Escuela Politécnica Superior

Predicción de la probabilidad de éxito en la adquisición de clientes

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor:
Benjamín Pamies Cartagena

Tutor/es:
Miguel Ángel Lozano Ortega

Septiembre 2017

Universitat d'Alacant
Universidad de Alicante

Predicción de la probabilidad de éxito en la adquisición de clientes

Benjamín Pamies Cartagena

GRADO EN INGENIERÍA INFORMÁTICA

Universitat d'Alacant
Universidad de Alicante

Escuela Politécnica Superior

A todas aquellas personas que creyeron que algún día acabaría este camino.

El razonamiento matemático puede considerarse
más bien esquemáticamente como el ejercicio
de una combinación de dos instalaciones,
que podemos llamar la intuición y el ingenio.

ALAN TURING

Saber mucho no es lo mismo que ser inteligente.
La inteligencia no es sólo información, sino también juicio,
la manera en que se recoge y maneja la información.

CARL SAGAN

Índice de contenidos

Índice de figuras

1. Introducción

El propósito general de toda empresa y la fuente principal de sus ingresos es la venta de sus bienes o servicios. La comercialización de sus productos se basa en la realización de ofertas a potenciales clientes con el objetivo de que éstos los adquieran y se transformen en beneficios. Atraer a un cliente requiere de mucho tiempo y esfuerzo que suponen gastos para la empresa por lo que cada vez que una oferta no es ganada significa perdida de dinero, pues los recursos invertidos en intentar atraer a ese cliente no se han transformado en una venta. Para solucionar este problema habría que analizar con detalle todos los datos posibles de los clientes potenciales, deducir los que van a aceptar las ofertas que se le hagan y centrarse en ellos para no gastar dinero en clientes potenciales que no van a comprar.

Es necesario estudiar miles de datos de posibles clientes, una tarea demasiado ardua para ser realizada por una persona por lo que hay una necesidad de automatizar el proceso de marketing. Las ciencias de la computación y, más concretamente, la inteligencia artificial tiene mucho que aportar en este sentido.

La inteligencia artificial lleva años ayudando al mundo empresarial en la gestión de sus negocios. Desde que Google se sumergió en este mundo, ya no sólo aplicándolo en su buscador o en su traductor, sino también adquiriendo DeepMind, la compañía de inteligencia artificial más importante del mundo y desarrolladora de AlphaGo, la primera máquina de Go en ganar a un jugador profesional, otras empresas la han imitado, como es el caso de Facebook que ha integrado estos algoritmos en el reconocimiento facial en las fotografías que subes a la red social, siendo capaz de reconocer las caras de tus amigos para sugerirte etiquetas.

Pero no sólo las empresas tecnológicas usan modelos de inteligencia artificial para ampliar sus oportunidades de negocio y maximizar sus beneficios. También se usa, por ejemplo, en entidades bancarias para detectar fraudes en transacciones y en el análisis histórico de préstamos para predecir impagos en el futuro, en el sector de la salud para el diagnóstico de enfermedades o el ejemplo más popular en la actualidad, el de la compañía automovilística Tesla y sus automóviles que conducen por ti. Todos estos ejemplos aplican la rama de la inteligencia artificial conocida como *machine learning* o aprendizaje automático.

El aprendizaje automático consiste en desarrollar técnicas que permitan a las máquinas emular la manera en que los humanos aprendemos. Para esto, es necesario el uso de la experiencia. Si sabemos de la existencia de una ciudad es porque alguna vez la visitamos, porque alguien nos habló de ella o porque nosotros leímos sobre ella y si sabemos lo que es un coche es porque hemos visto muchos a lo largo de nuestra vida. La forma en la que aprendemos es mediante la experiencia y este método puede ser igualmente usado por las máquinas para aprender. De esta manera, contando con datos de clientes de los que ya se conoce si han aceptado ofertas comerciales en el pasado, la máquina puede experimentar con ellos y en base a esa experiencia determinar clientes con características similares los cuales son los que, a priori, más probabilidad tienen de aceptar ofertas similares en el futuro.

Existen varios métodos de aprendizaje automático: por refuerzo, supervisados y no supervisados. El aprendizaje por refuerzo es el más parecido al que usamos los humanos, el algoritmo aprende observando el mundo que le rodea, a base de ensayo y error. Cuando la máquina comete un error le damos un refuerzo negativo y cuando acierta un refuerzo positivo. Las acciones que han llevado a la máquina a tener un refuerzo positivo tienden a ser las más repetidas. Este tipo de aprendizaje automático es utilizado, sobre todo, en robótica. Imaginemos un robot que quiere aprender a salir por la puerta de una habitación por sí solo. En este caso los refuerzos negativos

podrían ser las veces que se choca y los positivos las que consigue salir. Al principio del entrenamiento el robot se chocará muchas veces, pero a medida que avance el entrenamiento las acciones que le llevan a salir por la puerta sucederán con más frecuencia y el robot aprenderá a salir. Esta no es la técnica que vamos a utilizar, vamos a centrarnos en las técnicas supervisadas y no supervisadas.

Los métodos de aprendizaje automático supervisados y no supervisados utilizan ejemplos a partir de los cuales aprender. La diferencia entre el aprendizaje supervisado y el no supervisado es que en el supervisado todos los ejemplos están etiquetados con la clase a la que pertenecen y en el no supervisado no hay etiquetas. Por lo tanto, en el aprendizaje no supervisado la máquina tiene que aprender a distinguir las clases a las que pertenece cada ejemplo por sí sola.

En el aprendizaje supervisado lo que se hace es una clasificación o reconocimiento de patrones. Dado un conjunto de ejemplos etiquetados según su clase, llamado conjunto de entrenamiento, todos los ejemplos son clasificados. Después de clasificar todos los ejemplos se calcula un error asociado a la clasificación, cada ejemplo mal clasificado es un error. Se entrena hasta que no se encuentran errores al clasificar los elementos del conjunto de entrenamiento o hasta que el error es mínimo.

En este trabajo, se dispone de un amplio conjunto de datos de clientes potenciales etiquetados en dos posibles clases: fueron convertidos a clientes o no fueron convertidos. Estos son los ejemplos a partir de los cuales nuestros algoritmos van a aprender. En la segunda sección, marco teórico, se van a introducir técnicas de aprendizaje automático tanto supervisadas como no supervisadas, pero al disponer de un conjunto de ejemplos etiquetados, se implementarán algoritmos que sigan la técnica supervisada. Durante el entrenamiento, la máquina clasificará cada ejemplo en una de las dos clases. Las clasificaciones que coincidan con la etiqueta asociadas al ejemplo serán aciertos.

Lo importante llegados a este punto es que la máquina puede ser capaz de detectar las características que le han llevado a clasificar bien a esos clientes y, por el contrario, discriminar las características que, por lo general, le provocan errar en su clasificación. De esta forma, cuando el algoritmo reciba los datos de un nuevo cliente potencial puede, basándose en la experiencia de los entrenamientos previos, predecir si sus características lo pueden llevar a convertirse en un cliente real.

2. Marco teórico

Como paso previo a la realización del trabajo, se estudian varias técnicas, tanto supervisadas como no supervisadas, típicamente utilizadas en el marco teórico o estado del arte actual del aprendizaje automático.

Estas técnicas sirven de base para la investigación de métodos que puedan resolver el problema planteado. De entre ellas, dos serán implementadas, analizadas y comparadas. Son técnicas que intentan predecir resultados en base a los datos que se les pasan. Nuestros datos son las características de los clientes potenciales y la predicción que buscamos es si se van a convertir en clientes o no.

Podemos encontrarnos ante el problema de que algunas, o muchas, de las características de las que dispongamos sean irrelevantes para el problema ante el que nos encontramos. Conviene entonces extraer las características de los clientes potenciales que verdaderamente sean valiosas para la predicción.

2.1. Selección de características

Hay que encontrar un método que nos seleccione de entre el conjunto total de características de los futuribles clientes, un subconjunto de características relevantes que solucionen el problema y para eso están las técnicas de selección de características.

Básicamente, esta técnica consiste en partir del conjunto de características completo del que se dispone e ir quitando aquellas características que, al quitarlas, mejoran el resultado e ir repitiendo el proceso hasta que el resultado se quede estable. O bien, partir de un conjunto de características pequeño e ir añadiendo las características restantes, quedándose con las que mejoren el resultado y descartando las que no lo hagan. Al final de las iteraciones tendremos el conjunto de características que mayor porcentaje de aciertos nos proporcione.

Existen tres enfoques de selección de características:

- **Métodos de filtro**: Se basa en la correlación de las características generales con la variable a predecir. Los métodos de filtrado suprimen las características menos interesantes. Las no suprimidas son utilizadas para clasificar o predecir datos. Sin embargo, los métodos de filtrado tienden a seleccionar características redundantes porque no consideran las relaciones entre ellas.
- **Métodos wrapper**: Evalúan subconjuntos de características que permiten, a diferencia de los enfoques de filtros, detectar las posibles interacciones entre ellas. Tienen el problema de tener un gran coste computacional cuando el conjunto de características es grande.
- **Métodos embebidos**: Combinan las ventajas de los dos métodos anteriores. El algoritmo de aprendizaje selecciona y clasifica las características.

Hay varios aspectos muy importantes para tener en cuenta en las técnicas de selección de características. Que una característica mejore el desempeño al agregarla al subconjunto de características seleccionadas no quiere decir que sea relevante. Para que las características sean relevantes tiene que existir una dependencia entre ellas y las etiquetas de clasificación.

Puede haber dos características que sean redundantes en el sentido de que no aportan información adicional por agregarlas. Además, puede haber características que sean poco útiles por sí solas pero que al ser combinadas con otras aporten una mejora al desempeño del algoritmo, es decir, dos características que aparenten ser poco útiles en un principio ambas pueden ser útiles juntas.

El modelo de selección de características es muy útil pero también plantea problemas, como se ha visto. Es un método que suele usarse de pre-proceso para reducir el conjunto de características a uno más pequeño que pueda ser usado como entrada de datos para otra técnica de clasificación más precisa, como es el de las redes neuronales artificiales.

2.2. Redes neuronales

Las redes neuronales son el método más extendido actualmente para aplicar aprendizaje automático. Una neurona artificial intenta imitar el comportamiento que tiene una neurona biológica, imitando cada una de sus partes. Así, la información que llega a las neuronas biológicas a través de las dendritas de otras neuronas son los datos de entrada de una neurona artificial y las conexiones sinápticas se traducen a *pesos sinápticos* computacionales.

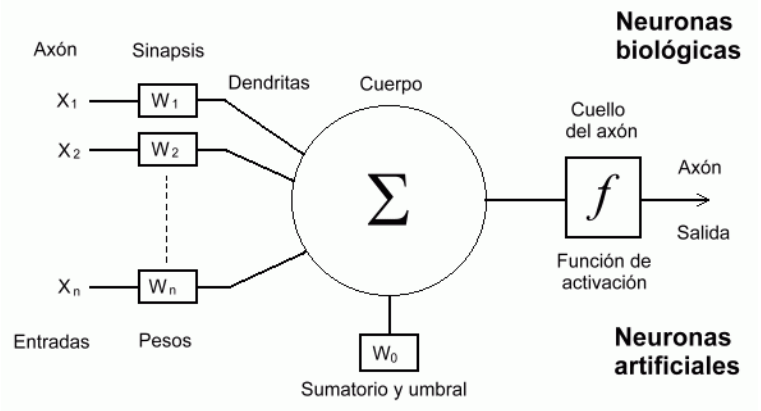

Figura 1: Comparativa entre neurona biológica y artificial

En el cuerpo de la neurona artificial se suman todas las entradas que recibe la neurona multiplicadas cada una por su peso correspondiente:

$$\sum_{i=1}^{n}(w_i \cdot x_i)$$

El resultado de este sumatorio es conocido como *net* o *entrada neta* y es el que recibe la función de activación. Esta función es la que define el dato de salida de la neurona y es importante porque se encarga de evitar la saturación de la neurona limitando el valor que puede llegar a alcanzar ya que si no lo limitásemos se irían produciendo valores de entrada a las neuronas siguientes cada vez mayores, hasta alcanzar valores inmanejables. Hay muchos tipos de funciones que se suelen usar en las redes neuronales artificiales como la función gaussiana o la función sigmoide. Una función simple sería la función escalón para determinar un resultado de salidas binario donde podríamos considerar 0 como estado inactivo de la neurona y 1 cuando la neurona está activada.

$$f(net) = \begin{cases} 1, & w \cdot x - u > 0 \\ 0, & en\ otro\ caso \end{cases}$$

Donde u es el "umbral", el cual representa el grado de inhibición de la neurona, es un término constante que no depende del valor que tome la entrada.

2.3. Backpropagation

La propagación hacia atrás de errores o retropropagación (del inglés backpropagation) es un algoritmo de **aprendizaje supervisado** que se usa para entrenar redes neuronales artificiales.

Este algoritmo se utiliza en redes neuronales multicapa, esto es, cuando hay varias capas de neuronas conectadas entre sí. En este tipo de redes neuronales, existe una primera capa de neuronas que reciben las entradas del conjunto de entrenamiento y una última capa que contienen la salida o las salidas del entrenamiento y entre medias existen varias capas ocultas de varias neuronas cada una que van procesando los datos. En las capas intermedias, los datos de entrada son los datos de salida de la capa anterior y los datos de salida serán los datos de entrada de la capa siguiente.

