TRABAJO FIN DE MÁSTER

MÁSTER EN CIENCIAS DE DATOS E INGENIERÍA DE COMPUTADORES

Algoritmos de Deep Learning para el tratamiento de series temporales de clasificación

Autor
Alberto Armijo Ruiz (alumno)

Director
Salvador García López

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

19 de julio de 2019

Título del Proyecto: Subtítulo del proyecto

Nombre Apellido1 Apellido2 (alumno)

Palabras clave: palabra_clave1, palabra_clave2, palabra_clave3, ...

Resumen

Poner aquí el resumen.

Project Title: Project Subtitle

First name, Family name (student)

Keywords: Keyword1, Keyword2, Keyword3, ...

Abstract

Write here the abstract in English.

Yo, **Nombre Apellido1 Apellido2**, alumno de la titulación TITULACIÓN de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI XXXXXXXXX, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Nombre Apellido1 Apellido2

Granada a X de mes de 201 .

D. **Nombre Apellido1 Apellido2 (tutor1)**, Profesor del Área de XXXX del Departamento YYYY de la Universidad de Granada.

D. **Nombre Apellido1 Apellido2 (tutor2)**, Profesor del Área de XXXX del Departamento YYYY de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado ***Título del proyecto, Subtítulo del proyecto***, ha sido realizado bajo su supervisión por **Nombre Apellido1 Apellido2 (alumno)**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a X de mes de 201 .

Los directores:

Nombre Apellido1 Apellido2 (tutor1) **Nombre Apellido1 Apellido2 (tutor2)**

Agradecimientos

Poner aquí agradecimientos...

Capítulo 1

Introducción

1.1. Objetivos

1.2. Motivación

A día de hoy, la minería de datos en series temporales son un tipo de problema en auge. La Clasificación de Series Temporales (TSC, Time Series Classification en Inglés) es una de las áreas donde más investigación se está realizando en los últimos años debido a su gran número de aplicaciones en diferentes sectores como la industria, salud o transporte. Obtener una buena precisión en este tipo de problema puede ofrecer grandes beneficios por lo que existe un gran interés en este área.

La Clasificación de Series Temporales tiene como objetivo clasificar datos durante el tiempo basado en su comportamiento. Por ejemplo, clasificar el comportamiento de un servidor durante el tiempo puede ayudar a detectar anomalías y a que la resolución de cualquier problema asociado a una anomalía sea más rápido.

El desbalanceo de clases es un problema asociado a la clasificación y muy común en problemas de la vida real. La mayoría de los algoritmos clásicos suelen trabajar sobre la suposición de que el número de datos de cada clase están balanceados, por lo que el desbalanceo entre las diferentes clases hace que el rendimiento de estos algoritmos sea pobre. Esto hace que cuando la clase minoritaria es aquella que tiene más interés en clasificar, como por ejemplo detección de un terremoto; se necesite buscar una solución.

Existen diferentes soluciones para intentar abordar el problema de desbalanceo entre clases a diferentes niveles: a nivel de datos, a nivel del algoritmo o combinaciones de estas dos. Los métodos que abordar el problema a nivel de datos intentan restablecer el balance entre clases mediante el muestreo de datos, añadiendo datos de la clase minoritaria (oversampling), eliminando datos de la clase mayoritaria (undersampling) o una combinación de ambos. Los métodos que abordan el problema a nivel del algoritmo enfatizan en la clase minoritaria manipulando e incorporando diferentes parámetros como por ejemplo el peso de cada clase.

Por otro lado, las redes neuronales son unos de los modelos más famosos actualmente y se utilizan en un gran número diferentes de problemas obteniendo buenos resultados. Las redes formadas por LSTM (Long Short Term Memory) son un tipo de red que se utiliza en un gran número de problemas de gran complejidad como es la toma de decisiones en videojuegos, reconocimiento del habla o traducción de textos. Este tipo de red son especialmente buenas en problemas donde hay que procesar secuencias de datos, como es el caso de las series temporales. Además, la redes neuronales son capaces de trabajar con datos con alta dimensionalidad obteniendo buenos resultados a diferencia de otros métodos de minería de datos clásicos.

En este trabajo se utilizarán diferentes conjuntos de datos sobre Clasificación de Series Temporales. Los conjuntos de datos utilizados de clasificación binaria y contienen desbalanceo entre las clases. Para el procesamiento de dichos conjuntos de datos se utilizará una red de LSTM y diferentes métodos de procesamiento para intentar obtener un buen rendimiento.

Para paliar el desbalanceo de clases se utilizarán métodos de oversampling, en concreto SMOTE; este algoritmo es uno de los más utilizados para oversampling y suele ofrecer buenos resultados. Los resultados obtenidos se compararán con algoritmos clásicos de Machine Learning como por ejemplo SVM.

1.3. Organización de la memoria

Capítulo 2

Antecedentes

En este capítulo se describirán los aspectos generales de los principales componentes del estudio. Primero se describirán diferentes algoritmos básicos usados para clasificación de series temporales. Tras esto se explicarán qué son las LSTM, su funcionamiento y estructura. Por último, se describirá el problema de desbalanceo de clases y diferentes enfoques para solucionarlo.

2.1. Clasificación de series temporales. Algoritmos Básicos

En esta sección se describirán algoritmos básicos para predicción de series temporales, así como algoritmos utilizados cuando se trabaja con series temporales de clasificación.

2.1.1. Series temporales de regresión.

Cuando tratamos con series temporales nos encontramos con un problema de regresión clásico; en principio, se puede intentar utilizar métodos clásicos de regresión, pero normalmente no suelen dar buenos resultados. Para este tipo de problemas, el modelo más utilizado es ARIMA.

ARIMA se trata de un modelo clásico para series temporales. Dicho modelo se trata de una combinación de otros modelos, modelos autoregresivos (AR, AutoRegressive) y modelos de medias móviles (MA, Moving Averages).Además de esto, incorporan la diferenciación de series temporales dentro del propio modelo.

En un modelo AR; a diferencia de un modelo de regresión normal, donde se utiliza una combinación lineal de características para predecir una

variable; se utiliza una combinación lineal de los valores anteriores de dicha variable. Un modelo AR de order p, al que nos referimos como $AR(p)$ se define de la siguiente forma:

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + ... + \phi_p y_{t-p} + \varepsilon_t$$

Donde ε_t es ruido blanco, el ruido blanco es un tipo de serie temporal donde su varianza es siempre la misma y cada valor no tiene correlación con el resto de valores de la serie. Los valores $y_{t-1}, ..., y_{t-p}$ son los p valores anteriores en la serie que se utilizan para calcular el valor actual. Los valores $\phi_1, ..., \phi_p$ son parámetros que el modelo modifica para ajustarse a la serie; c es un término independiente.

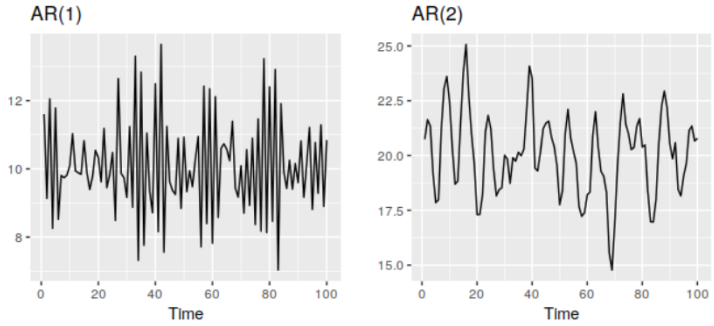


Figura 2.1: Diferentes modelos AR.

Un modelo MA, a diferencia de un modelo AR utiliza los errores de predicciones anteriores para generar un modelo de regresión. Un modelo MA con grado q, al que llamaremos $MA(q)$ se define de la siguiente manera:

$$y_t = c + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + ... + \theta_q \varepsilon_{t-q}$$

Donde ε_t es ruido blanco. Al igual que en el modelo anterior, $\theta_1, ..., \theta_q$ son parámetros que el modelo entrena para ajustarse a una serie temporal.

El último concepto que añade ARIMA es la diferenciación. La diferenciación es un proceso que se aplica a series temporales que no son estacionarias. Una serie temporal estacionaria es aquella cuya media y varianza se mantienen constantes. La diferenciación se trata del calculo de la diferencia entre observaciones consecutivas de la serie; por ejemplo una diferenciación de grado 1 en una serie se puede definir como $y'_t = y_t - y_{t-1}$ y de grado 2 como $y''_t = y'_t - y'_{t-1} = y_t - 2y_{t-1} + y_{t-2}$.

