------------------------------------------------
Final measures with test data, means of measures of all k = 10 iterations:
 - ACCURACY = 0,9724
 - PRECISION = 0,9688
 - RECALL = 0,9553
 - F1 = 0,9614
------------------------------------------------------------
```

(B) Para SVM:

```
------------------------------------------------------------
Final measures with test data, means of measures of all k = 3 iterations:
 - ACCURACY = 0,9772
 - PRECISION = 0,9859
 - RECALL = 0,9540
 - F1 = 0,9696
------------------------------------------------------------
------------------------------------------------------------
Final measures with test data, means of measures of all k = 5 iterations:
 - ACCURACY = 0,9807
 - PRECISION = 0,9903
 - RECALL = 0,9589
 - F1 = 0,9743
------------------------------------------------------------
```

\- -
Final measures with test data, means of measures of all k = 10 iterations:
 - ACCURACY = 0,9829
 - PRECISION = 0,9904
 - RECALL = 0,9625
 - F1 = 0,9760
\- -

(C) Para Naïve Bayes:

\- -
Final measures with test data, means of measures of all k = 3 iterations:
 - ACCURACY = 0,8717
 - PRECISION = 0,9595
 - RECALL = 0,6911
 - F1 = 0,8024
\- -
\- -
Final measures with test data, means of measures of all k = 5 iterations:
 - ACCURACY = 0,8768
 - PRECISION = 0,9597
 - RECALL = 0,7037
 - F1 = 0,8111
\- -
\- -
Final measures with test data, means of measures of all k = 10 iterations:
 - ACCURACY = 0,8834
 - PRECISION = 0,9606
 - RECALL = 0,7195
 - F1 = 0,8221
\- -

5. Interfaz de la aplicación

En la *Figura 12* se puede ver un el flujo de ventanas que se experimenta durante el uso de la aplicación. Este flujograma se hizo antes de empezar a programar la aplicación y se ha intentado ser estricto a la hora de seguir el plan de implementación inicial que representa la *Figura 12*.

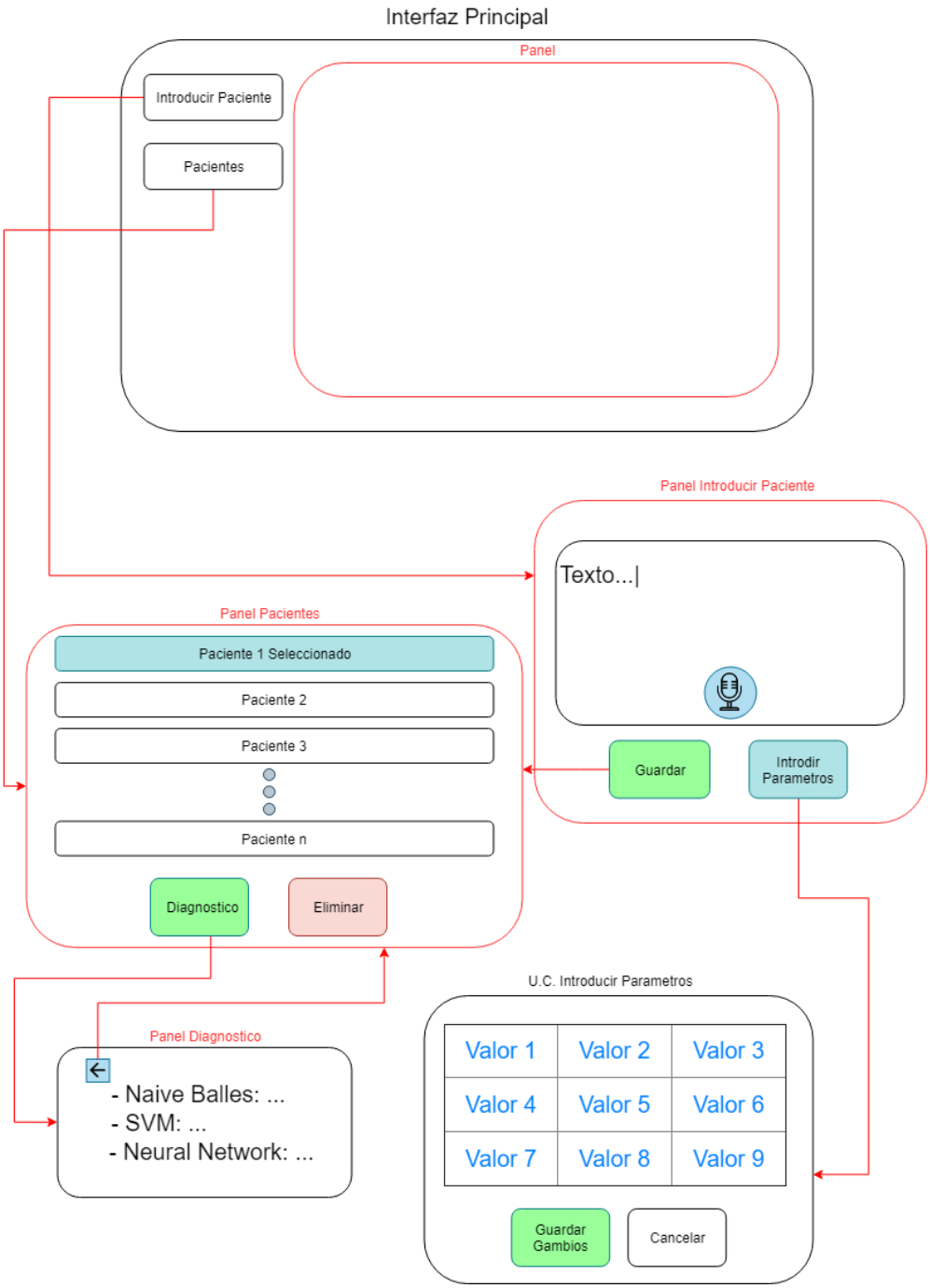


Figura 12: Flujograma de la Aplicación

Para diferentes partes este trabajo se ha intentado ser lo más estricto posible siguiendo los *objetivos*. Todo se ha programado en el lenguaje C#, y en el entorno de desarrollo *"Visual Studio"*. Se han utilizado libros y webs para extrapolar el código de los modelos de predicción, además de la realización de un reconocimiento de voz funcional, *"Servicio Web"* y sincronización con Base de Datos (BD).