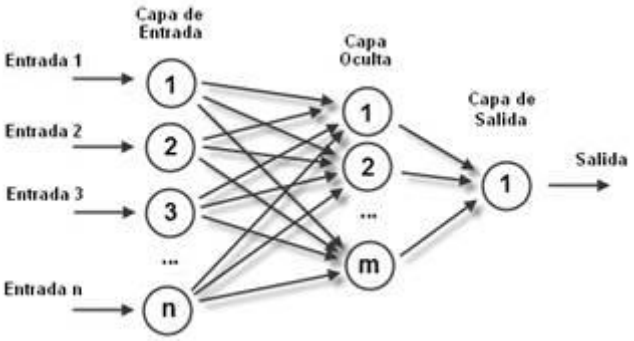

Figura 2: Red neuronal multicapa

Cuando un resultado no se corresponde con el resultado esperado se calcula una tasa de error. Cada neurona de la capa de salida es responsable de una parte de este error. Este error es utilizado para actualizar los pesos de las neuronas. Aquí es donde tiene lugar la propagación hacia atrás, en las redes neuronales con varias capas se vuelve hacía atrás por cada capa para ir calculando la culpabilidad del error que ha tenido cada neurona en todas las capas, desde la última capa hasta la primera, y así actualizar los pesos de todas las neuronas en todas las capas.

Esta técnica es útil ya que podemos ajustar los pesos de las neuronas de tal forma que las neuronas que contribuyen más al error tengan un peso menor mientras que las neuronas que suelen acertar tengan más peso.

Límites de la propagación hacia atrás

Un grave problema de los algoritmos de propagación hacia atrás es que el error se va diluyendo de forma exponencial a medida que atraviesa capas en su camino hasta el principio de la red. Esto es un problema porque en una red muy profunda (con muchas capas ocultas), sólo las últimas capas se entrenan, mientras que las primeras apenas sufren cambios. Por eso

a veces compensa utilizar redes con pocas capas ocultas que contengan muchas neuronas, en lugar de redes con muchas capas ocultas que contengan pocas neuronas (López, 2014).

Esto se soluciona con el **deep learning**.

2.4. Deep learning

Las técnicas de deep learning o aprendizaje profundo permiten tener muchas capas de neuronas artificiales interconectadas entre sí aprendiendo por sí solas. A diferencia de la propagación hacia atrás, donde no tiene mucho sentido tener muchas capas ocultas de neuronas pues las primeras apenas son entrenadas, con deep learning podemos entrenar redes todo lo profunda que queramos, de ahí su nombre.

Existen muchos modelos diferentes de redes neuronales para aplicar deep learning, cada uno de ellos adaptados a una problemática distinta. Vamos a centrarnos en una de las alternativas a la propagación hacia atrás más extendidas en su uso en la actualidad de la inteligencia artificial: las redes recurrentes.

Redes neuronales recurrentes

Mientras que en las redes multicapas utilizadas en backpropagation la información solamente se mueve hacia delante, en las redes recurrentes la información puede moverse en todas direcciones pudiendo formar ciclos, esto es, conectándose tanto con las neuronas de la capa siguiente como con las de la capa anterior e incluso conectándose consigo misma. Son redes con bucles de retroalimentación, que permiten que la información persista. Esto permite a la red tener memoria ya que los números que introducimos en un momento dado en las neuronas de entrada son transformados y continúan circulando por la red incluso después de cambiar los números de entrada por otros diferentes.

Existen dos tipos de redes recurrentes: las parcialmente recurrentes y las totalmente recurrentes. En las primeras no hay conexiones entre todas las neuronas de la red y sus conexiones recurrentes son fijas. Las totalmente recurrentes son aquellas que cada neurona puede estar conectada a cualquier otra y sus conexiones recurrentes son variables.

La complejidad de este tipo de redes es más alta ya que realiza un intercambio de información entre neuronas de una manera mucho más compleja, pero son más eficaces para resolver problemas no lineales[1]. Es una técnica muy utilizada para problemas de predicción, por ejemplo, la predicción de los índices bursátiles basándose en valores anteriores.

Fundamentalmente, la red neuronal recurrente tiene una capacidad más grande de representación de información que la red neuronal hacia delante y es especialmente más apropiada para representar los datos de series en el tiempo tales como señales del habla (Noguera, 2006).

[1] En investigaciones, tales como las de Williams y Zipser (1990) o Mak, M.; Ku, K. y Lu, Y. (1998), se ha hecho énfasis en que, a diferencia de las redes tradicionales, las redes recurrentes tienen comportamiento variable en el tiempo y esto ofrece posibilidades de resolución de problemas diferentes.

Se han propuesto varios modelos de redes recurrentes juntos con sus algoritmos de entrenamiento, como los modelos de Elman (1990), Jordan (1986), Williams y Zipser (1986), Pineda (1987) o de Rumelhart, Hinton y Williams (1986) cuyo algoritmo de aprendizaje es esencialmente el algoritmo de retropropagación.

2.5. Clustering

La técnica clustering o algoritmo de agrupamiento es un procedimiento mediante el cual una serie de características como datos de entrada son agrupados en varios clusters o grupos de acuerdo con un criterio. Esta técnica permite admitir o descartar características dependiendo de los resultados que obtienen los clusters a los que pertenecen durante el entrenamiento de aprendizaje.

Existe dos tipos de clustering o agrupamiento:

- Agrupamiento de partición dura.
- Agrupamiento de partición suave.

En el agrupamiento de **partición dura** cada característica o dato de entrada pertenece exclusivamente a un cluster. Además, los clusters deben cubrir totalmente el conjunto de datos, es decir cada dato tiene que pertenecer a alguno de los clusters.

En el agrupamiento de **partición suave** a cada característica o dato de entrada se le asigna un valor de pertenencia dentro de cada cluster por lo que un dato puede pertenecer parcialmente a más de un cluster. Esto es así porque no siempre es fácil clasificar un dato de entrada en uno de los clusters dado que dicho dato puede contener características pertenecientes a clusters distintos y encontrarse lo suficientemente cerca de dos clusters en el entrenamiento de aprendizaje.

Esta técnica es muy utilizada en minería de datos con el fin de encontrar similitudes entre datos complejos de entre una gran cantidad de datos.

Se trata de una técnica **no supervisada**.

Técnica clustering utilizando redes neuronales: Mapas auto-organizativos

Los mapas auto-organizativos, a partir de ahora SOMs, tienen por objetivo que neuronas asociadas a características similares sean vecinas. Es decir, el objetivo es tener un mapa ordenado de características.

Es el propio SOM el que, partiendo de una población de ejemplos (un conjunto de vector de características), debe finalmente agruparlos (realizar un clustering) utilizando como única información las diferencias y similitudes entre ellos.

Durante el aprendizaje del SOM se irá modificando la topología de la red neuronal hasta que esta se adapte a la estructura desconocida de los ejemplos de entrada. En cada iteración, se determinará una neurona "ganadora" y el resto de neuronas modificarán los pesos de sus características dependiendo de lo cerca que se hallen de la neurona ganadora. Neuronas más cercanas aplicarán más modificaciones y las más lejanas no serán modificadas en absoluto. Así, las neuronas que caen dentro de la vecindad se orientan hacia la unidad ganadora.

En la primera iteración, los pesos asignados se dan aleatoriamente. Estos pesos se van modificando atendiendo a los resultados que se obtienen aplicando los pasos descritos anteriormente.

Otra técnica: K-Means

El algoritmo k-means o k-medias es uno de los más famosos actualmente en la técnica del clustering. Consiste en agrupar en *k* clusters los *n* valores de datos que tenemos. El nombre viene porque representa cada uno de los clusters por la media (o media ponderada) de sus puntos, es decir, por su centroide[2].

Dado un conjunto de objetos,

$$Dn = (x_1, x_2, \dots, x_n)$$

El algoritmo tiene cuatro pasos:

1. Elegir aleatoriamente *k* objetos que forman los *k* clusters iniciales siendo x_i el valor inicial del centro de cada cluster al que pertenece.
2. Reasignar los objetos del cluster. Para cada objeto *x*, el prototipo que se le asigna es el que es más próximo al objeto, según una medida de distancia (habitualmente la medida euclidiana).
3. Una vez que todos los objetos son colocados, recalcular los centros de cada cluster.
4. Repetir los pasos 2 y 3 hasta que no se hagan más reasignaciones.

[2] En geometría, el centroide es el centro de simetría. En un conjunto de puntos el centroide sería un punto que representaría la media del conjunto.

3. Objetivos

Este trabajo tiene como finalidad principal tratar algunas de las técnicas introducidas y estudiadas en el marco teórico y comprobar cómo pueden estas técnicas ayudar al sector empresarial a predecir la probabilidad de éxito en la adquisición de nuevos clientes.

La idea es ver si los datos con los que contamos hacen posible hacer esa predicción. Por tanto, se trata de un trabajo de investigación cuya hipótesis inicial es que a partir del gran conjunto de datos de los que se dispone podemos predecir el éxito que se pueda tener en el futuro frente a un cliente potencial.

Se va a probar con distintos algoritmos, primero que se pueda confirmar esa hipótesis inicial y segundo comparar y ver cuál puede funcionar mejor.

En concreto, se van a implementar dos métodos: uno más sencillo, de tipo voraz, de selección de características y otro, una técnica de deep learning, que requiere más datos, más procesamiento y más recursos pero que es más prometedor.

Se realizará una comparativa entre ambos métodos y se concluirá cuál de ellos da unos resultados más aceptables ante la problemática propuesta, si esos resultados se pueden considerar aplicables ante un escenario real y detectar qué datos son los más adecuados para abordar el problema. Es capital, para poder concluir si alguno de los métodos es una solución, que nos proporcione información relevante para confirmar la hipótesis inicial.

En resumen, el objetivo es ver si somos capaces de extraer esa información relevante para predecir esa salida de éxito o no de si podemos obtener a un cliente.

4. Metodología

Para la implementación de los algoritmos se ha utilizado el lenguaje de programación Python. Es, sin lugar a duda, el lenguaje más utilizado en la investigación científica ya que cuenta con una extensa variedad de librerías de este campo.

Para poner a prueba los algoritmos cuento con una base de datos que servirá de entrenamiento y test a los algoritmos implementados con alrededor de veinte mil casos, que nos permite aplicar deep learning. El sistema de gestión de base de datos utilizado es PostgreSQL por su fácil integración con Python utilizando **psycopg2**.

He hecho uso del paquete de código abierto Numpy. Se trata de una extensión de Python fundamental para trabajos científicos realizados con este lenguaje de programación por la extensa biblioteca de funciones que aporta para la investigación científica. Entre otras cosas, el paquete Numpy contiene:

- Un potente array de objetos N-dimensional.
- Sofisticadas funciones.
- Herramientas para la integración de código C/C++ y Fortran.
- Álgebra lineal, transformada de Fourier y una gran capacidad para generar números aleatorios.

Como biblioteca de apoyo he usado la más famosa para aplicar aprendizaje automático desde que Google la publicó bajo licencia de código abierto en noviembre de 2015: TensorFlow.

4.1. TensorFlow

Figura 3: Logo de TensorFlow

TensorFlow es una biblioteca de código abierto para el aprendizaje automático creada originalmente para Python y extendida posteriormente a otros lenguajes de programación como C, Java o Go. Está desarrollada por Google y permite aplicar deep learning y otras técnicas de machine learning de una forma bastante potente.

Esta librería utiliza diagramas de flujo de datos para realizar sus cálculos numéricos siendo los nodos del grafo las operaciones matemáticas de la red neuronal mientras que los arcos representan las relaciones de entrada y salida entre los nodos y sus pesos correspondientes.

Su nombre procede de los tensores. Un tensor es la unidad central de datos en TensorFlow y consiste en un conjunto de valores primitivos formados en una matriz de cualquier dimensión. El rango de un tensor es su número de dimensiones.

Muchas empresas importantes usan esta librería para sus aplicaciones. Un ejemplo es el gigante de los seguros AXA, que utiliza esta biblioteca para predecir (con un 78% de acierto) las posibilidades de que un asegurado cause un grave accidente de coche[3].

4.2. Implementaciones

Con el uso de las herramientas y librerías mencionadas anteriormente he realizado dos implementaciones, cada una de ellas de una técnica de aprendizaje diferente.

Como los datos de los que dispongo están etiquetados y con la intención de aprovechar esto, las dos implementaciones que he realizado siguen el modelo supervisado. Una de las implementaciones es un método sencillo, de tipo voraz, de selección de características y la otra es una implementación más prometedora del estado del arte, usando redes neuronales artificiales, para realizar una comparativa entre ambas.

La técnica de selección de características devuelve un subconjunto de características óptimo de entre el total y la red neuronal, técnica muy utilizada en problemas de predicción, sirve como clasificador.

Podría aprovecharse el subconjunto de características obtenido en el primer algoritmo de selección de características como el vector de entradas de la red neuronal, pero se ha dejado que la red entrene con el conjunto total de características con los que se dispone. Esto se ha hecho para evitar que el aprendizaje de la red esté supeditado a la elección del algoritmo voraz. Es mejor que la red aprende por sí sola siempre y cuando el conjunto de características de entrada no sea excesivamente grande, lo cual no es el caso.