Combinando la diferenciación junto con los modelos AR y MA en un único modelo, se obtiene un modelo ARIMA. Un modelo ARIMA puede

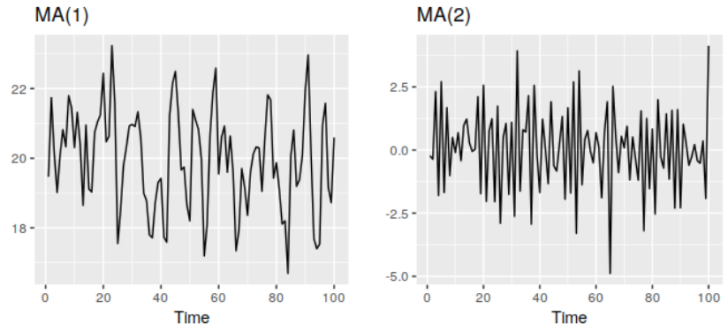


Figura 2.2: Diferentes modelos MA.

expresarse de la siguiente manera.

$$y'_t = c + \phi_1 y'_{t-1} + ... + \phi_p y'_{t-p} + \theta_1 \varepsilon_{t-1} + ... + \theta_q \varepsilon_{t-q} + \varepsilon_t$$

Donde y'_t es la serie diferenciada. De forma general, un modelo ARIMA se representa como $ARIMA(p, d, q)$ donde p representa el grado del modelo AR, d representa el grado de la diferenciación a la serie y q representa el grado de del modelo MA. Si p o q son 0, entonces nos encontramos con un modelo MA o AR, dependiendo cual es 0.

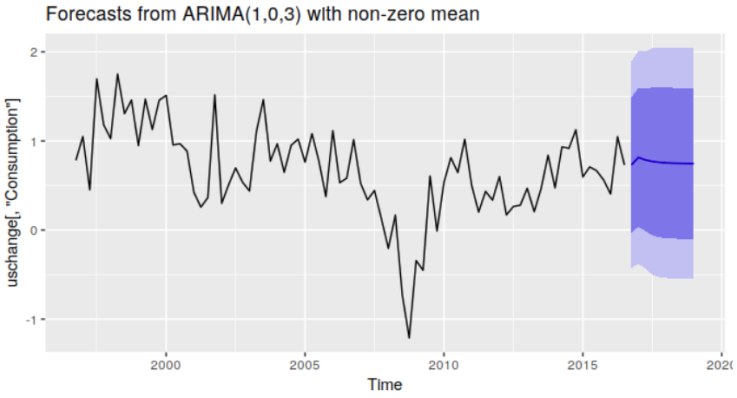


Figura 2.3: Ejemplo predicción con modelo ARIMA.

2.1.2. Series temporales de clasificación

En el apartado anterior se ha descrito modelos utilizados cuando se estudian series temporales de regresión; en este aparatado se verán diferentes modelos utilizados cuando se estudian series temporales de clasificación.

Las series temporales de clasificación tienen propiedades que hacen que ciertos aspectos del preprocesamiento haya que abarcarlos de forma diferente a diferencia de otro tipo de problemas; sin embargo, los métodos que se utilizan para su clasificación son los mismo que para cualquier otro tipo de problema de clasificación.

Los métodos más utilizados, a parte de las redes neuronales que no se discutirán en este apartado; son SVM, RandomForest y Boosting. A continuación, se describirá el funcionamiento de estos algoritmos.

SVM

El SVM (Support Vector Machine, Máquinas de Soporte Vectorial en español) fueron introducidas en 1992. Las SVM son un caso particular de "Kernel Machines" (KM). Este tipo de algoritmos utilizan el producto escalar de los datos de entrada, de esta forma, el producto escalar se usa como una forma de establecer una similitud entre los elementos.

Con este producto escalar, se puede definir un hiperplano como $< x, w > +b = 0$; modificando el valor de w, se puede adaptar dicho hiperplano para separar mejor los datos, un ejemplo clásico de esto es el perceptron.

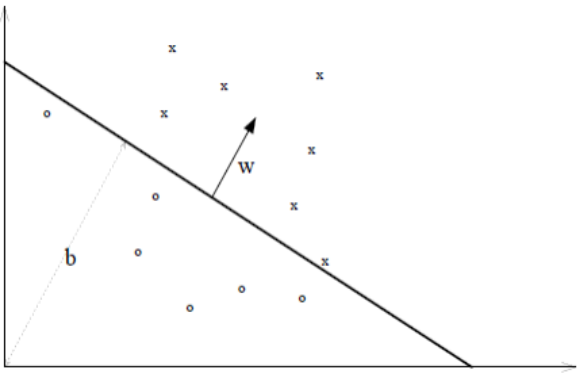


Figura 2.4: Ejemplo de hiperplano.

Para obtener un buen valor de w, se puede utilizar una combinación lineal de los datos de entrada junto con un coeficiente que refleje su dificultad, $w = \sum \alpha_i y_i x_i$. De esta forma, se puede representar ese hiperplano como $\sum \alpha_i y_i < x_i, x > +b$; para entrenar y ajustar este hiperplano simplemente se necesita modificar α_i, si $f(x) = \sum \alpha_i y_i < x_i, x > +b <= 0$ entonces $\alpha_i = \alpha_i + \eta$. Gracias a esta nueva representación, la función de decisión (hiperplano) solo necesita los datos de entrada.

La representación dual es la primera características de los SVM. El problema es que hasta ahora se han utilizado una separación lineal de los datos y en muchos casos los datos no son linealmente separables o existe ruido. La solución que da SVM para este problema es trasladar los datos, que no son linealmente separables a otro espacio donde sí lo sean, para ello utiliza los kernels.

Un kernel es una función que devuelve el valor del producto escalar entre los datos llevados a otro espacio. Se puede representar de la siguiente forma.

$$K(x_1, x_2) =< \phi(x_1), \phi(x_2) >$$

Donde x_1, x_2 son datos en el espacio original y ϕ representa la transformación para llevar un dato a otro espacio. Gracias a los kernels no es necesario saber la transformación ϕ, además, a partir de este momento en la función del hiperplano se puede sustituir el producto escalar por la función kernel.

Existen diferentes tipos de kernels, como por ejemplo el polinomial o el radial. A continuación se muestra un ejemplo de proyección en el espacio.

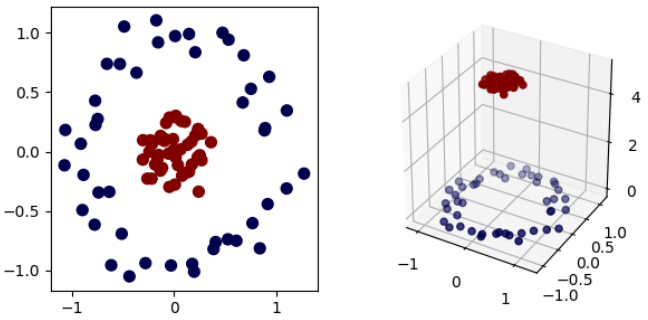


Figura 2.5: Ejemplo de proyección.

Por último, la última características de las SVM es que tiene la capacidad de maximizar el margen, es decir, de entre todos los hiperplanos que separan las dos clases, SVM se queda con aquel que maximiza la distancia a las dos clases, de esta forma se minimiza el riesgo de sobreaprendizaje.

RandomForest

RandomForest es uno de los métodos clásicos que forma parte del estado del arte para algoritmos de clasificación. RandomForest se trata de un multiclasificador, es decir, es un modelo formado por varios modelos más simples; en el caso de RandomForest se utilizan árboles de clasificación.

Los árboles de clasificación son un algorimto clásico de clasificación. El proceso que realiza un árbol de clasificación es dividir el espacio del problema hasta que llega decisión. El proceso que sigue para generar el árbol es el siguiente.

Primero se busca el atributo que mejor separe las clases sobre el conjunto de datos entero (nodo raíz). Segundo, se divide el conjunto de datos según el atributo elegido y se elimina este. Este proceso se repite en cada nodo hasta que todos los elementos en ese elemento sea de la misma clase. El proceso se repite hasta que todas las ramas del árbol llegan a una solución.

Para decidir que atributos son los que mejor separan las clase existe diferentes medidas como por ejemplo el índice GINI, Entropía, Ganancia de información o el Ratio de Gananacia. Las medidas GINI y Entropía miden interés en un atributo a partir de la frecuencia relativa entre clases para la partición de dicho atributo, la Ganancia mide la reducción en la Entropía al realizar una partición, el Ratio de Ganancia mide la Ganancia dividido entre el número de particiones que se realizan al particionar con ese atributo.