En primer lugar, se empezó con el esqueleto principal de la aplicación que se puede ver en la *Figura 13*, haciendo referencia a la *Interfaz Principal* de la *Figura 12*. Una vez programado esto se habilitaron dos paneles, uno para poder introducir pacientes y otro para visualizarlos y manipularlos.

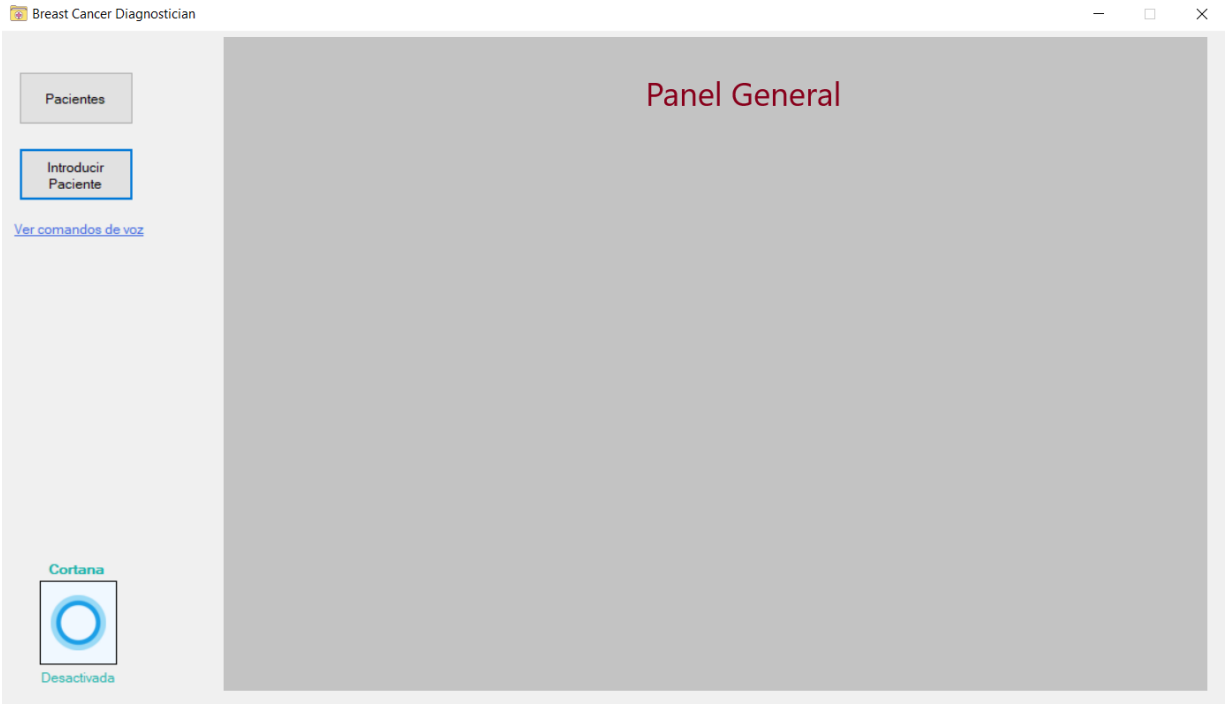


Figura 13: Interfaz Principal

Una vez en el *Panel Introducir Paciente* de la *Figura 12,* del cual se puede ver una captura en la *Figura 14,* se proponían los siguientes retos. Por un lado, habilitar a la interfaz para poder introducir tantos parámetros como atributos tiene el corpus del trabajo. Por otro lado, ser capaz de introducir estos datos vía voz. El poder filtrar tal cantidad tanto de datos como atributos mediante la voz exigía por sí mismo solventar un gran problema. Se utilizó una clase auxiliar que representaría al reconocedor de voz, llamada "*SpeechRecognizer*", una clase con muchísima carga datos que gestionar. Para gestionar todos estos datos han sido necesarios muchos métodos, entre los más importantes pueden verse, los cargadores y