Además, la ventaja de usar técnicas de clasificación tan potentes como el deep learning es que puede ser capaz de encontrar combinaciones de características que otras técnicas no son capaces. El método voraz sólo puede saber una a una si la característica mejora la clasificación o no y podemos estar perdiendo posibles combinaciones que pueden ser las buenas.

Se han realizado experimentos con ambas técnicas y a partir de los resultados se han realizado ajustes para intentar mejorar esos resultados.

4.3. Curvas ROC

Para presentar los resultados he hecho uso de las curvas ROC. Estas curvas representan gráficamente, según los valores obtenidos, el número de falsos positivos frente a falsos negativos que tenemos en los resultados.

Un verdadero positivo es cuando el test clasifica a un cliente potencial como convertido y, efectivamente, ese cliente potencial fue convertido a cliente. Si el test lo ha clasificado como convertido y se trata de un cliente potencial que no fue convertido será un falso positivo. Cuando nuestro test dice que un cliente potencial no se va a convertir en cliente y se equivoca se trata de un falso negativo y si acierta en su predicción se trata de un verdadero negativo.

[3] https://cloud.google.com/blog/big-data/2017/03/using-machine-learning-for-insurance-pricing-optimization

		Valor real		
		p	n	total
Predicción	p’	Verdaderos positivos	Falsos positivos	P’
	n’	Falsos negativos	Verdaderos negativos	N’
	total	P	N	

Cuando no existen ni falsos positivos ni falsos negativos entonces nos encontramos ante un resultado perfecto, es decir, un 100% de tasa de aciertos: el clasificador nunca se equivoca. Lo normal es encontrarnos en un resultado entre el 50% y 100%. Un 50% es lo peor que nos puede pasar, porque el resultado no sería diferente a tirar una moneda al aire.

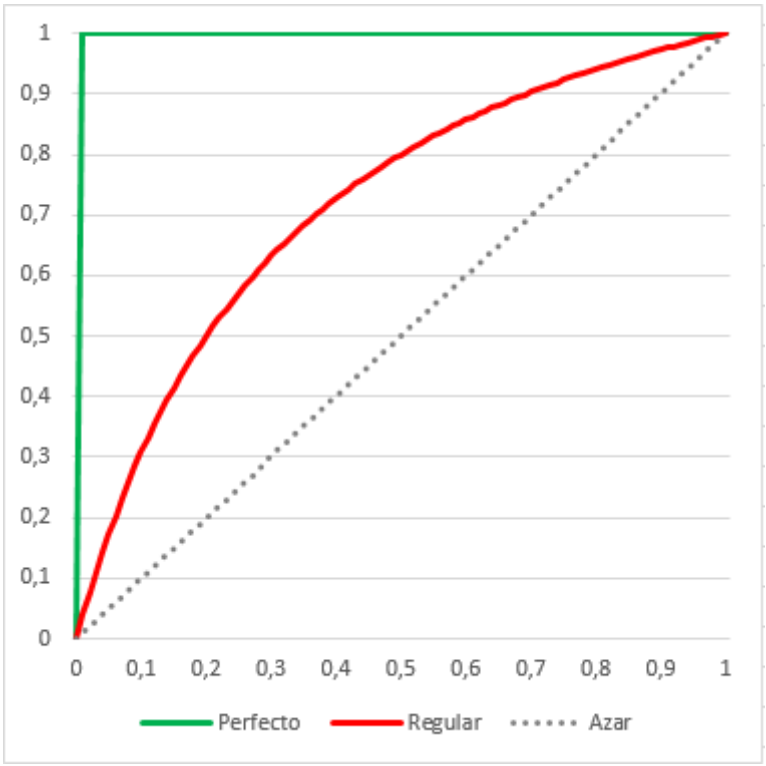

Figura 4: Ejemplos de curvas ROC

Calcular el área bajo la curva (AUC) nos da el *accuracy* o probabilidad de éxito de nuestro algoritmo.

5. Cuerpo del trabajo

En esta sección se incluyen los detalles de las implementaciones realizadas, la experimentación llevada a cabo, los ajustes realizados en función a esa experimentación y los resultados del trabajo, así como el análisis de dichos resultados.

5.1. Mínima redundancia y máxima relevancia

El primer algoritmo implementado para este trabajo es el algoritmo de selección de características *minimal-redundancy-maximum-relevance*, a partir de ahora mRMR. Pero antes de describir los detalles de este algoritmo hay que explicar los criterios basados en información mutua.

La información mutua entre dos características mide la dependencia mutua que existe entre ambas, es decir, la reducción de la incertidumbre, entropía, de una de las características debido al conocimiento del valor de la otra.

Tenemos un conjunto de características S y un conjunto de clases C, en nuestro caso compuesto de dos clases: convertido y no convertido. Hay que buscar un subconjunto de S que permita clasificar correctamente el conjunto de entrenamiento.

Se deduce que, aplicando la información mutua de una característica del conjunto de características S con una de las clases a clasificar obtendremos una cuantificación de la información que comparten y una medida de la dependencia entre la característica y la clase. Esta es la manera en la que un algoritmo de selección de características basado en la información mutua funciona.

La información mutua se define mediante las funciones de densidad de probabilidad *p* conjunta entre *s* y *c*, las funciones de densidad marginales de cada una y el logaritmo de dividir ambas:

$$\mathrm{I(S;C)} = \sum_{s \in S} \sum_{c \in C} p(s,c) log \frac{p(s,c)}{p(s)p(c)}$$

Donde *s* es una característica seleccionada del conjunto de características S y *c* es una clase del conjunto de clases C.

La ecuación anterior es equivalente a las dos siguientes:

$$(1)\ H(S) - H(S|C)$$

$$(2)\ H(S) + H(C) - H(S,C)$$

Usando la primera ecuación, la entropía condicional H(S|C) tiene que ser calculada. Para ello se tienen que calcular las entropías $\sum$ H (X|C = c) y *p* (C = c), lo que es factible ya que C es un conjunto discreto. Por otro lado, usar la segunda ecuación implica calcular la entropía conjunta de S y C. Según Bonev, B.; Escolano, F. y Cazorla, M. (2008) la ecuación (1) es más rápida, debido a la complejidad de la estimación de la entropía que depende del número de ejemplos.

Tenemos, pues, un método de selección de características basado en información mutua con el propósito de conseguir un subconjunto de S que conjuntamente tenga la máxima dependencia con la clase *c* y aplicando la ecuación (1):

$$máx D(S,c), D = H(S) - H(S|C)$$

Sin embargo, esto es inviable ya que, como se mencionó anteriormente, el cálculo de la entropía depende del número de ejemplos. El cálculo de la entropía suele ser muy costoso y produce malos resultados, sobre todo en espacios con una dimensión de características muy amplio y muchos ejemplos que clasificar. En su lugar es más exitoso aplicar el algoritmo que nos ocupa.

El algoritmo mRMR tiene dos partes. En primer lugar, trata de aplicar el criterio de *Máxima Relevancia*, a partir de ahora MaxRel, que selecciona las características con más información mutua con la clase *c*, esto es, encuentra un subconjunto de S que maximice $D(S,c)$, donde D se ha aproximado como el valor medio de la información mutua entre cada una de las características que componen el subconjunto S y la clase:

$$D = \frac{1}{|S|} \sum_{x_i \in S} I(x_i; c)$$

Segundo, trata de minimizar la redundancia de las características seleccionadas por el criterio anterior, de tal manera que si dos variables presentan alta dependencia el poder discriminatorio no disminuye si se excluye alguna de ellas. Lo hace aplicando la siguiente ecuación:

$$mín\ R(S), R = \frac{1}{|S|^2} \sum_{x_i \in S} I(x_i; x_j)$$

Donde las variables x_i y x_j son dos características cualesquiera tal que $i \neq j$. Finalmente se combinan ambas de acuerdo con la siguiente expresión:

$$\text{máx } \Phi(\text{D}, \text{R}), \Phi = \text{D} - \text{R}$$

Este algoritmo consigue maximizar la relevancia de la característica seleccionada y, al mismo tiempo, minimiza la redundancia entre esa característica y el resto de características pertenecientes al conjunto S. Existe un método iterativo que optimiza la expresión anterior, suponiendo un conjunto de datos de entrada con M características $X = \{x_1, x_2, \ldots, x_M\}$ y siendo S un subconjunto de *m* características de X, el criterio es:

$$\text{máx} \left[I(x_j; c) - \frac{1}{m-1} \sum_{x_i \in S_{m-1}} I(x_j; x_i) \right]$$

En la primera iteración del algoritmo el término de redundancia es nulo, entonces la primera variable seleccionada por mRMR es la misma que la escogida por MaxRel: aquella que tenga la máxima relevancia con la clase (Rodríguez, 2009).

En un primer momento, se ha implementado la primera parte del algoritmo: aquella que selecciona únicamente las características por su relevancia con la clase. De entre un conjunto de unas cincuenta características se seleccionan las diez más relevantes.

Seguidamente, se ha implementado el algoritmo optimizado y ha habido diferencias entre las características seleccionadas por ambas implementaciones. De entre las diez características seleccionadas por cada una, seis han coincidido y cuatro son diferentes.

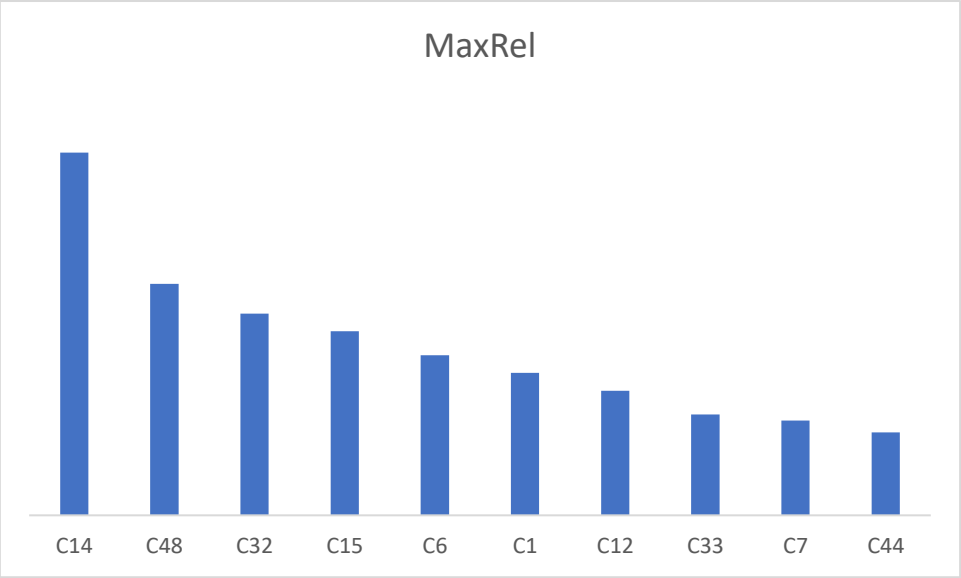

Figura 5: Características ordenadas por relevancia

Como se puede apreciar, la primera característica seleccionada por mRMR coincide con la primera seleccionada por MaxRel como ya se adelantó anteriormente. A partir de aquí, existen diferencias. Las características C6, C12, C33 y C44, por ejemplo, que fueron seleccionadas por MaxRel no se encuentran entre las seleccionadas por mRMR. Esto quiere decir que esas características, a pesar de ser relevantes respecto a la clase, son redundantes respecto al resto de características y no aportan información adicional.

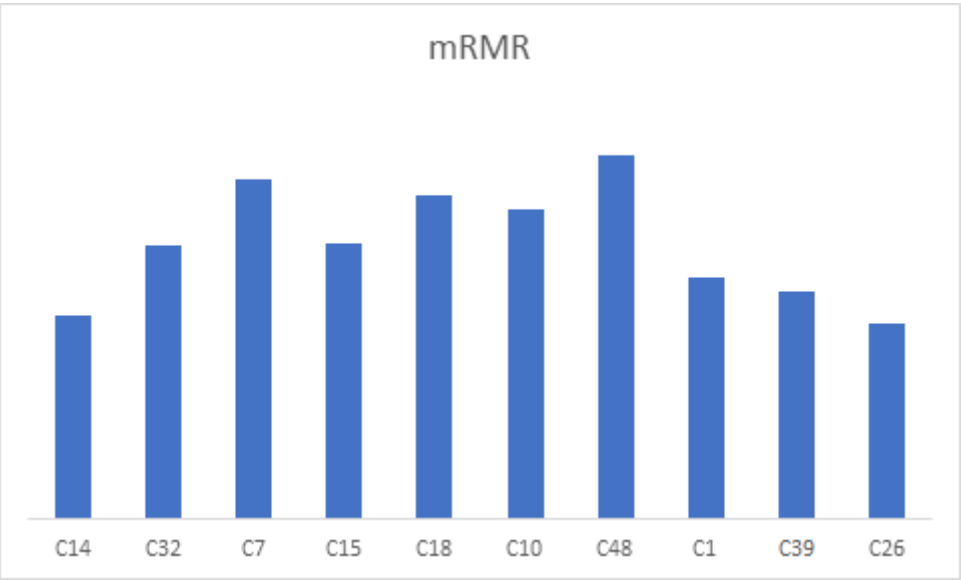

Figura 6: Características seleccionadas por mRMR

Las características seleccionadas por este algoritmo son las óptimas localmente, el conjunto óptimo global podría ser otro. Esto se puede comprobar probando las características obtenidas ante un clasificador.