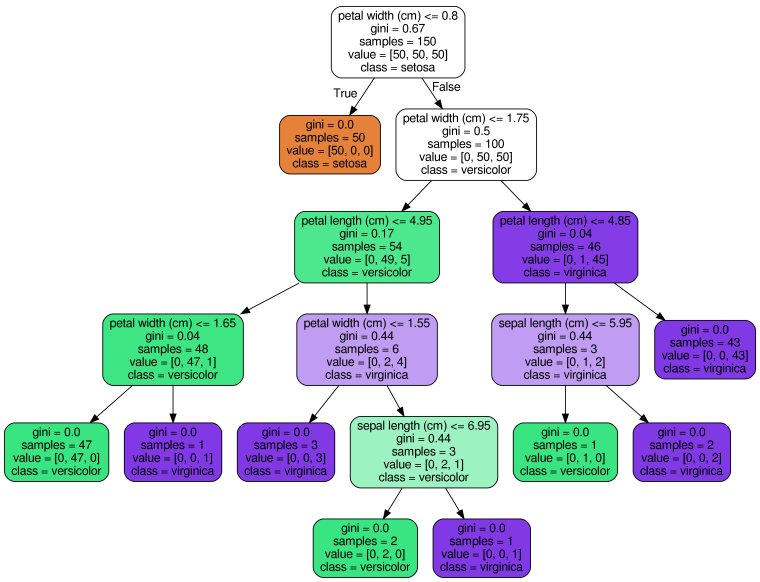


Figura 2.6: Ejemplo de árbol de clasificación.

RandomForest crea un conjunto de árboles de clasificación para realizar su decisión; una vez creados los árboles, se decide la clase final mediante voto. Para generar los árboles se utiliza *Bagging*. Para cada uno de los árboles, se eligen m atributos y un subconjunto de instancias del conjunto de datos, de esta forma se obtiene modelos más simples que pueden aprender mejor ciertos atributos que si se creara un solo árbol de decisión.

Boosting

Boosting, al igual que RandomForest se trata de un multiclasificador, aunque su enfoque es diferente. Boosting puede utilizarse tanto con árboles de decisión como con otros algoritmos.

A diferencia de RandomForest, donde cada árbol que se genera es independiente, en Boosting cada árbol o clasificador que se genera utiliza información de los árboles que se han creado anteriormente.

El algorimto más conocido de Boosting es AdaBoost. AdaBoost sigue el siguiente proceso para crear N clasificadores.

- Primero, elige un subconjunto de instancias del conjunto de datos según su peso y entrena un clasificador simple.

- Segundo, prueba el clasificador sobre el conjunto de datos completo y aumenta el peso de aquellos datos mal clasificados y reduce el de aquellos bien clasificados.
- Por último, asocia al clasificador entrenado un peso/importancia basada en el error cometido.

Una vez creados los N clasificadores, la clase final se decide mediante una combinación lineal de la salida de cada clasificador y su importancia.

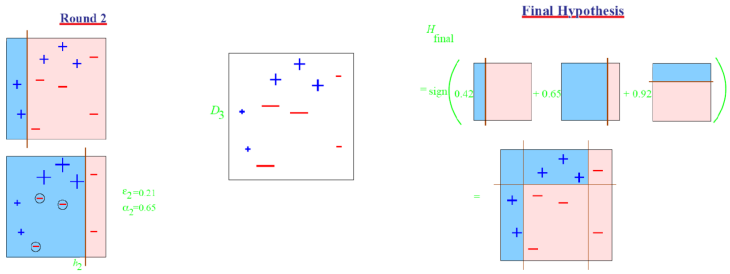


Figura 2.7: Ejemplo funcionamiento AdaBoost.

Para este estudio, el algoritmo que se utilizará es XGBoost; este es un algoritmo de Boosting, específicamente Gradient Boosted Trees. Este algoritmo utiliza el Boosting para generar nuevos árboles que puedan predecir la clase de los errores que han cometido árboles anteriores. Para minimizar la pérdida al añadir nuevos modelos se utiliza gradiente descendiente.

XGBoost añade también el concepto de regularización a los Gradient Boosted Trees. La que es un mecanismo para evitar el sobreaprendizaje, para conseguir esto se mide la complejidad del modelo, de forma que se premian modelos más sencillos. XGBoost, a parte de optimizar la pérdida para conseguir mejores modelos, optimiza la regularización para obtener modelos más sencillos. Esto hace que pueda producir mejores resultados mejores que los Gradient Boosted Trees, ya que modelos más sencillos suelen generalizar mejor y por lo tanto sufrir menos sobreaprendizaje y siendo más estables.

2.2. LSTM

Las LSTM (Long Short-Term Memory) son un tipo de red neuronal recurrente; este tipo de red es capaz de procesar de procesar secuencias de datos, como por ejemplo vídeos o frases. Este tipo de red neuronal se inventaron para ser usadas en problemas donde la información que se procesa es dependiente de información anteriormente procesada, y esta relación debe ser recordada hasta que deje de ser útil. Actualmente las redes con LSTM se utilizan en diversas tareas, como por ejemplo reconocimiento del habla, traducción, etc.

Una unidad de LSTM o una neurona de LSTM tiene la siguiente estructura:

- Puerta de entrada.
- Puerta de salida.
- Puerta de olvido.
- Célula.

La célula es la encargada de almacenar información; el valor de la célula depende de los valores de las puertas de entrada, salida y olvido. Además, los valores de dichas puertas van cambiando en cada instante de tiempo. En la siguiente imagen se puede ver una representación de las estructura de una LSTM.

La puerta de olvido tiene como función olvidar información anterior dependiendo de la información actual, dicha función podría expresarse de la siguiente forma.

$$f^{(t)} = \sigma(W_f[h^{(t-1)}, x(t)] + b_f)$$

Donde:

- $x^{(t)}$: entrada de la puerta en el momento t.
- $h^{(t-1)}$: valor de la salida de la célula en el instante anterior.
- W_f: matriz de pesos de la puerta de olvido.
- b_f: sesgo de la puerta de olvido.
- σ: función de activación (valores entre 0 y 1) de la puerta de olvido.

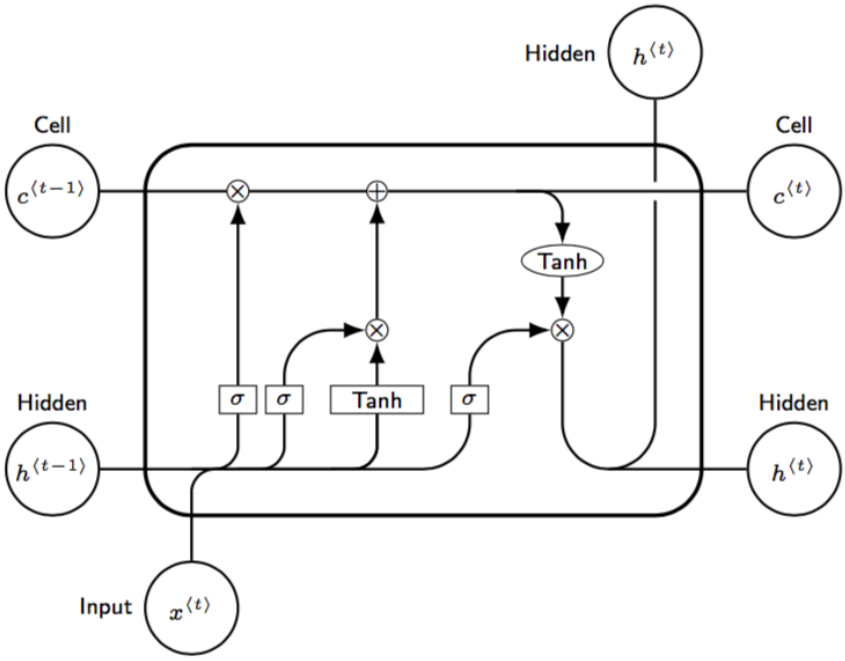


Figura 2.8: Estructura de una LSTM.

La puerta de entrada tiene como función aprender los parámetros de entrada para su composición con el estado de la LSTM. PAra ello se debe calcular el estado de la LSTM y seleccionar la información que se utilizará para actualizar el estado. Esto se puede expresar mediante las siguientes funciones:

$$i^{(t)} = \sigma(W_i[h^{(t-1)}, x^{(t)}] + b_i)$$
$$s'^{(t)} = \tanh(W_s[h^(t-1), x^{(t)}] + b_s)$$

Donde:

- $s'^{(t)}$:estado local de la LSTM, sin tener en cuenta el estado anterior.
- W_i: matriz de pesos de la puerta de entrada.
- W_s: matriz de pesos del estado de la LSTM.
- b_i: sesgo de la puerta de entrada.
- b_s: sesgo del estado.

Tras esto, se calcula el nuevo estado de la LSTM de la siguiente forma:

$$s^{(t)} = f^{(t)} s^{(t-1)} + i^{(t)} s'^{(t)}$$

De esta forma, el estado se forma con una parte del estado anterior, la que no se olvida; y con la composición de la entrada con el estado local ($s'^{(t)}$).