administradores de gramáticas, los encargados de descargar estas gramáticas para que no entren en conflicto con otras funcionalidades y los analizadores de texto. La función principal los métodos descargadores de gramáticas es que el reconocimiento de voz general del programa no esté en conflicto con el reconocimiento de atributos y números.

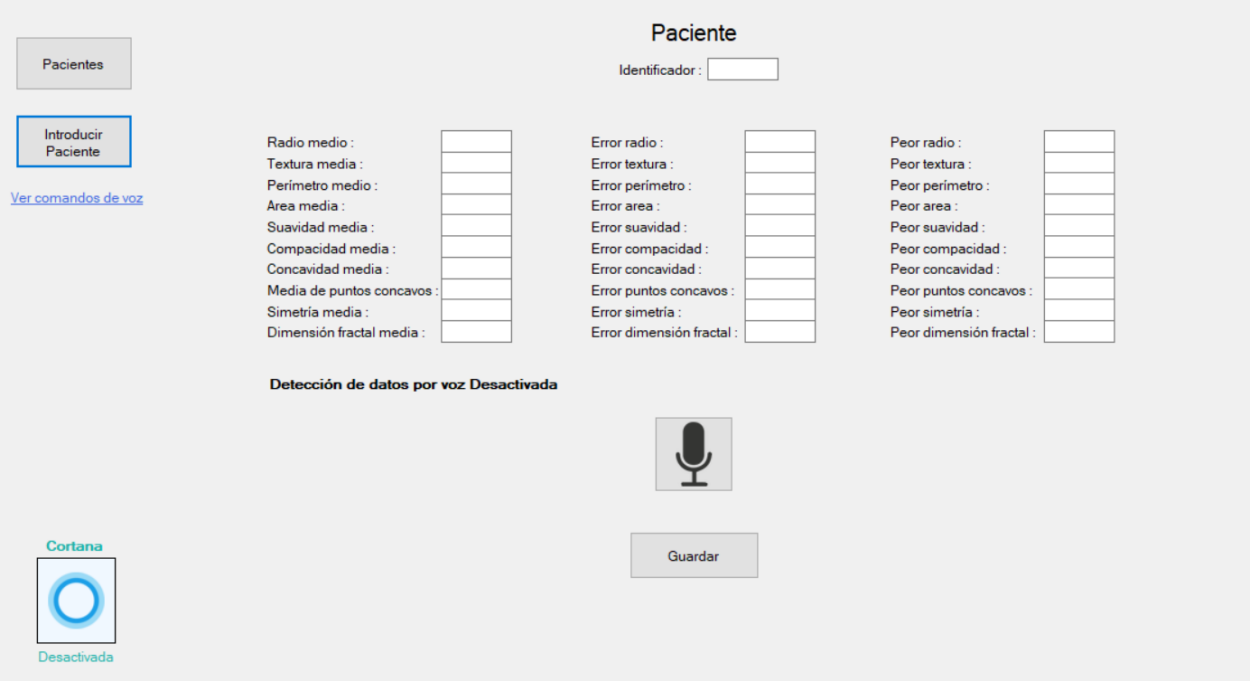
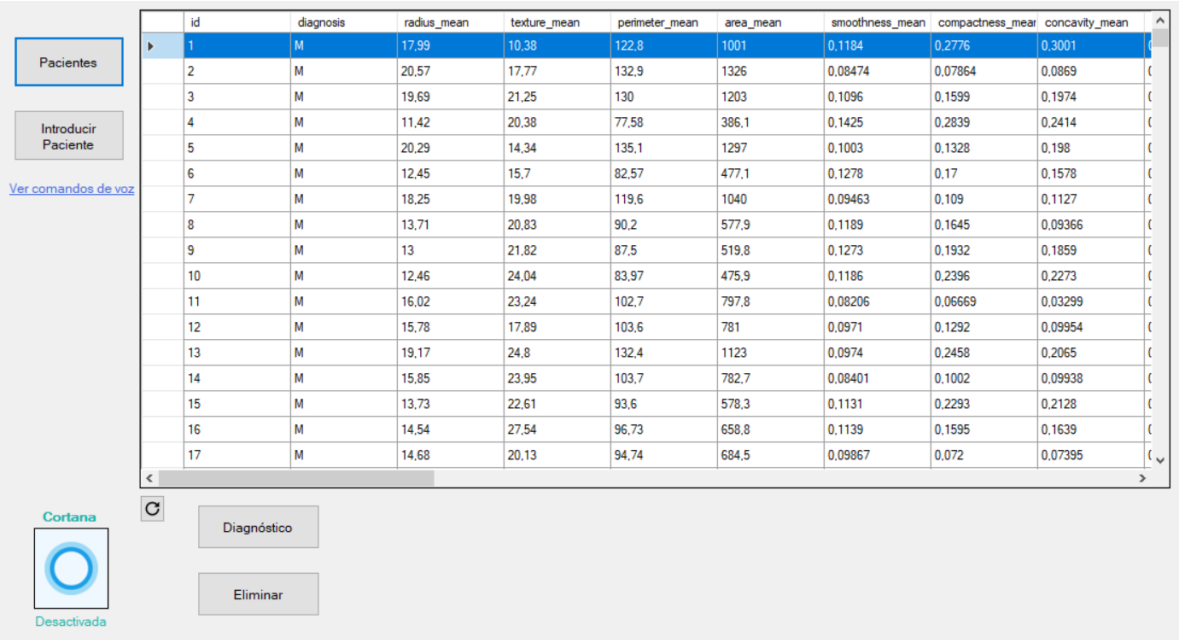


Figura 14: Panel Introducir Paciente

Una vez introducido el paciente es posible guardarlo en la base de datos, donde se le asigna una *ID* automática si no se elige una en concreto. Para introducirlo en la base de datos primero se convierten los datos introducidos en el *Panel Introducir Paciente* en un objeto de la clase *Paciente*. A los objetos *Paciente* se les ha otorgado la capacidad de transformarse a ellos mismos en una cadena de texto procesable por el lenguaje SQL. Esta cadena de texto es básicamente una orden del tipo "*Insert*" con todos los datos que necesita la *BD* para introducir un nuevo paciente. Una vez obtenida la orden tipo *Insert* del *Paciente*, se utiliza la clase llamada "*DataAccess"* para comunicarse con nuestra *BD* e introducir en ella finalmente el Paciente.