Algunos métodos de clasificación pueden ser el de KNN o k vecinos más cercanos o el de agrupamiento o clustering no supervisado K-means. Pero los clasificadores más utilizados actualmente y con más potencial son las redes neuronales artificiales.

5.2. Algoritmo recurrente

El segundo algoritmo implementado en este trabajo es una red neuronal de tipo recurrente. La diferencia entre este tipo de redes respecto a las redes *feedforward* o unidireccionales es que estas redes forman ciclos, es decir, los datos no se mueven solamente hacia delante, sino que también pueden moverse hacia capas neuronales anteriores. La ventaja está en que la red almacena la historia reciente de la secuencia lo que permite que la salida ante unas determinadas entradas pueda variar en función de la configuración interna actual de la red. Esto lo hace por medio de una representación en forma de estado.

No se ha utilizado las características seleccionas por mRMR para entrenar a la red, sino que se ha utilizado todo el conjunto de características. La red neuronal es un método alternativo que aprenderá ella misma las características que sean relevantes y el peso que deberá darles en combinación con el resto. Después del entrenamiento y ya acabados los ajustes de la red entonces se comparan los resultados de ejecutar la red con el total de características y de ejecutarla pasándole las características seleccionadas por mRMR para comparar los resultados obtenidos en cada clasificación.

El algoritmo o tipo de red utilizado es la LSTM (del inglés *long short-term memory*) que resuelve los problemas de las redes recurrentes tradicionales de ser incapaces de aprender cuando la información relevante guardada internamente se encuentra muy atrás en el tiempo.

Aquí se introduce el término de celda. Una celda en una red LSTM es un conjunto de neuronas aisladas que comparten un estado *u* que es el estado de la celda. Las celdas están organizadas por bloques. Estos bloques poseen unas compuertas de entrada (φ) y unas compuertas de salida (γ) que deciden qué datos entran y salen al bloque de las celdas. Las entradas en este modelo son las entradas de la red y las salidas de la red en el instante anterior. La compuerta de entrada se encarga de permitir o impedir el acceso de estos valores al interior del bloque de memoria. La compuerta de salida realiza una acción similar sobre la salida del bloque, tolerando o reprimiendo la difusión del estado al resto de la red.

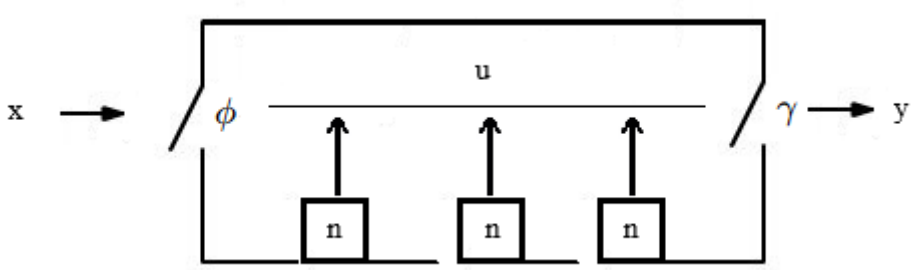

Figura 7: Una celda LSTM

En la Figura 7 se muestra cómo varias neuronas se conectan al "bus" del estado *u* actualizándolo dependiendo de la salida de cada una de las neuronas.

La activación de las compuertas de entrada y de salida del i-ésimo bloque de celdas en un instante t se calcula como:

$$X = g\left(\sum_{j=1}^{n_M}\sum_{k=1}^{n_C} w_{i,jk}\cdot h_{jk}[t-1] + \sum_{j=1}^{n_X} w_{i,j}\cdot x_j[t] + w_i\right),$$

$$X = \varphi_i[t] = \gamma_i[t]$$

Donde h es la celda correspondiente y g es la función de activación. La función de activación utilizada es la sigmoide. El comportamiento de esta función y el por qué es la utilizada viene explicado en el siguiente capítulo.

La variable n_X representa el número de neuronas y n_M y n_C representan el número de bloques y celdas, respectivamente.

El estado interno de la celda de memoria se calcula sumando la entrada modificada por la compuerta correspondiente con el estado en el instante $t-1$:

$$u_{ij}[t] = u_{ij}[t-1] + \varphi_i[t] \cdot g$$

La salida de la celda se calcula ajustando el estado mediante la función de activación y multiplicando el valor resultante por la activación de la compuerta de salida:

$$h_{ij}[t] = \gamma_i[t] \cdot g$$

Los pesos que inciden en las compuertas de entrada y salida se suelen iniciar de forma que $\varphi_i[0]$ y $\gamma_i[0]$ estén cerca de 0; de esta manera los bloques de memoria están desactivados inicialmente y el entrenamiento se centra en las conexiones directas entre la entrada y las neuronas de salida. Así, el protagonismo de los bloques de memoria va aumentando paulatinamente conforme el algoritmo de aprendizaje determina su rol (Pérez, 2002).

En resumen, una red LSTM está explícitamente diseñada para evitar el problema de dependencia a largo plazo, recordando información durante largos períodos de tiempo.

Una vez que ya hemos decidido el tipo de red recurrente a utilizar, el siguiente paso es diseñar la red neuronal siguiendo los pasos de diseño de toda red, esto es, elegir la función de activación que decida los valores de salida de las neuronas artificiales, el algoritmo de actualización de los pesos sinápticos de las entradas de cada neurona y la topología de la red: el número de capas, el número de neuronas por cada capa y la interconexión entre ellas.

5.3. Diseño de la red neuronal

Función de activación

Para obtener en las salidas valores comprendidos entre 0 y 1 se utiliza una función de tipo sigmoide. Esta función viene definida por la siguiente fórmula:

$$\sigma(z) = \frac{1}{1 + \mathrm{e}^{-z}}$$

Donde *z* es el resultado de sumar las entradas de las neuronas multiplicadas cada una por su peso asociado.

Veamos un ejemplo: suponiendo una neurona artificial con tres entradas y sus tres pesos correspondientes con valores de entrada $x_1 = 2$, $x_2 = 1$ y $x_3 = 3$ y pesos $w_1 = -1$, $w_2 = -2$ y $w_3 = 2$, el valor de *z* sería 2 * (-1) + 1 * (-2) + 3 * 2 = 2. Si sustituimos este valor en la función definida anteriormente el resultado que obtendríamos sería $\sigma(2) = 0'88$, un valor comprendido entre 0 y 1.

Cuando z es un número positivo alto $e^{-z} \approx 0$ y $\sigma(z) \approx 1$. Es decir, cuando la suma de las entradas por sus pesos es un positivo alto, la salida de la neurona sigmoide es aproximadamente 1. En caso contrario, cuando z es un número muy negativo entonces $e^{-z} \to \infty$ y $\sigma(z) \approx 0$.

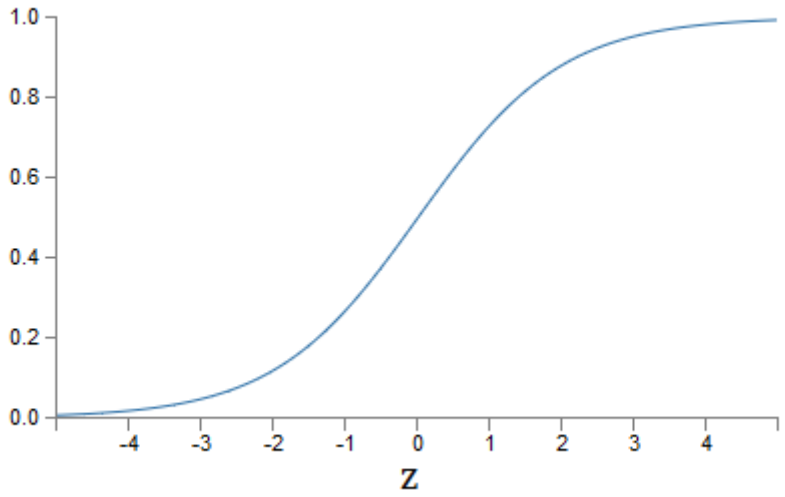

Figura 8: Gráfica de la función sigmoide

Para ver esto vamos a utilizar otro ejemplo basándonos en la neurona artificial sigmoide del ejemplo anterior modificando sus datos de entrada, pero manteniendo los valores de los pesos. Los nuevos valores de entrada son $x_1 = 2$, $x_2 = 5$ y $x_3 = 4$, por lo tanto, el nuevo valor de z sería 2 * (-1) + 5 * (-2) + 4 * 2 = -4. Con este nuevo valor de z el resultado de aplicar la función sigmoide es $\sigma(-4) = 0'0179$, aproximadamente 0.

Este comportamiento resulta muy parecido a simplemente tener una salida binaria, 0 y 1. Pero esta función facilita la elección de pequeños cambios en los pesos para lograr cualquier pequeño cambio deseado en la salida. Así, hace que sea mucho más fácil averiguar cómo el cambio de los pesos cambiará la salida (véase el apartado siguiente, actualización de los pesos sinápticos). Además, el tener como salida cualquier número real entre 0 y 1 como 0´173… y 0´689… puede ser útil para representar un valor de probabilidad de pertenencia. E incluso, para casos en los que necesitamos una clasificación exacta de 0 y 1, podemos establecer una convención para tratar esto, por ejemplo, decidiendo interpretar cualquier salida de al menos 0´5 como 1 y cualquier salida inferior a 0´5 como 0 (Nielsen, 2015).

Volvamos, de nuevo, al primer ejemplo de la neurona artificial sigmoide. Ahora, vamos a mantener los valores de entrada, pero modificamos los pesos sinápticos correspondientes. Los nuevos pesos son $w_1 = -1$, $w_2 = -1$ y $w_3 = 1$, el valor de z sería 2 * (-1) + 1 * (-1) + 3 * 1 = 0. Por tanto, $\sigma(0) = 0'5$. La modificación de los pesos a acarreado un gran cambio en el valor de la salida[4].

Como se puede apreciar la modificación de los pesos es muy importante si queremos obtener resultados óptimos por lo que hay que seguir un adecuado método de reemplazo de

[4] Para valores de z positivos el resultado de σ será un número mayor que 0'5 y para valores negativos un número menor que 0´5. Para un z igual a 0 el valor de σ es exactamente 0'5.

los valores de los pesos. La función sigmoide es muy útil para este objetivo porque es fácilmente derivable, lo cual es una gran ventaja para actualizar pesos aplicando el método explicado a continuación.

Actualización de los pesos sinápticos

Para ir actualizando los pesos de cada una de las entradas de las neuronas utilizamos el método del **descenso por gradiente**. Este método va a buscar encontrar los pesos sinápticos y la polarización que minimizan el error cuadrático de cada patrón de ejemplo.

La función de error cuadrático es la medida del error que describe la adecuación de la salida proporcionada por la red al valor deseado, definida para el instante t como:

$$E[t] = \frac{1}{2}\sum_{1=1}^{n}(d_i[t] - y_i[t])^2$$

Donde $d_i[t]$ es la salida deseada para la i-ésima neurona de salida en el instante t e $y_i[t]$ es la salida correspondiente de la red.

El descenso por gradiente consiste en ajustar los pesos derivando parcialmente la función de error cuadrático respecto a cada peso a ajustar, de manera que cada nuevo peso se ajusta de acuerdo con el siguiente criterio:

$$w_i[n+1] = w_i[n] - \alpha\frac{\partial E[n]}{\partial w_i[n]}$$

Donde $w_i[n]$ es el valor del peso actual, $w_i[n+1]$ es el nuevo valor que va a tomar el peso en la siguiente iteración y α es una constante llamada *learning rate* o tasa de aprendizaje que ha de tomar un valor convenientemente pequeño. Por lo tanto, el incremento del peso en la siguiente iteración es:

$$\Delta w_i[n] = w_i[n-1] - w_i[n] = -\alpha\frac{\partial E[n]}{\partial w_i[n]}$$

Para valores positivos muy pequeños de la tasa de aprendizaje este método permite que la función de error cuadrático descienda en cada iteración. La tasa de aprendizaje α tiene, por tanto, una enorme influencia en la convergencia de este método. Si α es pequeña, la velocidad de aprendizaje aumenta, pero existe el riesgo de que el proceso de aprendizaje diverja y el sistema se vuelva inestable.

Es habitual añadir un término *momento* que en ocasiones puede acelerar el aprendizaje y reducir el riesgo de que el algoritmo se vuelva inestable. La nueva ecuación de actualización del parámetro ajustable w_i tiene la forma:

$$\Delta w_i[n] = -\alpha\frac{\partial E[n]}{\partial w_i[n]} + \mu\,\Delta w_i[n-1]$$

Donde μ es la constante del momento.

El efecto del momento es el siguiente: si la derivada parcial del error tiene el mismo signo algebraico durante varias iteraciones seguidas, el término $\Delta w_i[n]$ irá creciendo y el incremento del parámetro será mayor; si la derivada parcial cambia de signo constantemente, el valor de $\Delta w_i[n]$ se va reduciendo y el parámetro se ajusta de forma más precisa (Pérez, 2002).

Ya habíamos visto en el apartado anterior que es importante suavizar el comportamiento de la actualización de pesos de manera que pequeños cambios en los pesos sinápticos provoquen pequeños cambios en la salida de la neurona artificial.