La puerta de salida es la encargada de aprender los parámetros de salida, esto puede expresarse mediante la siguiente función:

$$o^{(t)} = \sigma(W_o[h^{(t-1)}, x^{(t)}] + b_o)$$

Por último, la salida se calcula como:

$$h^{(t)} = o^{(t)} \tanh(s^{(t)})$$

La función para calcular la salida puede ser otra, como ReLu; las funciones de activación de las puertas también pueden ser otras.

Como puede verse, las LSTM son capaces de guardar información relevante durante un largo periodo de tiempo y eliminar la información poco importante; por ello, son muy utilizadas en problemas con series temporales, ya que la información en un momento t es dependiente de la información que había en instantes anteriores y las LSTMs son capaces de detectar dicha dependencia y aprenderla.

Las LSTM son un tipo de red recurrente, por lo que se pueden construir arquitecturas basadas en una LSTM que procesa la secuencia de izquierda a derecha y otra de derecha a izquierda; a estas estructuras se les llama LSTM bidireccionales, en la siguiente imagen se muestra un ejemplo de esta estructura.

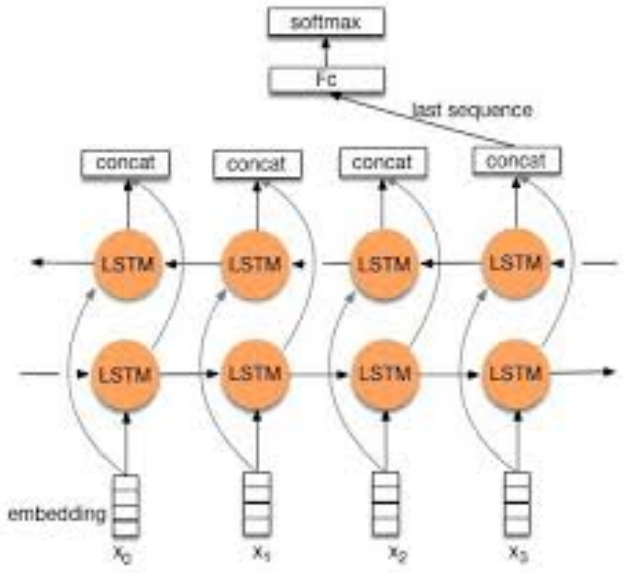
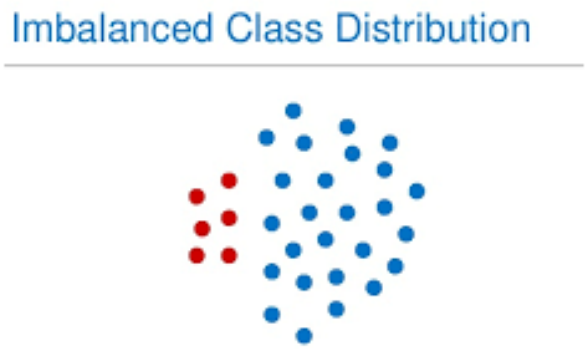


Figura 2.9: Ejemplo de estructura de LSTM bidireccional.

Gracias a este tipo de estructuras pueden resolverse problemas donde la importancia/sentido de un dato no depende de datos anteriores sino por información posterior; por ejemplo, el significado de una palabra puede ser diferente dependiendo de las palabras siguientes que haya en una secuencia y no en las anteriores.

2.3. Clasificación con clases no balanceadas

El problema de desbalanceo de clases es un tipo de problema que aparece en problemas de clasificación. el desbalanceo de clases ocurre cuando el número de elementos de una clase es mucho mayor que el número de elementos de la otra clase; a esta clase se le llama clase minoritaria. En la siguiente imagen se puede ver un ejemplo en 2D de un problema de desbalanceo de clases.



El principal problema que tiene el desbalanceo de clases es que afecta a la capacidad de aprendizaje de la mayoría de los modelos de minería de datos actuales, exceptuando algunos como los árboles de decisión, aunque también trabajan mejor si no hay desbalanceo. Además, si el desbalanceo es muy grande es posible que los modelos no aprendan directamente la clase minoritaria.

Otro problema es que afecta a las medidas utilizadas normalmente en problemas de clasificación como es el Accuracy; esta medida representa el número de elementos bien clasificados sobre el total de elementos; si por ejemplo la clase minoritaria representa el 0.1 % de los datos y un clasificador predijera todos los datos como elementos de la clase mayoritaria, su Accuracy sería del 99.9 %; por ello, se debe utilizar otras medidas. Las medidas usuales son Precision, Recall, AUC, G-Mean, F1-Score, etc; todas estas medidas tienen en cuenta la clase minoritaria de forma que si ocurre lo anterior su valor sea bajo.

Como solución al problema del desbalanceo existen dos propuestas, una basada en modificación de algoritmos y otra basada en modificación del

conjunto de datos.

2.3.1. Técnicas basadas en modificación del algoritmo

Este enfoque se centra en modificar clasificadores ya existentes para aliviar el sesgo hacia la clase mayoritaria en vez de alterar el conjunto de datos. Esto requiere un buen conocimiento interno del modelo que se quiere modificar y saber las razones por las cuales falla al identificar la clase minoritaria.

Modificar un modelo reduce su flexibilidad y por lo tanto lo hace válido para un número menor de problemas, pero a cambio ofrece una mayor especialización para el tipo de problema que se modifica.

Algunos ejemplos de modelos modificados son árboles de decisión que utilizan la distancia de Hellinger para la separación de nodos, utilizar un método diferente de cálculo de pertenencia de clases para KNN, por ejemplo con pesos en vez de distancias, uso de kernels específicos con SVM, etc.

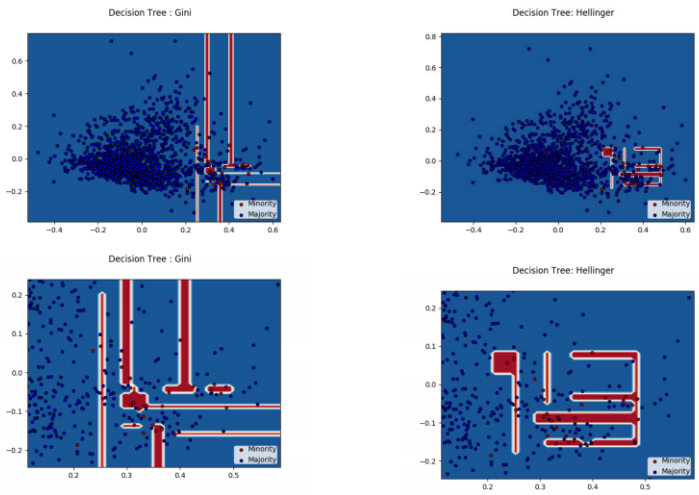
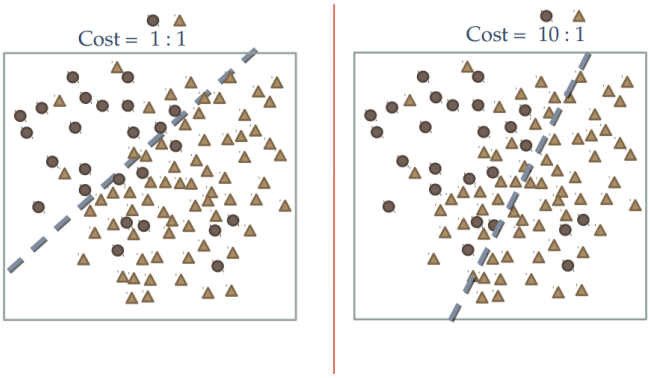


Figura 2.10: Ejemplo comportamiento algoritmo.

Otra posible solución es modificar el peso asociado a clasificar mal un elemento, dando un peso mayor a clasificar mal un elemento de la clase minoritaria que de la mayoritaria, de forma que cuando se entrena el modelo intenta minimizarse el coste. El peso asociado a cada fallo se almacena en la matriz de costes, dicha matriz puede ser calculada con diferentes heurísticas o aportada por un experto en el problema que se plantee. El la siguiente imagen se muestra un ejemplo del uso de la matriz de costes para un clasificador simple.



Existen dos formas de usar la matriz de costes, la primera es integrar el uso de la matriz dentro del algoritmo, lo que significa que hay modificarlo para que use dicha matriz; la segunda forma es preprocesando los datos de entrada asignándole un peso o asignando a cada elemento la clase que se cree que tendrá menor coste (para ello se hace uso del Teorema de Bayes).

2.3.2. Técnicas basadas en la modificación del conjunto de datos

La idea de este tipo de técnica es manipular la distribución de los datos con los que entrena el clasificador, para ello se añaden o suprimen elementos del conjunto de datos. Cuando se añaden elementos, se llama "oversampling" y cuando eliminan "undersampling"; dependiendo del problem se puede usar una de estas técnicas o ambas.