Una vez creado el *Panel Introducir Paciente* se procedió con la creación del *Panel Pacientes* del cual puede verse una captura de pantalla en la *Figura 15*, este panel es el encargado de mostrar a los pacientes y dar la posibilidad de interactuar con ellos al usuario. Utilizando un simple "*GridView*", y dando una orden tipo *"Select"* con la clase *DataAcces* se muestran por pantalla todos los pacientes que contiene la *BD.* Cuando ya han sido mostrados existe

la posibilidad de *"Diagnosticarlos"* o de *"Eliminarlos"* como bien se puede observar en los botones disponibles de este panel en la *Figura 15*. La utilidad de *Eliminar* se programó de inmediato haciendo uso de la clase *DataAcces* y la orden SQL tipo *"Delete"*.



id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean
1	M	17,99	10,38	122,8	1001	0,1184	0,2776	0,3001
2	M	20,57	17,77	132,9	1326	0,08474	0,07864	0,0869
3	M	19,69	21,25	130	1203	0,1096	0,1599	0,1974
4	M	11,42	20,38	77,58	386,1	0,1425	0,2839	0,2414
5	M	20,29	14,34	135,1	1297	0,1003	0,1328	0,198
6	M	12,45	15,7	82,57	477,1	0,1278	0,17	0,1578
7	M	18,25	19,98	119,6	1040	0,09463	0,109	0,1127
8	M	13,71	20,83	90,2	577,9	0,1189	0,1645	0,09366
9	M	13	21,82	87,5	519,8	0,1273	0,1932	0,1859
10	M	12,46	24,04	83,97	475,9	0,1186	0,2396	0,2273
11	M	16,02	23,24	102,7	797,8	0,08206	0,06669	0,03299
12	M	15,78	17,89	103,6	781	0,0971	0,1292	0,09954
13	M	19,17	24,8	132,4	1123	0,0974	0,2458	0,2065
14	M	15,85	23,95	103,7	782,7	0,08401	0,1002	0,09938
15	M	13,73	22,61	93,6	578,3	0,1131	0,2293	0,2128
16	M	14,54	27,54	96,73	658,8	0,1139	0,1595	0,1639
17	M	14,68	20,13	94,74	684,5	0,09867	0,072	0,07395

Figura 15: Panel Pacientes

Cuando estos paneles estuvieron desarrollados se procedió con el desarrollo de la funcionalidad más compleja, *Diagnosticar*. Para desarrollar esta funcionalidad se eligió hacerlo a través de un *Servicio Web*, dando la posibilidad de alojar los módulos de predicción de ayuda para el diagnóstico de cáncer de mama en la red, y ser utilizados a su vez por otras aplicaciones. Para el desarrollo de un *Servicio Web* es necesario crear otro proyecto en *Visual Studio* del tipo "*WCF Service*". En la *Figura16* se puede ver una captura de lo que se obtiene cuando es diagnosticado alguno de los pacientes.