Al aplicar la regla de la cadena se tiene que la derivada parcial de la función de error cuadrático respecto a un peso sináptico cualquiera se expresa en función de dos derivadas: la derivada del error respecto al potencial resultante *z*, que indica cómo varía el error al variar la entrada de la neurona; y la derivada del potencial resultante *z* respecto al peso, que indica cómo varía la entrada de la neurona al variar el peso. La expresión es la siguiente:

$$\frac{\partial E[n]}{\partial w_i[n]} = \frac{\partial E[n]}{\partial z[n]} \frac{\partial z[n]}{\partial w_i[n]}$$

El resultado de la segunda derivada es la entrada correspondiente a la neurona y al resultado de la primera derivada se le denomina delta (δ). Por lo que la expresión anterior puede representarse también como:

$$\frac{\partial E[n]}{\partial w_i[n]} = \delta * x_i[n]$$

Por lo tanto, el incremento del peso sináptico se nos queda como,

$$\Delta W_i[n] = -\alpha * \ \delta * x_i[n]$$

Y si hacemos uso del momento,

$$\Delta w_i[n] = -\alpha * \ \delta * x_i[n] + \mu \, \Delta w_i[n-1]$$

Para calcular el valor de δ también hay que aplicar la regla de la cadena de la siguiente manera.

$$\delta = \frac{\partial E[n]}{\partial x_i[n]} \frac{\partial x_i[n]}{\partial z[n]}$$

La derivada parcial de la entrada de la neurona respecto de *z* es la derivada de la función de activación. Aquí nos encontramos con la ventaja de haber utilizado la función sigmoide σ como función de activación por tratarse de una función que es fácilmente derivable ya que su derivada es $\sigma * (1 - \sigma)$, o lo que es lo mismo:

$$\sigma' = \frac{d\sigma}{dz} = \frac{\mathrm{e}^{-\mathrm{z}}}{(1 + \mathrm{e}^{-\mathrm{z}})^2}$$

La derivada parcial de la función de error cuadrático respecto de la entrada de la neurona es el resultado de cambiar el signo a la diferencia entre la salida deseada y la salida real: $-(d_i[n] - y_i[n])$. Cuando nos encontramos en la neurona de salida esto es aplicable directamente, pero cuando nos encontramos en la neurona de una capa oculta hay que seguir aplicando la regla de la cadena, puesto que no podemos determinar cuál es la salida deseada para una neurona oculta. En este caso, el resultado de la derivada parcial es el sumatorio de los productos de los deltas y los pesos. Por lo tanto, el valor de δ dependiendo de si nos encontramos en una neurona de salida (1) o en una neurona oculta (2) es:

$$(1) - (d_i[n] - y_i[n]) * \ \sigma'$$

$$(2) - \sum_{i=1}^{n} (\delta * w_i[n]) * \sigma'$$

Como la ecuación (2) es más complicada, añadir muchas capas ocultas en la red neuronal dificulta el proceso de aprendizaje aumentando el costo del mismo. Con el fin de evitar esto y de aplicar el algoritmo de red neuronal recurrente explicado en el capítulo 5.2, se ha implementado una única capa oculta como se muestra a continuación.

Topología de la red

Una red neuronal de tipo recurrente tiene la forma de un grafo dirigido con ciclos. Los ciclos en la red que he implementado están dentro de la misma capa oculta, esto es, las salidas de la capa en un instante siguen circulando por la red y serán parte de las entradas de la capa en el instante siguiente junto al resto de entradas de la red.

Vamos a entender capa oculta como bloque de memoria de celdas de una red LSTM puesto que son equivalentes en el sentido de que ambas funcionan como caja negra entre la capa de entrada y la capa de salida. Puede haber varios niveles de bloques de memoria interconectados entre sí como varias capas ocultas, pero yo sólo he implementado una.

Para visualizar la forma del grafo de la red implementada vamos a ayudarnos del **TensorBoard**. El TensorBoard es un conjunto de herramientas de visualización de TensorFlow que permite visualizar la red neuronal, trazar métricas cuantitativas sobre la ejecución de la red y mostrar todos los datos que se desee de la misma, como entradas, salidas, pesos, etc.

Un ejemplo de visualización:

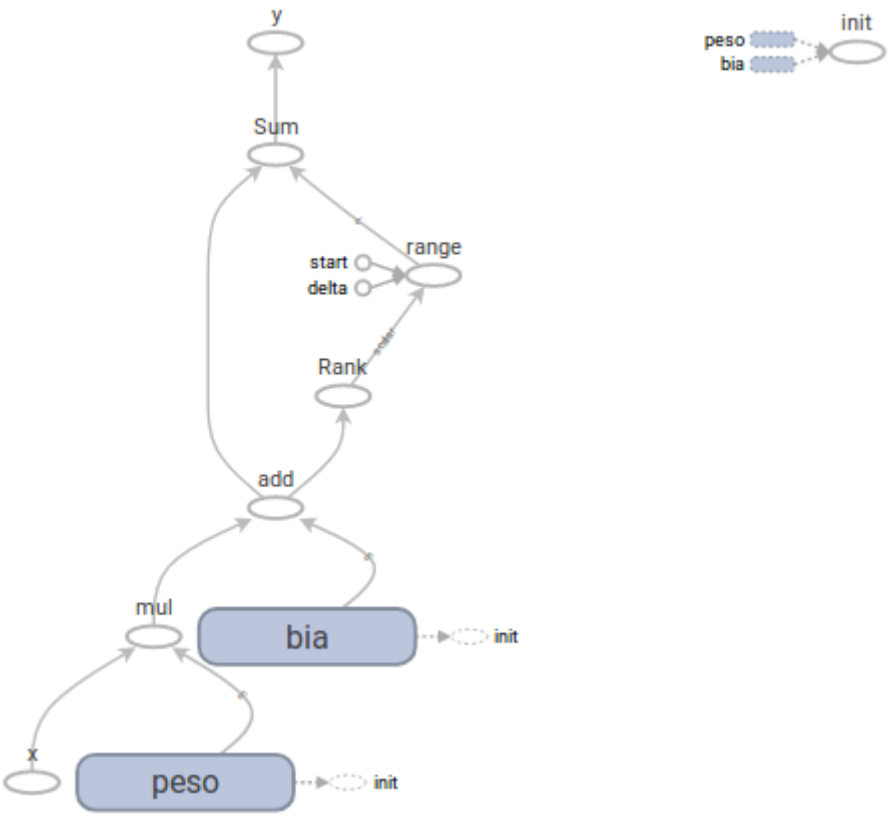

Figura 9: Red neuronal simple en el TensorBoard

En este ejemplo simple se puede ver una sola neurona donde el valor de entrada *x* se multiplica al peso, que es un valor inicializado internamente, y el resultado de esa operación se suma a una bia[5] también inicializada internamente. Esta operación se repite para cada dato de entrada, encontrándose conectada a un sumatorio que es el que produce el dato de salda *y*.

Este es nuestro punto de partida. Esta neurona simple será la neurona de salida de la red neuronal, pero no va a recibir directamente los datos de entrada. Las entradas de esta neurona procederán de los datos de salida de las neuronas pertenecientes a una capa oculta anterior. Las neuronas de esa capa oculta serán las que reciban los datos de entrada del conjunto de entrenamiento.

En la Figura 10 se puede observar cómo la capa oculta se muestra como una caja negra: sólo se ve lo que entra y lo que sale, pero no su interior. La salida de la capa oculta está conectada directamente con la parte mostrada en la Figura 9. Obsérvese que los pesos y las bias se han inicializado con datos aleatorios y que además se ha aplicado el descenso por gradiente.

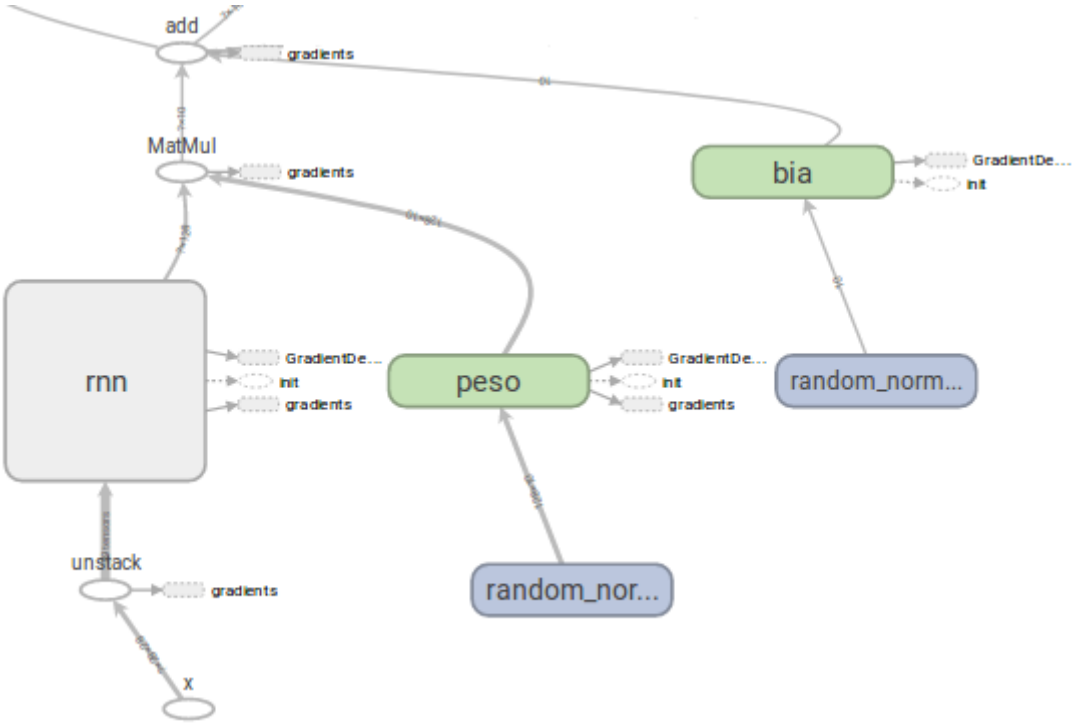

Figura 10: Topología LSTM

Si expandimos el bloque en el TensorBoard se nos muestra el interior de la capa oculta. Está formado por celdas de neuronas artificiales, los estados recurrentes que persisten en el modelo y, en definitiva, los elementos típicos de un bloque de memoria LSTM explicados con anterioridad.

[5] Las bias son un "sesgo" o peso adicional que se suman al resultado de multiplicar los pesos sinápticos por las entradas. Se usan para evitar que el estado inactivo de la neurona se deba a multiplicar por 0. Se actualizan de la misma manera que los pesos sinápticos.

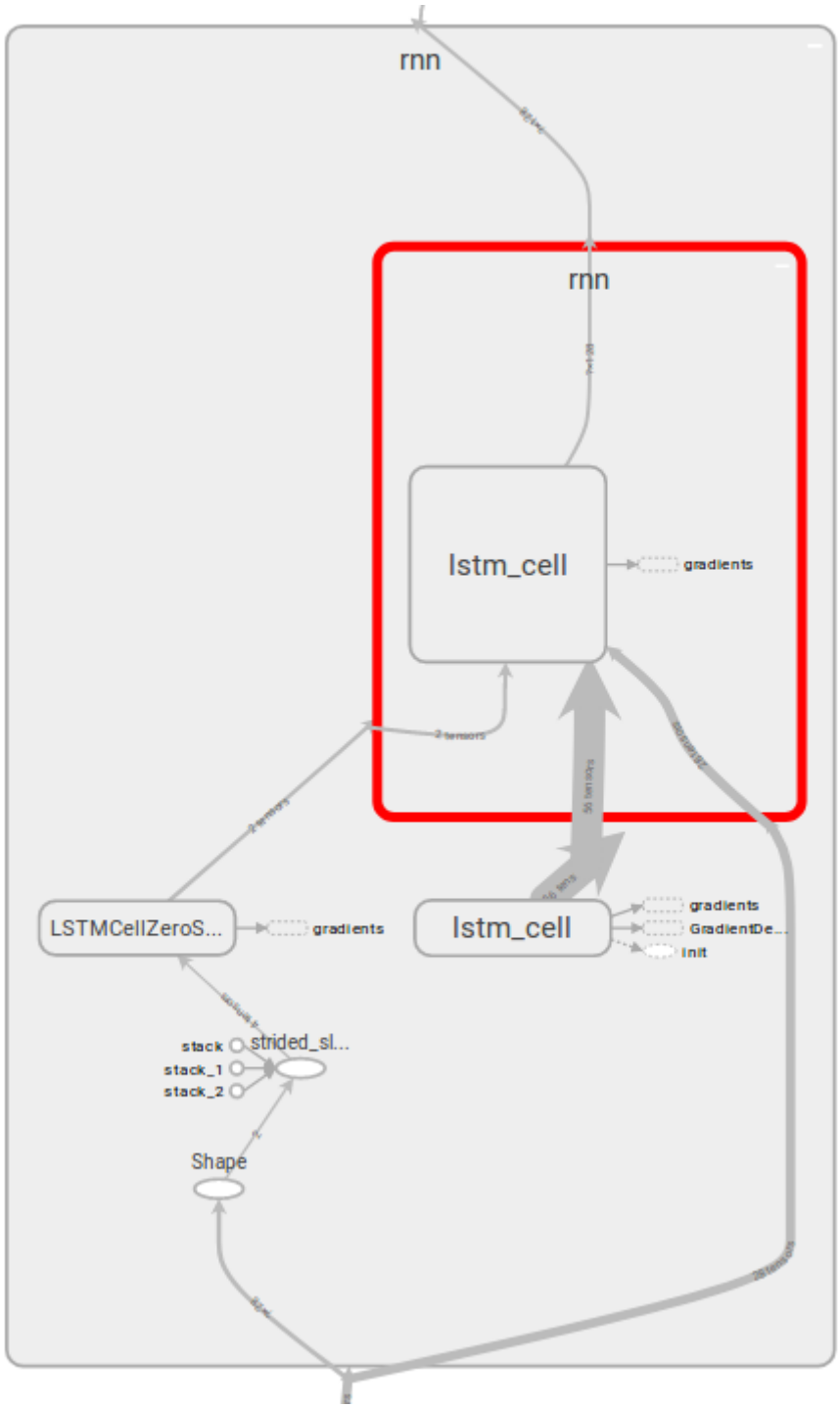

Figura 11: Celdas representadas en TensorBoard

En TensorFlow para crear las celdas de una red neuronal recurrente LSTM se hace uso de las LSTMCell[6] que son una implementación ya dada de este tipo de celdas a la que puedes configurar indicándole parámetros como el número de neuronas que quieres que haya en la celda, la función de activación a utilizar o la inicialización de los pesos, entre otros.