Un ejemplo clásico de algoritmo de undersampling es Tomek Links; este algoritmo elimina elementos de la clase mayoritaria que sean fronterizos con elementos de la clase minoritaria; para ello calcula las parejas de elementos de clases diferentes que estén a la distancia mínima entre ellos, es decir, que no haya ningún otro elemento dentro del conjunto de datos que su distancia hacia ese elemento sea menor; y elimina aquellos que sean de la clase mayoritaria. En la siguiente imagen se muestra un ejemplo del uso de este algoritmo.

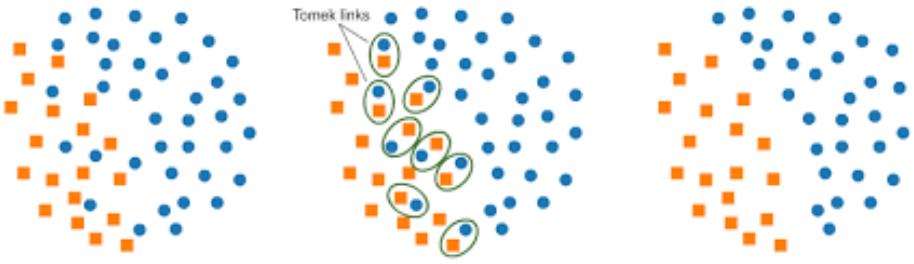


Figura 2.11: Ejemplo funcionamiento algoritmo Tomek-Links.

Otros métodos de undersampling son OSS, CNN y combinaciones de ellos. El problema del undersampling es que al eliminar elementos de la clase mayoritaria hace que cierta información se pierda, y dicha información puede ser importante al evaluar.

Un algoritmo clásico de oversampling es SMOTE, este algoritmo crea nuevas instancias usando una combinación de K instancias de la clase minoritaria que sean vecinas. En la siguiente imagen se muestra un ejemplo del

funcionamiento del SMOTE.

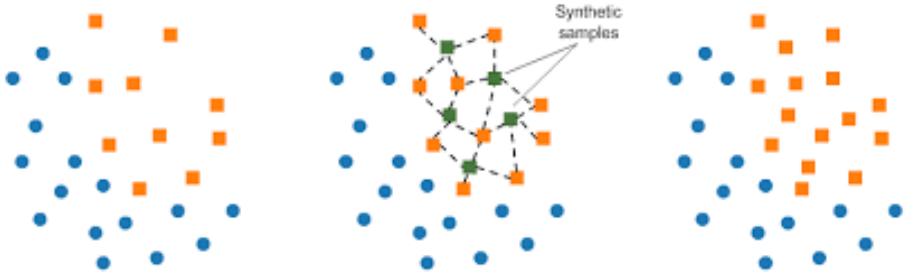


Figura 2.12: Ejemplo funcionamiento algoritmo SMOTE.

Otros algoritmos de oversampling son modificaciones de SMOTE, como por ejemplo Borderline-SMOTE (centrado en la generación de instancias en la frontera), ADASYN, etc.

El problema del oversampling es que pueden generalizar demasiado la clase minoritaria y provocar ruido en el conjunto de datos.

2.3.3. Otros enfoques

Una última opción para paliar el desbalanceo sin implicar modificaciones de los algoritmos o del subconjunto de datos, esta es el uso de modelos ensemble; los modelos ensemble son aquellos que están formados por varios modelos más simples, para ello, los modelos simples se entrenan con un subconjunto de los datos, de forma que en esos subconjuntos el desbalanceo no tiene porque ser tan grande. Para elegir una clase, los modelos ensemble pueden combinar los resultados de los modelos simples (para problemas de regresión) o mediante voto. Un ejemplo de modelo ensemble es RandomForest.

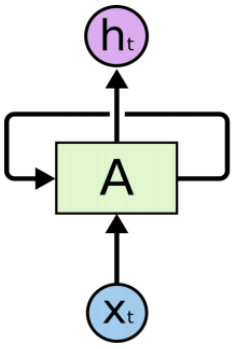
Capítulo 3

LSTMs para clasificación de series temporales

En este capítulo se describirán el uso de las LSTM para problemas de series temporales de clasificación, así como una pequeña descripción de las LSTM y las series temporales de clasificación.

3.1. Descripción

Las LSTM son un tipo de red recurrente, esto quiere decir que tiene la capacidad de utilizar como entrada información que ya ha sido procesada; en el caso de las LSTM, dicha información es de gran importancia ya que es utilizada para calcular la salida de la LSTM a parte de la nueva información que se haya obtenido.



Las series temporales de clasificación son un caso especial de serie temporal, ya que el objetivo tras estas series no es obtener una predicción sobre el comportamiento futuro de la serie sino predecir una clase. Sobre este ti-

po de series temporales se pueden distinguir dos tipos, aquellos donde la serie entera corresponde con una clase y otros donde cada instante de la serie corresponde con una clase. El primer caso se podría expresar de la siguiente forma $S \rightarrow Y$ donde S es una serie cualquiera e Y es la clase asociada a esta serie. El segundo caso se podría expresar de la siguiente forma $S = (S_0, S_1, ..., S_m) \rightarrow (Y_0, Y_1, ..., Y_n)$ donde S es una serie temporal de clasificación de longitud n, S_i representa la serie en un momento determinado e Y_i la clase asociada a dicho momento.

3.2. Arquitectura

Para procesar series temporales de clasificación con LSTM se debe utilizar una estructura como la siguiente; una primera capa formadas por LSTM y una segunda capa para la predicción de la clase, esta capa no tiene porque ser LSTM.

La primera capa de esta red es la encargada de procesar la información, las LSTM procesan una serie temporal o un trozo de una serie, dependiendo del tipo de problema que se esté procesando, en cada momento; una vez procesada la información la salida de cada una de las LSTM se pasa a la segunda capa que es la encargada de predecir la clase correspondiente. Para el cálculo de la clase se utiliza una función de activación, para el caso de problemas de clasificación binaria se puede utilizar una función sigmoide, si se trata de un problema de clasificación múltiple se puede añadir tantas neuronas como clases diferentes hay o utilizar la función de activación softmax, que tiene un comportamiento muy parecido a la función sigmoide pero está adaptada a más de dos clases.

Una vez se ha calcula la clase, al igual que cualquier otra red neuronal, se comprueba si esta es correcta y se recalculan los pesos de las neuronas de la red para mejorar el rendimiento de esta; al tratarse de un problema de clasificación, las métricas de rendimiento deben ser las usuales para clasificación como por ejemplo Accuracy.

La red descrita arriba es un ejemplo de la arquitectura más simple necesaria para afrontar este tipo de problemas. A partir de esta red pueden crearse redes más complejas y que en problemas específicos pueden tener un mejor rendimiento, como por ejemplo añadiendo más capas intermedias o combinando esta red con una red convolucional.

Para este trabajo se utilizará la red básica descrita, ya que se utilizarán problemas diferentes para los cuales unas estructuras pueden ser más útiles que otras.

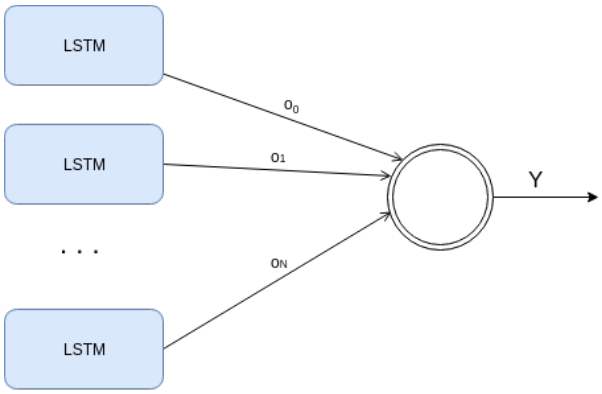


Figura 3.1: Estructura básica de una red de LSTM para clasificación

3.3. Justificación

Las series temporales son un tipo de dato que tiene dependencia temporal, es decir, el valor de una serie en un momento determinado está relacionado con el valor de la serie en momentos anteriores. Como se ha dicho al principio del capítulo, las LSTM son un tipo de red que utilizan la información anterior para procesar nueva información, por lo que lo hace adecuado para este tipo de problema.

Además, las redes LSTM actualmente son un tipo de red muy utilizada en problemas complejos como la detección del habla, traducción, reconocimiento de palabras escritas a mano o incluso son utilizadas para la creación de IA capaces de jugar a videojuegos mejor que una persona.