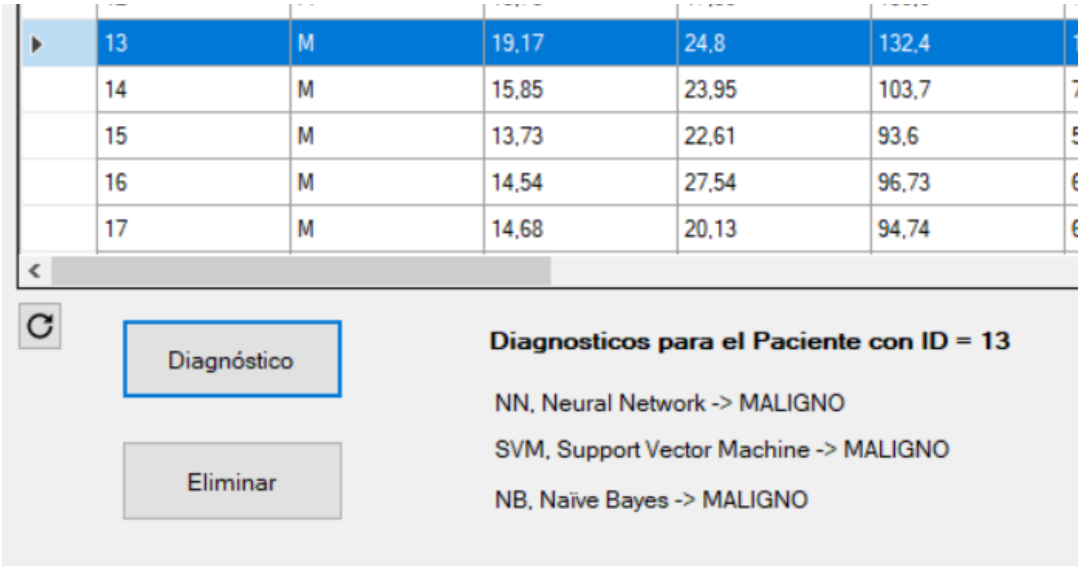


Figura 16: Demostración de Diagnostico

En este *Servicio Web* se implementó una clase tipo interfaz con métodos de entrenamiento y diagnóstico. También se implementó un objeto equivalente a *Paciente* de la aplicación principal. Finalmente, en la clase principal del *Servicio Web* se implementó la clase interfaz y con ello sus métodos, para estos métodos, tanto de entrenamiento como de diagnóstico, era necesario programar los módulos de predicción.

Para programar los módulos se crearon primero tres aplicaciones de consola donde poder trabajar con ellos con libertad y evaluarlos como ya se ha visto anteriormente. Para la programación de estos módulos se hizo uso de diversos libros y la extrapolación de los ejemplos que contienen, tales libros están agregados en la bibliografía y citados en el marco teórico. Los métodos de entrenamiento requieren de acceso a librerías, para crear y leer archivos temporales con los datos entrenados y de entrenamiento. Con la creación del *Servicio Web* fue imprescindible crear una clase Cliente en el proyecto principal para comunicarse con ese servicio, tanto para hacer las peticiones de entrenamiento como para hacer las de diagnóstico.

Finalmente se implementó el reconocimiento de voz general de la aplicación. Fue evidente la creación de un apartado en la aplicación para visualizar todos los comandos de voz disponibles en la aplicación. Poco a poco se fueron agregando todas las funcionalidades al reconocimiento de voz, siendo un problema evidente los conflictos con el reconocimiento de voz que ya existía para la introducción del paciente. Por ello, se decidió separar el reconocimiento de acciones y el de datos del paciente. En la *Figura 17* se pueden ver todas

las funcionalidades disponibles en lo que a reconocimiento de voz se refiere. Debido al gran número de comandos disponibles se han guardado en un documento de texto que se administra con la clase *"DataResources"*, no siendo tanto por las acciones sino por el reconocimiento de números.

Figura 17: Comandos de Voz

6. Conclusiones

La conclusión general sobre el trabajo, es que el futuro de la inteligencia artificial es incierto pero muy prometedor. Con conocimientos fundamentales sobre los módulos de análisis y predicción, se han conseguido grandes resultados de precisión en las predicciones, sin importar cuál de los módulos de análisis se observe.

El módulo de clasificación Bayesiano ha sido con diferencia el más arcaico, limitado e impreciso, pero no todo son malas impresiones sobre este módulo de análisis, dado que muchos estudios respaldan que la clasificación Bayesiana es capaz de conseguir bastantes buenos resultados con una base de datos de entrenamiento pobre (McCallum & Nigam, 1998; Tsangaratos & Ilia, 2016). Además, este módulo es muy rápido a la hora de entrenarse, esto también es debido a su simpleza en las operaciones, ya que es puramente probabilístico, basándose todas sus matemáticas en modelos euclidianos, ya sea en una o varias dimensiones.