De la misma manera que con el bloque si expandiésemos las celdas también se mostraría su interior.

[6] https://www.tensorflow.org/api_docs/python/tf/contrib/rnn/LSTMCell

Con la red implementada ya sólo nos queda entrenarla.

Entrenamiento

El entrenamiento de una red neuronal consiste en realizar experimentos sobre la red e ir ajustando los parámetros de la red según los resultados. Una red neuronal se tiene que entrenar hasta que los resultados del test sean aceptables.

Los parámetros ajustables de la red neuronal son:

- El número de capas ocultas y el número de neuronas artificiales por cada capa oculta de la red.
- Los pesos sinápticos y bias.
- El *learning rate* o α.
- El momento (μ).
- El umbral que decide el valor de la neurona de salida.
- Los errores obtenidos.

Como ya se ha mencionado antes, se ha implementado una única capa oculta de neuronas. El número de neuronas de esta capa es importante por varios motivos: si el número de neuronas es excesivo puede provocar *overfitting* o sobre-entrenamiento en el que la red particulariza, no generaliza; por el contrario, si el número de neuronas es pequeño puede provocar *underfitting*, es decir, la red no soluciona el problema. Lo recomendable es empezar a entrenar con pocas neuronas en la capa oculta e ir añadiendo neuronas durante el entrenamiento de manera iterativa hasta que se alcance un resultado aceptable. El número inicial de neuronas artificiales en la capa oculta ha sido de 5.

El siguiente paso es la inicialización de los pesos y bias. Lo recomendable aquí es que los valores de los pesos al comienzo del entrenamiento sean pequeños. En mi caso, he generado valores iniciales aleatorios dentro del rango [-0´5, 0´5]. En las siguientes fases del entrenamiento y a medida que se van añadiendo neuronas artificiales a la red, es importante guardar los valores que los pesos sinápticos van tomando mientras la red aprende para que en futuros entrenamientos los pesos aprovechen los ajustes de entrenamientos previos y no vuelvan a empezar con valores aleatorios.

La tasa de aprendizaje α tiene que ser un valor muy pequeño como ya se explicó en el apartado de actualización de pesos sinápticos. El valor inicial de α ha sido de 0´01. Como este valor es una constante no se ajusta automáticamente durante el entrenamiento de la red. Es responsabilidad del entrenador modificar este parámetro e ir viendo como la red se desempeña para valores distintos de α.

No se ha hecho uso del momento al inicio del entrenamiento. Durante las primeras fases la red ha aprendido sin la constante μ para después ser añadida con la finalidad de observar las diferencias en el comportamiento de la red al entrenar con o sin este parámetro.

Al utilizarse la sigmoide como función de activación el resultado de la neurona de salida es un número comprendido entre 0 y 1 pero los valores de salida esperados son exactamente las clases 0 y 1, por lo que hay que decidir qué valores de salida se corresponden con una clase o con la otra. Se ha decidido en un principio que los valores superiores a 0´5 son considerados como 1, mientras que cualquier otro valor devolverá la salida 0. Este criterio se puede ir modificando para ver las diferencias en las tasas de acierto a medida que se modifica este umbral.

Con la finalidad de que la red recuerde su desempeño y del mismo modo que con los pesos sinápticos es importante guardar los errores obtenidos durante los entrenamientos.

El entrenamiento de la red viene representado por el siguiente diagrama:

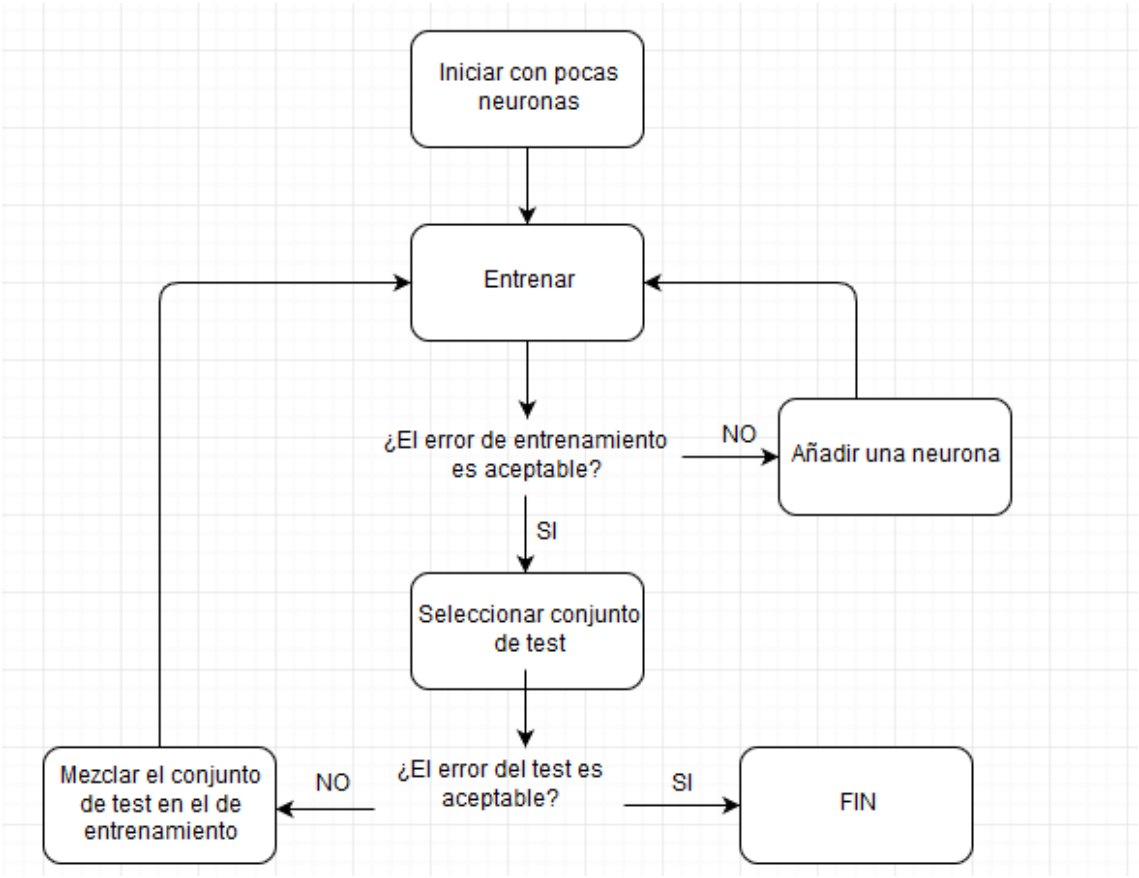

Figura 12: Diagrama del proceso de entrenamiento

5.4. Resultados

En este capítulo se presentan los resultados del entrenamiento.

Se muestran los ajustes de la red como consecuencia de la experimentación, así vemos los resultados que se han obtenido con ciertos ajustes y las mejores respecto a ajustes anteriores. La mayoría de los resultados se ven a través de curvas ROC, aunque también se muestran otras gráficas.

No se van a mostrar resultados que no difieran demasiado, solamente aquellos que aporten una gran mejoría al desempeño de la red.

Con los valores iniciales comentados en el capítulo anterior se ha conseguido la siguiente curva ROC:

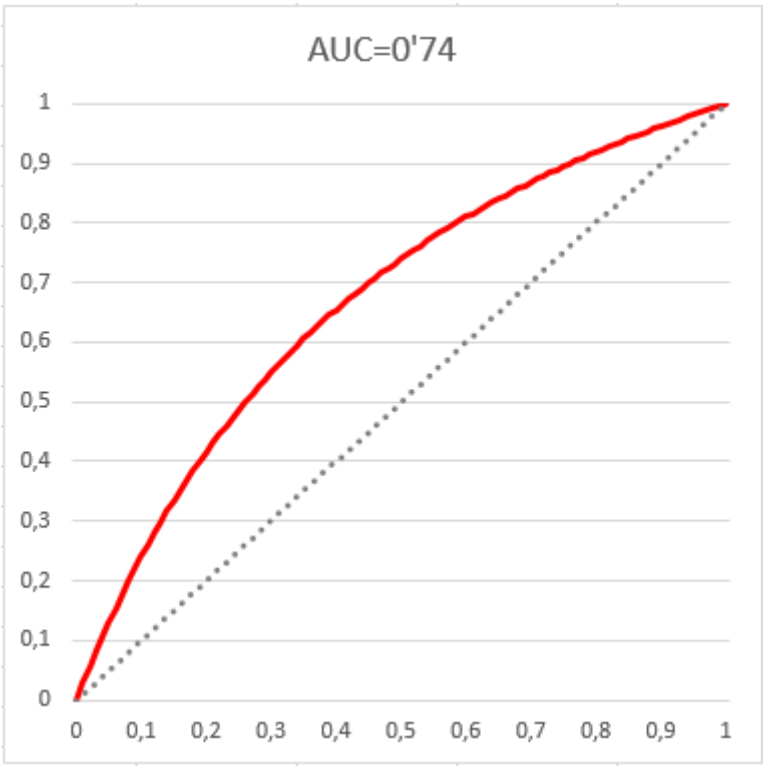

Figura 13: Resultado: Primer resultado

Este no es un buen resultado. De cada diez clientes en los que nuestro algoritmo nos aconseja que invirtamos, perderíamos tres. Además, dejaríamos de ganar clientes que nuestro algoritmo habría descartado.

Como guía para interpretar los resultados en las curvas ROC se han establecido los siguientes intervalos para los valores de AUC:

[0´5, 0´6): Test malo.

[0´6, 0´75): Test regular.

[0´75, 0´9): Test bueno.

[0´9, 0´97): Test muy bueno.

[0´97, 1): Test excelente.

Aunque se encuentra cerca, nuestro test ni siquiera llega a unos resultados de los que pueda considerarse un test bueno, por lo que la red requiere de muchos ajustes para intentar mejorar este resultado.

En TensorFlow se puede dividir la ejecución del algoritmo en épocas e iteraciones. Una época se corresponde con cada intervalo de tiempo en el cual el algoritmo ha entrenado con todos los ejemplos del conjunto de entrenamiento. En TensorFlow el conjunto de entrenamiento puede dividirse en "lotes" llamados batch. Cada batch se compone de un subconjunto de ejemplos del total de ejemplos. Así, en lugar de recibir los ejemplos de uno en uno el algoritmo puede recibirlos en lotes. Cada pase de un lote de ejemplos es una iteración.

Recordemos que para este trabajo se dispone de un conjunto de veinte mil ejemplos, de los cuales se utilizan la mitad para entrenar, puesto que también necesitamos un conjunto de ejemplos para test, que sería la otra mitad.

Después de varios ensayos se ha estimado un tamaño de batch de cien ejemplos, por lo que se necesitan cien iteraciones para completar una época. Cuantas más iteraciones más épocas se completan. Esto supone más tiempo de ejecución, pero mejores resultados. Con los ejemplos con los que se dispone, cien mil iteraciones completan mil épocas.

En los entrenamientos se han utilizado entre cincuenta mil y cien mil iteraciones indistintamente, es decir, entre quinientas y mil épocas, puesto que en este intervalo se han dado resultados similares. Con menos iteraciones el algoritmo daba peores resultados, necesitaba más iteraciones para llegar al mejor resultado posible con esos parámetros, esto es, hasta que el accuracy se quedaba estable.

Entonces, el número de lotes y épocas a utilizar han estado en la primera etapa de ajuste de parámetros. Entre los primeros ajustes realizados también están el número de neuronas a utilizar en la capa oculta. Se ha empezado con cinco neuronas y se han ido aumentando de una en una hasta encontrar el mejor número de neuronas en cuanto a resultado/tiempo.

La evolución de la tasa de aciertos ha ido aumentando hasta mantenerse estable a partir de las veinte neuronas, por lo que se he elegido este valor como el número de neuronas que tendrá nuestra capa oculta en los entrenamientos.