A parte de las LSTM, existen otro tipo de neuronas recurrentes como son el caso de las RNN; fueron la primera estructura recurrente para redes neuronales, tienen el inconveniente de no ser capaces de recordar información durante mucho tiempo; o las GRU, son también un tipo de neurona muy utilizada, más sencilla que las LSTM ya que tiene un número de puertas lógicas menor y más adecuada dependiendo del problema, sin embargo, al ser más sencilla no es tiene tanta potencia como las LSTM.

Por todo esto, las LSTM son una estructura válida para este trabajo y serán las que se utilicen para procesar los diferentes problemas de clasificación que se verán más adelante.

Capítulo 4

Oversampling para clasificación de series temporales

En este capítulo se describirán diferentes algoritmos de oversampling que se utilizarán más adelante en el estudio. Con dichos algoritmos de oversampling se busca mejorar el rendimiento de los modelos que se probarán en el estudio. Los algoritmos que a continuación se van a describir son SMOTE, MWMOTE y ADASYN.

4.1. SMOTE

SMOTE (Synthetic Minority Oversampling Technique) es uno de los algoritmos de oversampling más utilizados a día de hoy. Este algortimo fue inventado en 2002 y actualmente se considera como uno de los algoritmos en el estado del arte del oversampling.

Para balancear los datos de las diferentes clases, SMOTE propone generar instancias de la clase minoritaria de forma sintética. Para ello, SMOTE utiliza las instancias de la clase minoritaria para generar nuevas instancias de dicha clase con características parecidas a las ya existentes.

Para conseguir dicho objetivo SMOTE sigue el siguiente proceso, elige una instancia de la clase minoritaria cualquiera, tras esto se buscan sus K vecinos más cercanos que sean también de la clase minoritaria; dicho parámetro K se puede cambiar y afecta a como son las instancias generadas sintéticamente; si por ejemplo su valor fuera $K = 1$, las instancias se generan a partir de dos instancias de la clase minoritaria, esto hace que las características de dichas instancias nuevas sean muy parecidas a las de las instancias que se han utilizado; en cambio si K tiene un valor muy grande

entonces las instancias generadas tienen menos parecido a las instancias con las que se han generado.

Una vez se han obtenido el número de vecinos elegido, la nueva instancia se genera como una combinación de las instancias que se utilizan, por ejemplo si $K = 1$, la nueva instancia se generaría como la media de los valores de dichas instancias.

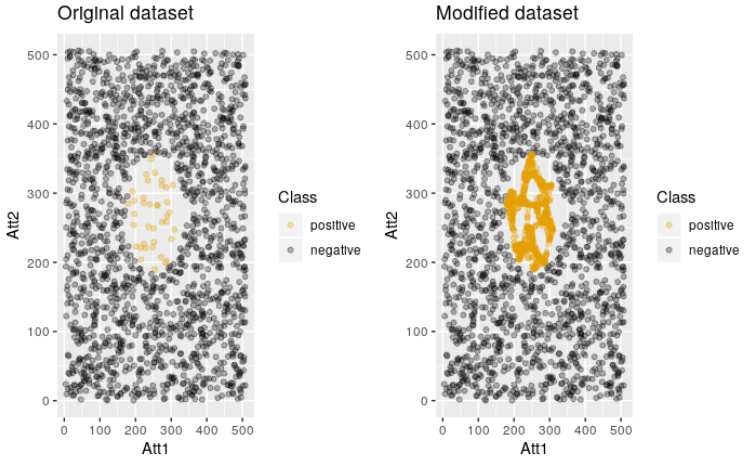


Figura 4.1: Ejemplo de generación de instancias con SMOTE.

Uno de los principales problemas de SMOTE es que no es bueno utilizarlo cuando las clases del conjunto de datos están mezcladas, es decir, dentro de los datos de la clase mayoritaria puede haber un pequeño conjunto de la clase minoritaria (la cual podría considerarse ruido); si utilizaramos SMOTE, el número de instancias de ese conjunto sería mayor y esto afectaría al rendimiento de un modelo. Por ello, antes de utilizar este método es bueno realizar otro tipo de preprocesamientos como limpieza de ruido, imputación de valores perdidos, etc...

4.2. MWMOTE

MWMOTE (Mayority Weighted Minority Oversampling Technique) se trata de una modificación del algoritmo SMOTE para evitar de instancias ruidosas.

El objetivo de este algoritmo es mejorar la forma de seleccionar instancias y la forma de generar las instancias sintéticas. Para ello este algoritmo cuenta con tres fases.

En la primera fase se identifican los ejemplos de la clase minoritaria, a este conjunto lo podemos llamar S_{min}; que son más difíciles de aprender, para ello se genera un conjunto con dichos ejemplos al que llamaremos S_{imin}. Para generar este conjunto el algoritmo elimina todas aquellas instancias de S_{min} que no tenga ninguna instancia de S_{min} en su vecindario; para calcular el vecindario se utiliza el mismo proceso que para SMOTE. Una vez eliminados, se calculan aquellas instancias que son fronterizas con un conjunto de datos de la clase mayoritaria, estas instancias son las que forman S_{imin}.

En la segunda fase, cada uno de los ejemplos de S_{imin} se le agrega un peso dependiendo de la importancia de dicho ejemplo dentro de el conjunto de datos de S_{min}. Para ello se calcula la cantidad de información que se obtiene de cada uno de los puntos, esta medida se calcula teniendo en cuenta como de cercano está cada instancia al conjunto de datos de la clase mayoritaria. Por último, una vez calculado el peso de cada instancia se le asocia una probabilidad a dicho peso.

En la tercera fase, el algoritmo genera instancias sintéticas a partir de S_{imin} usando dicho peso y los añade al conjunto de datos de la clase minoritaria. Para generar dichas instancias, se detectan diferentes clústers dentro de S_{min} y se por cada instancia que se genera se elige un elemento del conjunto S_{imin} x, se elige otra instancia y de S_{min} que se encuentre en el mismo clúster que x y se genera una nueva instancia de la siguiente forma: $s = x + \alpha \times (y - x)$.

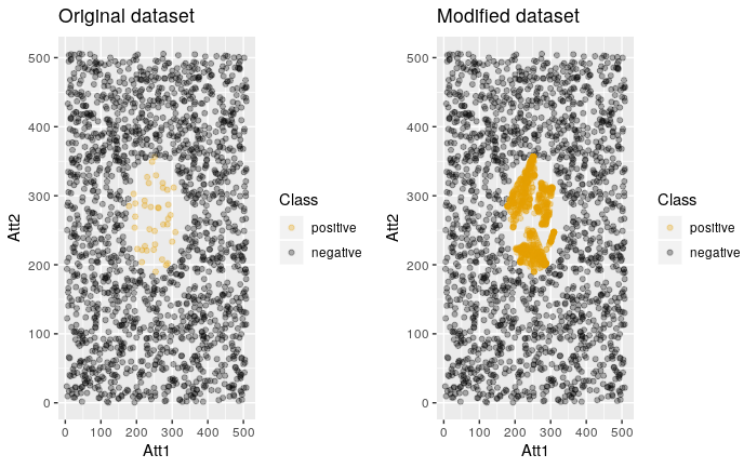


Figura 4.2: Ejemplo de generación de instancias con MWMOTE.

4.3. ADASYN

ADASYN (Adaptive Synthetic Sampling) se trata de un algoritmo motivado por SMOTE. Al igual que el algoritmo de la sección anterior, propone otra forma de generar instancias. El objetivo principal de este algoritmo es reducir el sesgo entre ambas clases mientras aprende de forma adaptada. el procedimiento que sigue este algoritmo es el siguiente.

Primero, calcula el ratio entre instancias de la clase minoritaria y mayoritaria; este ratio después se utiliza para calcular el número de instancias de la clase minoritaria a generar.

Tras esto, por cada instancia perteneciente a la clase minoritaria, se calcula su valor r_i, este valor indica la dominancia de ejemplos de la clase mayoritaria en el vecindario de esta instancia; a valores mayores de esta medida es más difícil aprender sobre dicha instancia. Una vez calculadas todas las medidas se normalizan.

Una vez calculadas todas las medidas r_i, se calcula el número de instancias a generar en el vecindario de dicha instancia de la siguiente manera: $G_i = Gr_i$; donde G es el número de instancias a generar que se calculó al principio.

Por último, cada una de las instancias en un vecindario i se genera de la siguiente forma: se selecciona una instancia del vecindario, x_i; después se selecciona de forma aleatoria otra instancia vecina a la instancia x_i, la llamaremos x_{zi}; por último se calcula un número aleatorio entre 0 y 1 y se calcula la nueva instancia como la siguiente fórmula: $s_i = x_i + (x_{zi} - x_i)\lambda$.