Sobre redes neuronales, se observó durante el trabajo que por muchas razones es el más prometedor de los tres. Es el más rápido, y fácilmente interpretable cuando se acompaña la explicación de un modelo de *"red neuronal real"*, incluso de una neurona en particular. Su forma de hacer los cálculos y el entrenamiento es elástica, fluida y relativamente fácil de entender. También es el modelo más preciso y más eficiente, tanto a la hora de clasificar objetos nuevos como de entrenarse a sí mismo. No obstante, también es cierto que es muy dependiente de una base de datos excelente para realizar un buen trabajo.

Con respecto a SVM, se puede decir que ha sido el más complejo e ineficiente en la relación de tiempo de cálculo y precisión. Ha sido con diferencia el más difícil de comprender. De hecho, para comprender por completo SVM y entender todos y cada uno de sus entresijos se requeriría un trabajo como este dedicado en su totalidad a este modelo. Esta complejidad se debe a la forma tan intrincada y antinatural de deducir los *vectores de soporte*. Además, es bastante dependiente de la calidad del corpus y es muy lento en los cálculos. Como punto positivo, es muy preciso y no tan dependiente de la calidad del corpus como el modelo de predicción basado en redes neuronales.

Como conclusión sobre la aplicación final, se ha observado que el reconocimiento de voz podría resultar muy útil para muchos trabajos en general, pero más incluso para un quirófano, donde el cirujano podría básicamente dictar los datos que va midiendo del tumor mientras la aplicación lo registra e interpreta todo. En unos instantes, tal vez incluso sin necesidad de cerrar de nuevo al paciente, este podría ser intervenido si se llegara a visualizar una alta tasa de éxitos en las predicciones de la aplicación durante un uso prolongado en el tiempo.

7. Bibliografía

AbdulHussien, A. A. (2017). Comparison of machine learning algorithms to classify web pages. *International Journal of Advanced Computer Science and Applications*, 5.

Bar, Y. D. (2015). Deep learning with non-medical training used for chest pathology identification. (I. S. Photonics, Ed.) 7.

Bar, Y., Diamant, I., Wolf, L., Lieberman, S., Konen, E., & Greenspan, H. (2015). Chest pathology detection using deep learning with non-medical training. 4.

Bramer, M. (2007). *Principles of data mining* (Vol. 180). London: Springer.

Chris, A. (25 de Diciembre de 2017). *From Perceptron to Deep Neural Nets*. Obtenido de https://becominghuman.ai/from-perceptron-to-deep-neural-nets-504b8ff616e

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems,* , 12.

Ephraim, Y., & Malah, D. (1984). Speech enhancement using a minimum-mean square error short-time spectral amplitude estimator. *IEEE Transactions on acoustics, speech, and signal processing*, 4.

Fisher, R. A., & Yates, F. (1948). *Statistical tables for biological, agricultural and medical research* (6 ed.). Londres: ISBN. Obtenido de https://web.archive.org/web/20110928045150/http://digital.library.adelaide.edu.au/coll/special//fisher/stat_tab.pdf

García Gutiérrez, Á. (2016). Machine learning en bases de datos de lenguaje natural. 54.

Gebru, T., Krause, J., Wang, Y., Chen, D., Deng, J., Aiden, E. L., & Fei-Fei, L. (2017). Using deep learning and google street view to estimate the demographic makeup of the us. *arXiv preprint arXiv: 1702.06683*, 41.

Hagan, M. T. (1994). Training feedforward networks with the Marquardt algorithm. *IEEE transactions on Neural Networks*(989-993), 5.

Hinton, G., Deng, L., Yu, D., Dahl, G., Mohamed, A. R., Jaitly, N., . . . Sainath, T. (2012). Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal processing magazine*, 29. Obtenido de https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/HintonDengYuEtAl-SPM2012.pdf

Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. *14*(2), 7.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *nature*, 10. Obtenido de https://creativecoding.soe.ucsc.edu/courses/cs523/slides/week3/DeepLearning_LeCun.pdf

Lee Koo, C., Jing Liew, M., Mohamad, M., & Hakim Mohamed Salleh, A. (2013). A Review for Detecting Gene-Gene Interactions Using Machine Learning Methods in Genetic Epidemiology. *BioMed research international*(432375), 14. doi:10.1155/2013/432375

Lin, H., Jia, J., Guo, Q., Xue, Y., Li, Q., Huang, J., . . . Feng, L. (2014). User-level psychological stress detection from social media using deep neural network. *Proceedings of the 22nd ACM international conference on Multimedia*, 10.