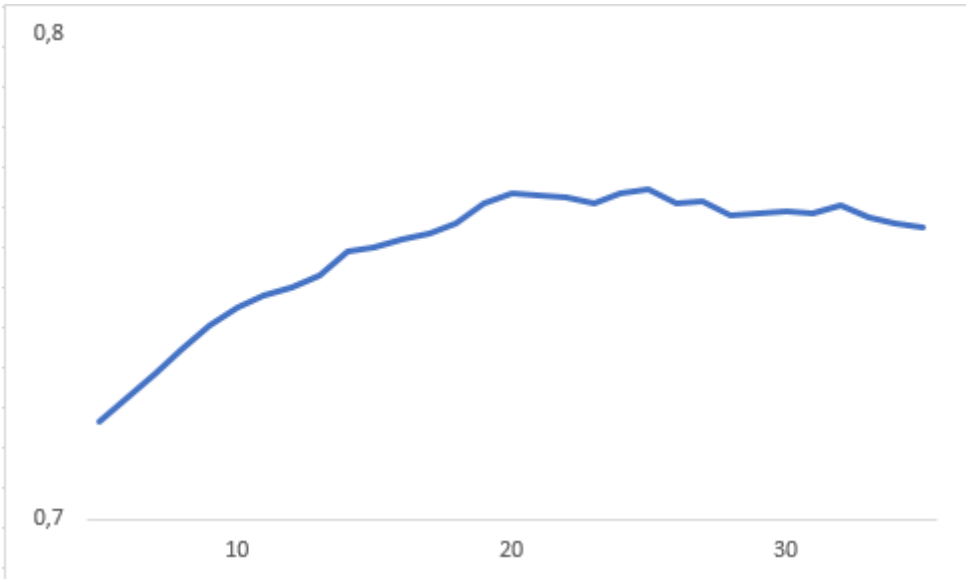

Figura 14: Resultado: Evolución de la tasa de aciertos al añadir neuronas

Como se puede observar en la Figura 14, con números superiores a veinte neuronas se ha conseguido tasas de acierto ligeramente superiores, pero se trata de mejorías casi inapreciables, por lo que veinte es la mejor opción puesto que escoger más no nos garantiza obtener una mejor tasa de aciertos, pero sí aumenta el tiempo de entrenamiento[7].

Cuando se añade una nueva neurona el vector de pesos aumenta. Estos nuevos pesos son iniciados aleatoriamente, pero los pesos de las neuronas que ya estaban se conservan de entrenamientos anteriores.

En la segunda etapa de entrenamientos, el siguiente valor a aumentar es el learning rate o tasa de aprendizaje. Este valor empieza siendo especialmente bajo y se va aumentando hasta encontrar una estabilidad en la tasa de aciertos.

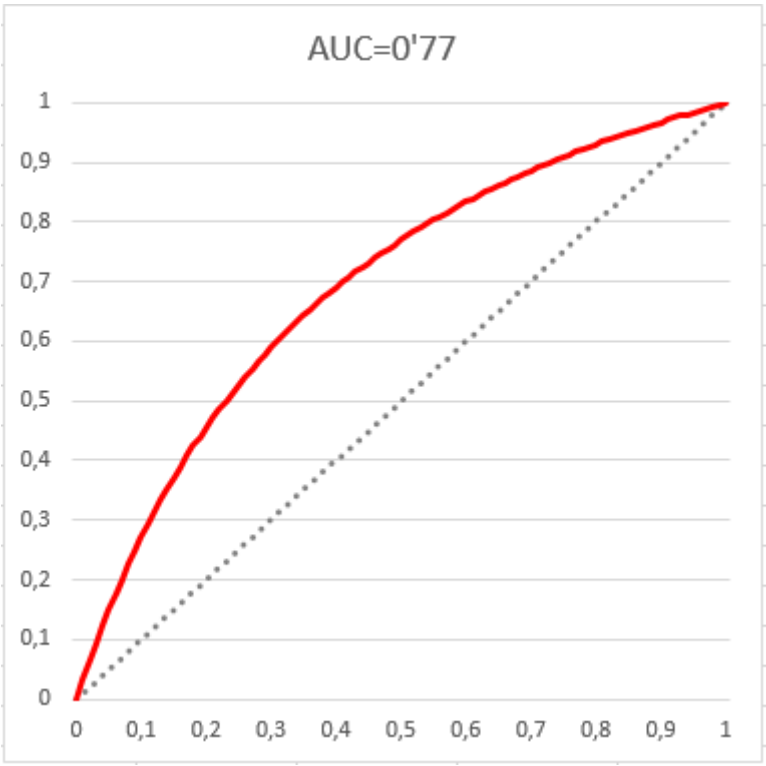

Figura 15: Resultado: Con α = 0'05

El primer valor con el que se ha probado el learning rate ha sido 0´01. Con valores inferiores los resultados han sido peores. Al incrementarlo, los resultados eran mejores. Desde 0´01 hasta 0´1 se ha ido incrementando en intervalos de 0´01 y a partir de 0´1 en adelante en intervalos de 0´1.

Hasta $\alpha = 0'1$ los resultados van mejorando, a partir de 0´1 se mantienen estables y a partir de 0´2 empeoran.

Si recordamos la fórmula utilizada para el cálculo del incremento de los pesos sinápticos explicada en el capítulo 5.3. Diseño de la red neuronal se podía utilizar, además de α, otra constante μ llamada momento que se utiliza para optimizar el cálculo del incremento de pesos y acelerar su operación. En estos resultados no se ha utilizado el momento, pero más adelante veremos cómo ha influido la constante μ en los resultados.

[7] Todos estos ensayos se han hecho con un valor de α igual a 0´01.

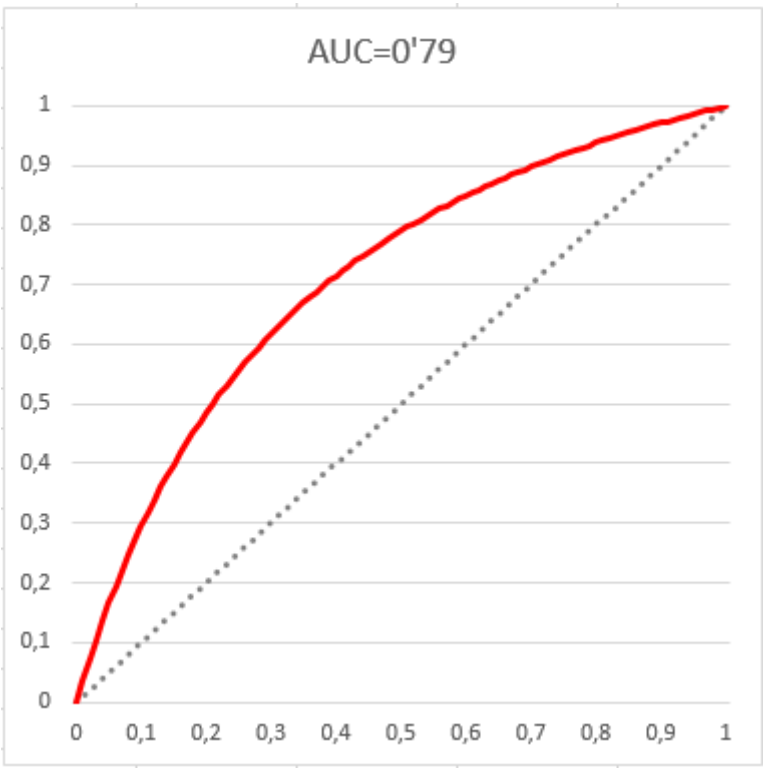

Figura 16: Resultado: Con α = 0'1

Aunque en la literatura 0´1 se considera un valor alto de tasa de aprendizaje y se recomienda utilizar valores más bajos, en mi caso valores más bajos no son adecuados porque empeoran el resultado. Quizá debido a que el ajuste demasiado bajo de los pesos provoca un estancamiento en algún mínimo local.

Las pruebas para ajustar la tasa de entrenamiento se han realizado con los siguientes parámetros:

- Tamaño de lote o batch: **100**
- Épocas: **750**
- Neuronas en la capa oculta: **20**

Con TensorBoard, además de visualizar los grafos que representan la red que hemos diseñado, también podemos ver gráficas de métricas obtenidas de ejecutar los entrenamientos de nuestra red gracias a lo que TensorFlow denomina el sumario de operaciones. Podemos decirle a TensorFlow qué métricas queremos ver representadas en el TensorBoard y mediante programación configurar las operaciones que TensorFlow va a "monitorizar". En realidad, lo que TensorFlow hace es escribir los resultados de esas métricas en ficheros que luego el TensorBoard lee y dibuja en forma de gráficas e histogramas.

Las métricas que yo he configurado para que el TensorBoard dibuje son las del accuracy o tasa de aciertos y las de la tasa de error. Podemos ver cómo evoluciona el accuracy a la par que el error va siendo minimizado con el paso de las épocas mediante el descenso por gradiente.

La siguiente gráfica del TensorBoard representa la evolución del accuracy en uno de los entrenamientos:

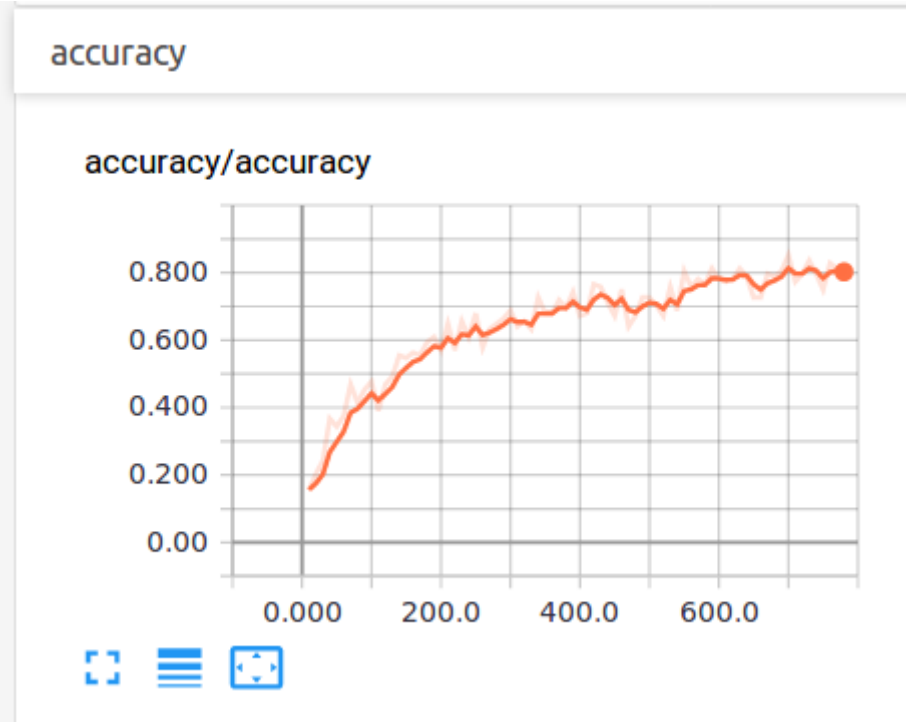

Figura 17: Resultado: Evolución del accuracy en un entrenamiento

La siguiente gráfica del TensorBoard representa la evolución del error en uno de los entrenamientos:

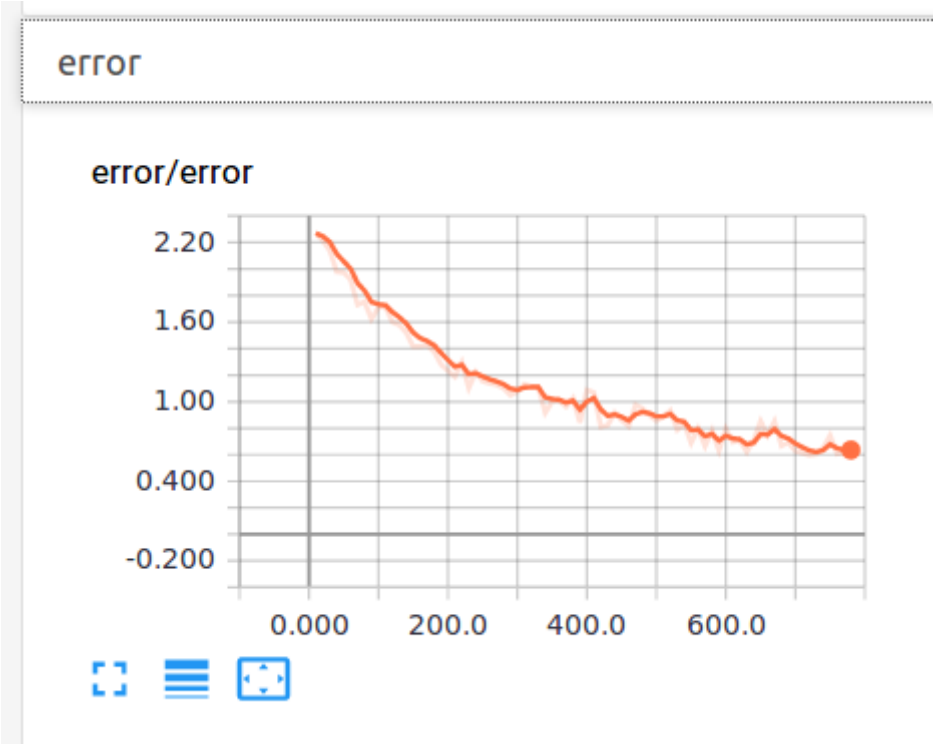

Figura 18: Resultado: Evolución del error en un entrenamiento

Cuando ya no se obtenían mejoras en el entrenamiento se ha añadido el parámetro momento μ en la fórmula de optimización del coste y se ha producido una ligera mejoría. Los mejores resultados se han obtenido para un μ igual a 0´8. Al contrario que con el learning rate, con el momento se obtiene más mejoría cuando aumenta su valor, obteniendo los peores resultados con 0´1 como valor de μ y mejorando a medida que este valor aumentaba.