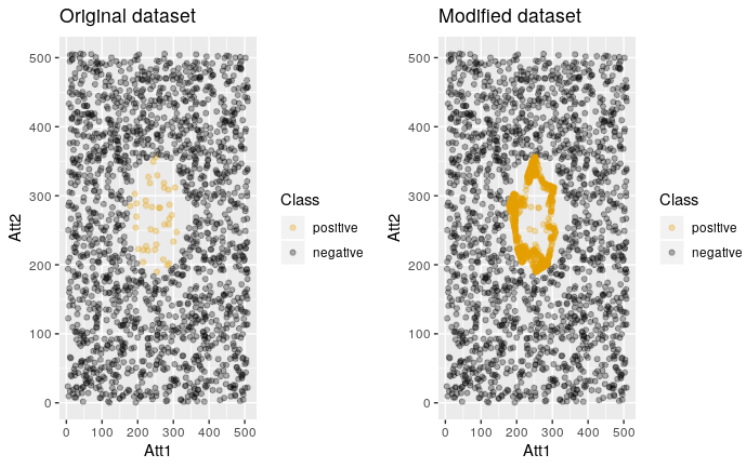


Figura 4.3: Ejemplo de generación de instancias con ADASYN.

Uno de los problemas de ADASYN es que puede generar instancias en vecindarios donde el número de instancias de la clase minoritaria es muy reducida, produciendo ruido.

Capítulo 5

Tratamiento de series temporales de clasificación multivariadas

En este capítulo se describirán las características principales de las series temporales de clasificación multivariadas, así como diferentes métodos de preprocesamiento que se han utilizado durante el estudio para limpiar dichas series he intentar mejorar los resultados.

Las series temporales multivariadas de clasificación, al igual que cualquier otro conjunto de datos, deben ser preprocesadas para ofrecer un mejor rendimiento a la hora de predecir.

Podemos definir una serie temporal multivariada como S, la cual está formada por un conjunto de series temporales univariadas, es decir $S = \{S_1, S_2, ..., S_n\}$, cada una de estas series univariadas tienen sus propias características. Las series temporales multivariadas de clasificación son caso de serie temporal multivariada para las cuales además de las series univariadas existen etiquetas en el conjunto de datos. Para las series temporales multivariadas de clasificación existen dos tipos.

El primer tipo de serie temporal multivariada de clasificación es aquella donde cada una de las series temporales univariadas tienen asociadas una clase, es decir $S = \{(S_1, y_1), (S_2, y_2), ..., (S_n, y_n)\}$, donde cada S_x es una serie univariada e y_x es su clase asociada. Un ejemplo de serie temporal de este tipo puede ser las formadas por imágenes, cada una de las imágenes tiene una clase asociada.

El segundo tipo de serie temporal multivariada es aquella donde en cada momento el conjunto de series univariadas tienen asociada un clase, es decir

$S = \{(S_{11}, S_{12}, ..., S_{1m}, y_1), (S_{21}, S_{22}, ..., S_{2m}, y_2)\}, ..., (S_{n1}, S_{n2}, ..., S_{nm}, y_n)$, donde S_{xt} es el valor asociado a la serie univariada x en un momento t e y_x es la clase asociada a dicho momento. Un ejemplo de este tipo de serie temporal puede ser predecir el movimiento de una persona a partir de los datos obtenidos de diferentes sensores.

A efectos prácticos, esto significa que dependiendo del tipo de series multivariada que sea las series univariadas se encuentran como filas (primer tipo) o como columnas (segundo tipo) y se debe tener en cuenta a la hora de realizar preprocesamiento.

A diferencia del tratamiento normal que se le puede hacer a un conjunto de datos aquí se tiene que tener en cuenta que las series temporales multivariadas son un conjunto de series temporales que juntas forman la representación de un problema y hacen que sea capaz de aprender una solución. Una sola serie temporal no es suficiente para poder aprender sobre el problema que se estudie pero sí puede tener características totalmente diferentes al resto de series temporales; por ello, el preprocesamiento se debe hacer de forma individual.

Un paso importante en el preprocesamiento si se están utilizando redes neuronales o cualquier otro modelo que se vea afectado por variaciones en la escala de diferentes características (o serie temporal en este caso) es la normalización de valores de las series.

Otro tipo de tratamiento necesario para series temporales es analizar valores perdidos en la serie temporales es analizar valores perdidos en la serie. Si existen valores perdidos hay dos posibilidades; la primera es eliminar dichos valores perdidos, esta opción no es muy recomendable si la serie tiene muy pocos datos, la otra opción es imputar los valores perdidos. Para imputar dichos valores, se puede utilizar biblioteca específicas para series temporales o utilizar métodos clásicos de imputación, aunque estos no tienen por qué tener en cuenta las particularidades de las series temporales.

Un buen método para imputar datos puede ser realizar un media móvil. La idea es calcular la media de los x momentos anteriores y x momentos siguientes, de esta forma se puede imputar cada dato con un valor cercano al resto de valores. La siguiente imagen muestra un ejemplo de media móvil. A parte de este método, se pueden utilizar métodos clásicos de predicción con series temporales como ARIMA, usando los datos anteriores al dato que se quiere imputar y realizar una predicción del valor de dicho dato con el modelo entrenado; sin embargo esto puede ser bastante costoso si hay bastantes datos perdidos ya que habría que generar un modelo para cada uno de ellos.

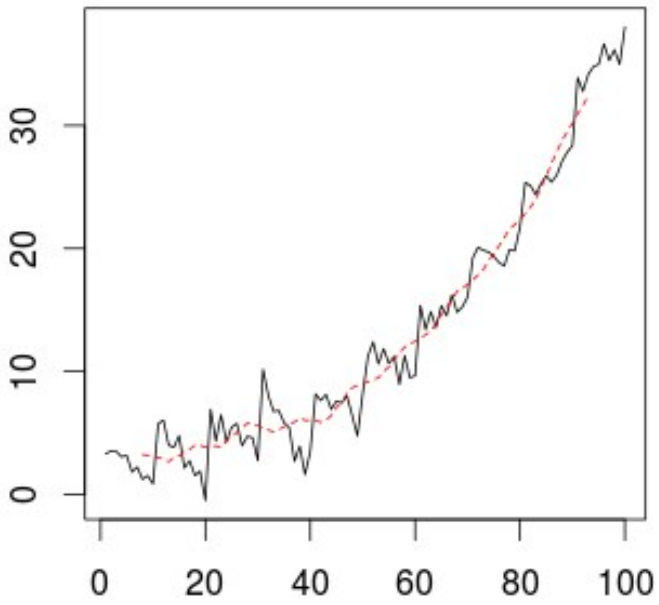


Figura 5.1: Ejemplo algoritmo medias móviles.

Además de esto, también se pueden analizar los outliers que existan en la serie; si los outliers que se encuentren no aportan ninguna información adicional es mejor modificar su valor. También se podrían borrar pero esto podría provocar una pérdida de información importante, por ello es mejor no hacerlo.

Para la detección de outliers se puede utilizar cualquier método de detección de outliers univariable. Un método sencillo es el cálculo de outliers mediante IQR (InterQuartile Range, rango intercuartílico en español), para ello se calcula el IQR, el Q_3 y Q_1 el de los datos para los cuales se quieren detectar outliers; se consideran outliers aquellos datos que estén fuera del rango definido por $[\ Q_1 - f * IQR, Q_3 + f * IQR]$, donde f es una constante.

Normalmente f suele tomar el valor 1.5 , pero para el caso de las series temporales este valor provoca que se detecten demasiados datos como outliers; por ello, en este estudio se ha utilizado $f = 3$ para asegurarse de que los outliers detectados sean realmente anomalías en los datos de la serie y no una pequeña variación. Una vez detectados los outliers, se pueden utilizar diferentes métodos para modificar dichos outliers, por ejemplo el método comentado anteriormente (medias móviles).

Otro proceso que se puede aplicar es crear ventanas de tiempo, es decir, en vez de utilizar solamente los datos en un momento dado, utilizar también los datos de n momentos anteriores para intentar mejorar el rendimiento de un clasificador. Elegir el tamaño de la ventana puede ser complicado ya que la ventana se aplica por igual a todas las series, por lo que requiere de experimentación para comprobar si hay una mejora en el rendimiento.

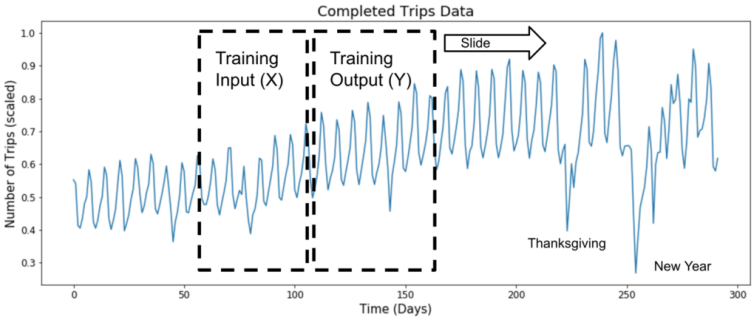


Figura 5.2: Ejemplo método de la ventana.