Litjens, G., Kooi, T., Bejnordi, B. E., Setio, A. A., Ciompi, F., Ghafoorian, M., . . . Sánchez, C. I. (2017). A survey on deep learning in medical image analysis. 38.

López, J. M., & Roca, S. C. (2006). La tradautomaticidad: un concepto aplicado a la evaluación de sistemas de traducción automática. *Procesamiento del lenguaje natural*, 37. Obtenido de http://sinai.ujaen.es/sepln/ojs/ojs/index.php/pln/article/download/2754/1272

McCaffrey, J. (2014). *Neural Networks using C# Succinctly.* Morrisville,: Syncfusion Inc.

McCaffrey, J. (Marzo de 2019). *Support Vector Machines Using C#*. Obtenido de https://msdn.microsoft.com/en-us/magazine/mt833291.aspx?f=255&MSPPError=-2147217396

McCallum, A., & Nigam, K. (July de 1998). A comparison of event models for naive bayes text classification. *AAAI-98 workshop on learning for text categorization, 752*(1), 8.

Metsis, V., Androutsopoulos, I., & Paliouras, G. (July de 2006). Spam filtering with naive bayes-which naive bayes? *CEAS, 17*, 9.

Montiel, J. P. (2018). Detección y corrección de errores basados en reglas gramaticales del inglés en conjunto con el mejoramiento de estilos de escritura, aplicado en artículos científicos mediante Algoritmos de Procesamiento de Lenguaje Natural. 58. Obtenido de http://opac.pucv.cl/pucv_txt/txt-8000/UCC8107_01.pdf

Nguyen, D. T., Joty, S., Imran, M., Sajjad, H., & Mitra, P. (2016). Applications of online deep learning for crisis response using social media information. *arXiv preprint arXiv: 1610.01030*, 6.

Perez, A., Larranaga, P., & Inza, I. (2006). Supervised classification with conditional Gaussian networks: Increasing the structure complexity from naive Bayes. *International Journal of Approximate Reasoning*, 25.

Platt, J. C. (21 de Abril de 1998). Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines. *Microsoft Research*, 21. Obtenido de https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/tr-98-14.pdf

Ravì, D., Wong, C., Deligianni, F., Berthelot, M., Andreu-Perez, J., Lo, B., & Yang, G. Z. (2016). Deep learning for health informatics. *IEEE journal of biomedical and health informatics, 21*, 18.

Roberto de Souza, C., & Brigante Pizzolato, E. (2013). Sign Language Recognition with Support Vector Machines and Hidden Conditional Random Fields. 15.

Stamford, J. (13 de Abril de 2015). *Python Perceptron*. Obtenido de http://stamfordresearch.com/python-perceptron-re-visited/

Tsangaratos, P., & Ilia, I. (October de 2016). Comparison of a logistic regression and Naïve Bayes classifier in landslide susceptibility assessments: The influence of models complexity and training dataset size. *145*, 164 -179. Obtenido de Catena.

Xu, Y., Mo, T., Feng, Q., Zhong, P., Lai, M., Eric, I., & Chang, C. (2014). Deep learning of feature representation with multiple instance learning for medical image analysis. *IEEE international conference on acoustics, speech and signal processing (ICASSP)*, 5.

Yang, Y., & Webb, G. I. (July de 2002). Non-disjoint discretization for naive-bayes classifiers. *ICML , 2*(76), pág. 8.

Zeng, Z. Q., Yu, H. B., Xu, H. R., Xie, Y. Q., & Gao, J. (2008). Fast training support vector machines using parallel sequential minimal optimization. *3rd international conference on intelligent system and knowledge engineering* (pág. 25). IEEE.

Zhang, Z., He, Q., Gao, J., & Ni, M. (2018). A deep learning approach for detecting traffic accidents from social media data. *Transportation research part C: emerging technologies*, 30.