Haciendo uso del momento:

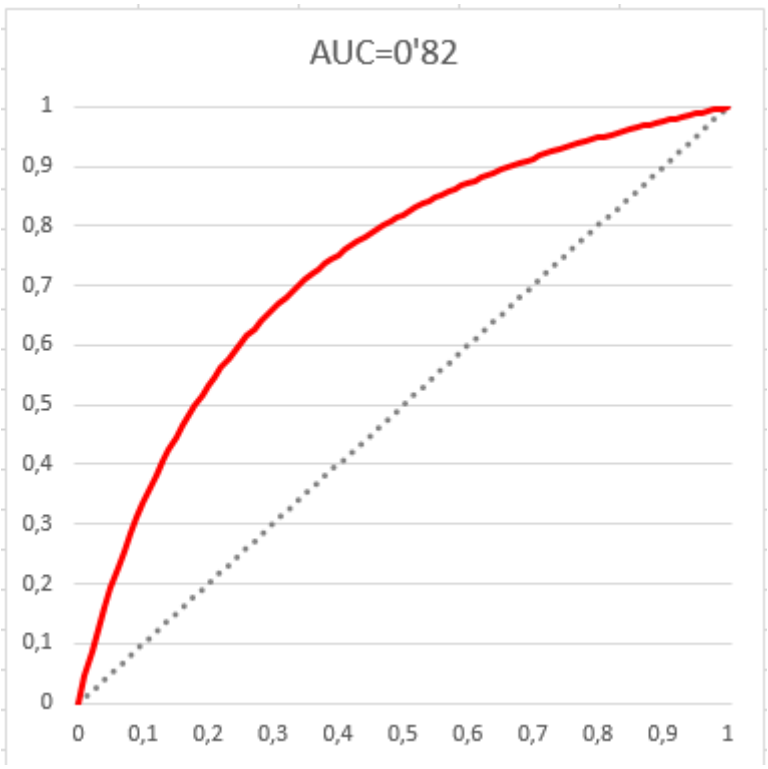

Figura 19: Resultado: Con momento μ = 0´8

Se ha probado con valores de momento comprendidos entre 0´1 y 0´9 en intervalos de 0´1. Las mejoras son significativas, en este caso μ ayuda mucho al optimizador de la función de coste a minimizar los errores obtenidos.

Los resultados obtenidos para los diferentes valores de μ, para un learning rate de 0'1, se encuentran en la siguiente tabla:

Resultados para diferentes valores de μ									
momento (μ)	0'1	0'2	0'3	0'4	0'5	0'6	0'7	0'8	0'9
accuracy	79%	80%	79%	80%	80%	80%	81%	82%	81%

Por último, se le ha pasado al algoritmo un conjunto de datos de entrada que no engloba el total de características de la base de datos, sino que engloba las diez características seleccionadas por el algoritmo mRMR.

Se han utilizado los parámetros que mejores resultados han dado durante el entrenamiento de la red con el conjunto del total de características.

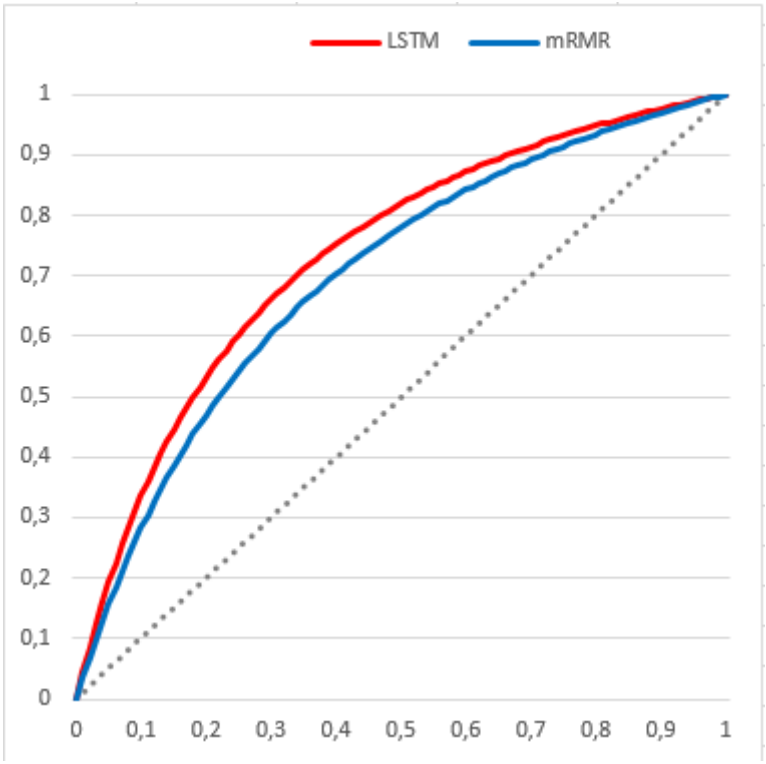

Figura 20: Resultado: LSTM vs mRMR

Los resultados de ejecutar el clasificador con las características obtenidas con el algoritmo de selección de características (78%) es peor que los resultados de ejecutar el clasificador con el total del conjunto de características (82%).

Esto puede deberse a que el algoritmo no funcione bien para un conjunto de características insuficiente o que la red diseñada sea capaz de encontrar combinaciones de características que mRMR no.

Desde los resultados obtenidos al principio del entrenamiento con los parámetros iniciales hasta los obtenidos con los últimos ajustes se ha logrado una ganancia de casi diez puntos en la tasa de aciertos, por lo que la red evoluciona y aprende con el tiempo y haciendo mejoras sobre ella.

Aun así, los resultados no son todo lo buenos que se esperaría de ellos. Unos muy buenos resultados superarían una tasa de aciertos del 90%.

Es posible que a partir de los datos de los que se dispone no se pueda clasificar bien este problema porque las características no sean las óptimas. De todas formas, no se puede concluir que las técnicas de machine learning no sean capaces de resolver el problema propuesto.

Podemos concluir que de los datos con los que disponemos no es suficiente para obtener una muy buena clasificación. Con las características apropiadas podría extraerse información relevante de los clientes potenciales y a partir de ellos determinar si podemos adquirirlos.

6. Conclusiones

En este trabajo se han estudiado varias técnicas de machine learning aplicables a la predicción de datos. Se ha propuesto un problema no abordado anteriormente en el que probar varias de las técnicas y los resultados, aunque no son del todo buenos, son prometedores en algunos sentidos. Lo positivo que se puede extraer, sobre todo de la red neuronal recurrente LSTM, es que la red ha evolucionado y ha conseguido mejorar sus resultados iniciales, por lo que no se puede descartar que se puedan obtener todavía mejores resultados en el futuro con nuevas mejoras en la red.

Obviamente, si comparamos estos resultados con otros resultados obtenidos en otros problemas de predicción tradicionales, los obtenidos aquí son mucho peores. Existen muchos ejemplos de *datasets* online que sirven de ejemplo para probar redes neuronales. Uno de los más conocidos es el conjunto de datos MNIST, una gran base de datos con 70.000 pequeñas imágenes de dígitos escritos a mano que puede ser usada para predecir el número dibujado. Este conjunto de datos es ideal para la predicción, pues se consigue con ellos una tasa de aciertos del 99%. Al pasar este conjunto de datos a mi red neuronal se dan excelentes resultados, por lo que se puede concluir que el problema está en los datos de los que se dispone y no en el diseño de la red.

Se han empeorado los resultados al aplicar *minimal-redundancy-maximum-relevance* sobre el conjunto total de características, perdiendo hasta cuatro puntos en la predicción del clasificador, es decir, la red neuronal.

El problema con el algoritmo de selección de características es que no se contaba con un conjunto de características apropiado en la base de datos. Esta técnica es aprovechable sobre todo cuando se cuenta con un altísimo número de características, del orden de millares. En lugar de partir de una base de datos, se podría aplicar alguna técnica de minería de datos capaz de extraer los datos del cliente potencial del internet, como por ejemplo de sus redes sociales o de su página web si el cliente es una organización. Si hacemos esto, la cantidad de datos puede ser enorme (estamos hablando de Big Data) y la mayoría de características que se pueden obtener se trataría de ruido y no serían útiles. En este caso es esencial aplicar el algoritmo de selección de características para que, de ese gran conjunto de características, obtenga las que podrían ser realmente relevantes y despreciar la mayoría de características que no nos aporten nada.

En este caso sí podríamos aprovechar esas características para pasárselas a la red neuronal y que entrene con ellas, ya que utilizar un desproporcionado número elevado de características, del orden del que se está proponiendo como trabajo futuro, obtenidas de aplicar técnicas de extracción de datos no sería óptimo sobre todo en tiempos de computación.

7. Bibliografía

Bonev, B.; Escolano, F. y Cazorla, M. (2008) *Feature selection, mutual information, and the classification of high-dimensional patterns: Applications to image classification and microarray data analysis*. Disponible en: DOI: 10.1007/s10044-008-0107-0

Elman, J. L. (1990) *Finding Structure in Time*, Cognitive Science, 14, pp. 179-211. Disponible en: DOI: 10.1207/s15516709cog1402_1

Escolano, F. [et al.] (2003) *Inteligencia artificial. Modelos, técnicas y áreas de aplicación*, Thomson. ISBN: 84-9732-183-9

Federico, L. (2005) *Entrenamiento de redes neuronales basado en algoritmos evolutivos*, Tesis doctoral. Disponible en: http://laboratorios.fi.uba.ar/lsi/bertona-tesisingenieriainformatica.pdf

Julián, G. (2014) *Las redes neuronales: qué son y por qué están volviendo*. En: Xataka. 30 diciembre 2014. [Consulta 23 julio 2017]. Disponible en: https://www.xataka.com/robotica-e-ia/las-redes-neuronales-que-son-y-por-que-estan-volviendo

López, R. (2014) *¿Qué es y cómo funciona "Deep Learning"?* 7 mayo 2014. [Consulta 25 julio 2017]. Disponible en: https://rubenlopezg.wordpress.com/2014/05/07/que-es-y-como-funciona-deep-learning/

Mak, M.; Ku, K. y Lu, Y. (1998) *On the improvement of the real time recurrent learning algorithm for recurrent neural networks*, Theory, Architectures, and Applications, 13. Disponible en: DOI: 10.1016/S0925-2312(98)00089-7

Nielsen, M. (2015) *Neuronal Networks and Deep Learning*, libro en línea. Disponible en: http://neuralnetworksanddeeplearning.com/index.html

Noguera, O. (2006) *Redes neuronales recurrentes: principios y aplicaciones*, Tesis doctoral. Disponible en: http://tesis.ipn.mx/bitstream/handle/123456789/2708/1017_2006_ESIME-CUL_MAESTRIA_noguera_sanchez_oscar.pdf

Olah, C. (2015) *Understanding LSTM Networks*. 27 agosto 2015. [Consulta 5 agosto 2017]. Disponible en: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Peng, H.; Long, F. y Ding, C. (2005) *Feature selection based on mutual information: criteria of max-dependency, max-relevance, and min-redundancy*. Disponible en: DOI: 10.1109/TPAMI.2005.159

Pereira, A. (2015) *Selección de características para el reconocimiento de patrones con datos de alta dimensionalidad en fusión nuclear*, Tesis doctoral. Disponible en: http://www-fusion.ciemat.es/PhDThesis/Pereira.pdf

Pérez, J. A. (2002) *Modelos predictivos basados en redes neuronales recurrentes de tiempo discreto*, Tesis doctoral. Disponible en: http://www.dlsi.ua.es/~japerez/pub/pdf/tesi2002.pdf

Pineda, F. (1987) *Generalization of back-propagation to recurrent neural networks*, American Physical Society. Disponible en: DOI: 10.1103/PhysRevLett.59-2229

Rodríguez, I. (2009) *Selección de variables mediante programación cuadrática*, Trabajo de Fin de Máster. Disponible en: https://repositorio.uam.es/handle/10486/10022

Rumelhart, D.; Hinton, G. y Williams, R. J. (1986) *Learning representations by back-propagation errors*, Nature, 323, pp. 533-536

Russell, S. J. y Norvig, P. (2004) *Inteligencia artificial: un enfoque moderno. Segunda edición*, Pearson. ISBN: 84-205-4003-X

Williams, R. J. y Zipser, D. (1989) *A learning algorithm for continually training recurrent neural networks*, Neural Computation, 1, pp. 270-280

Williams, R. J. y Zipser, D. (1990) *Gradient-Based Learning Algorithms for Recurrent Networks and Their Computational Complexit*, La Jolla, CA Press. California.