Por último, se puede aplicar una selección de características, o en este caso de series temporales; para ello se pueden utilizar algoritmos de selección de características como por ejemplo Boruta, este algoritmo utiliza un modelo RandomForest y comprueba qué características son las más utilizadas

Capítulo 6

Marco Experimental

En este capítulo se describirán los dataset utilizados en el estudio, además de las medidas usadas para valorar los resultados y los parámetros de la red neuronal usadas para predicción.

6.1. Datasets

Los datasets utilizados para el desarrollo de este estudio son:

- **Adiac**. El dataset Adiac contiene los datos de un estudio sobre algas unicelulares basados en imágenes, para pasarla a series temporales se usó el contorno de cada una de las algas en las imágenes. El dataset contiene 390 instancias para entrenamiento y 391 instancias para test, cada uno de los datos cuenta con 176 características y hay 37 clases diferentes. El objetivo es aprender a diferenciar correctamente cada una de las algas.
- **Swedish Leaf**. El dataset Swedish Leaf se trata de un conjunto de datos que contiene el contorno de hojas de árboles suecos. Este conjunto de datos contiene 500 imágenes para entrenamiento, 625 para test, 128 características y 15 clases distintas. El objetivo es aprender a diferenciar cada una de las hojas según su silueta.
- **EGG eye**. La información en este dataset fue obtenida con una medición EGG (medición de actividad eléctrica en el cerebro durante un tiempo) mediante un sensor. El objetivo de este dataset es diferenciar entre un ojo cerrado y otro abierto. El dataset contiene 14980 instancias y 15 atributos.
- **Face All**. El dataset contiene el contorno de 14 personas diferentes; el conjunto de entrenamiento cuenta con 560 imágenes y el conjunto

de test con 1690. El objetivo es diferenciar la silueta de cada una de las personas.

- **Ozone**. Este dataset contiene 2536 instancias, cada una con de ellas con 73 atributos. El objetivo de este dataset es diferenciar entre un día normal y un día con niveles extraños de ozono, para ello se utilizan información de sensores sobre velocidad del viento, temperatura máxima, etc... Este conjunto de datos contiene un gran desbalanceo entre ambas clases.

- **HAR**. El dataset HAR contiene información sobre el movimiento de personas utilizando diferentes sensores en un móvil (giróscopo, acelerómetro, etc...). El objetivo de este dataset es diferenciar entre 6 actividades diferentes. Cada una de las instancias de este dataset contiene 561 características.

- **Wafer**. Este dataset contiene 2 clases, tiene 1000 datos para entrenamiento y 6164 para test; cada uno de los datos contiene 152 características. Los datos de este dataset contiene los valores de datos obtenidos por sensores (cada una de las columnas) para un proceso de creación de planchas de semiconductores para procesadores. El objetivo es distinguir entre planchas defectuosas y no defectuosas. El conjunto de datos contiene un gran desbalanceo entre clases.

- **Yoga**. El dataset contiene 2 clases, tiene 300 datos para entrenamiento y 3000 para test; cada uno de los datos contiene 426 características.El objetivo en este dataset es diferenciar entre una mujer y hombre realizando yoga en imágenes. Los datos que contienen este dataset transformaciones de una imagen a una señal unidimensional, para ello tomaron el outline de cada imagen y se mide la distancia de cada punto con el centro.

Como se ha podido ver en la descripción, algunos de los conjuntos de datos son multiclase, por ello, dichos dataset se han transformado a problemas binarios escogiendo una clase, haciendo esa clase positiva y estableciendo el resto de clases como negativa; de esta forma también se crea desbalanceo entre clases.

6.2. Medidas utilizadas

En este apartado se describirán las medidas utilizadas en el estudio, así como las características de cada una de ellas.

Como ya se mencionó en el segundo capítulo, las medidas usuales no sirven para problemas con desbalanceo, por ello hay que utilizar diferentes medidas que sean útiles para este tipo de problemas.

Antes de describir las diferentes medidas que se van a utilizar, se van a describir algunos conceptos básicos sobre problemas de clasificación. Un modelo de clasificación o clasificador es una función que permite decidir qué elementos de un conjunto de datos pertenecen a una cierta clase; para realizar esta decisión, el clasificador puede utilizar un umbral sobre un valor real o directamente obtener un valor discreto.

Pongamos el caso de un problema de clasificación binario, en la que los datos se clasifican como positivos, lo representaremos con la letra p, o negativos, lo representaremos con la letra n. Una vez se ha entrenado un clasificador y se han obtenido los resultados, estos se pueden representar en una Matriz de confusión (para clasificación binaria es una matriz de 2x2) de la siguiente forma.

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

Figura 6.1: Ejemplo matriz de confusión.

Los Verdaderos Positivos (VP) son aquellos datos que han sido clasificados como positivo y su clase real es positiva, los Verdaderos Negativos (VN) son aquellos datos que han sido clasificados como negativos y su clase real es negativa, los Falsos Negativos (FN) son aquellos datos que han sido clasificados como negativos y su clase real es positiva; por último, los Falsos Positivos (FP) son aquellos datos que han sido clasificados como positivos pero su clase real es negativa.

Sabiendo los valores de cada uno de estos datos, se pueden calcular medidas que contemplen un mal rendimiento por parte del clasificador al predecir alguna de las clases; esto es interesante para problemas con desbalanceo porque permite ver el rendimiento específico del clasificador para predecir la clase minoritaria.

Las medidas que se utilizarán para el estudio son las siguientes: AUC, Precision, Recall, F1 Score y G-Mean.

La medida Precision representa el porcentaje de los datos clasificados como positivos que realmente son positivos, es decir, de todos los datos que el clasificador ha predicho como positivos, cuántos de ellos son realmente positivos. Esta medida toma valores entre 0 y 1, valores cercanos a 1 indican un buen rendimiento por el clasificador para reconocer datos de la clase positiva. Esta medida se representa como:

$$Precision = \frac{TP}{TP + FP}$$

La medida de Recall o también conocida como True Positive Rate (TPR) representa el número el porcentaje de datos de la clase positiva que han sido clasificados como positivos. Toma valores entre 0 y 1, valores cercanos a 1 indican un buen rendimiento para reconocer la clase positiva, si el clasificador no fuera capaz de reconocer la clase positiva el número de Falsos Negativos sería muy alto y su valor se acercaría a 0.Esta medida se puede expresar como:

$$Recall = \frac{TP}{TP + FN}$$

A su vez, existe la medida True Negative Rate (TNR), que representa el porcentaje de ejemplos negativos bien clasificados, se define con la siguiente expresión.

$$TNR = \frac{TN}{TN + FP}$$

La medida F1 Score tiene en cuenta las dos anteriores para ser calculada, puede expresarse como:

$$F_1Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Esta medida mide el equilibrio entre las medidas Precision y Recall, si alguna de las dos tiene un valor bajo, el valor de F1 Score baja, para que aumente, el valor de ambas medidas debe aumentar. Toma valores entre 0 y 1.

La medida G-Mean representa la media geométrica, sirve también para medir el equilibrio entre Precision y Recall. Se representa de la siguiente forma:

$$G - Mean = \sqrt{Precision * Recall}$$

Para esta medida, si cualquier de las medidas toma valores bajo su valor es 0.
La última medida es AUC (Area Under Curve), esta medida representa el área bajo la curva ROC. La curva ROC es una representación gráfica del rendimiento entre TPR y TNR; cuanto más alto sea el valor de ambas medidas, mayor será el valor de la medida AUC.

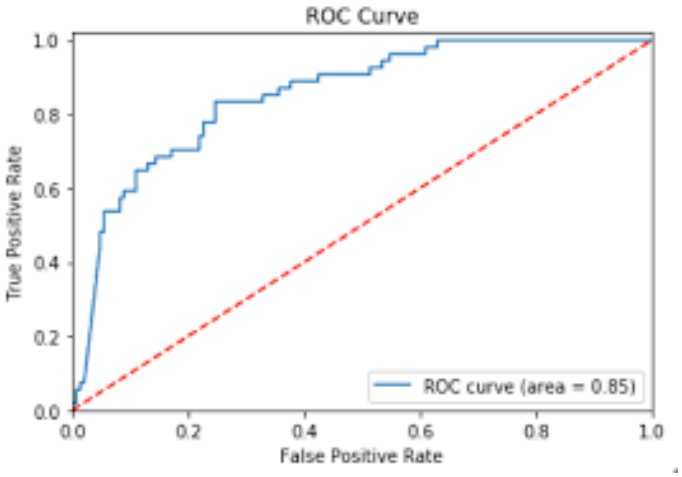


Figura 6.2: Ejemplo curva ROC.

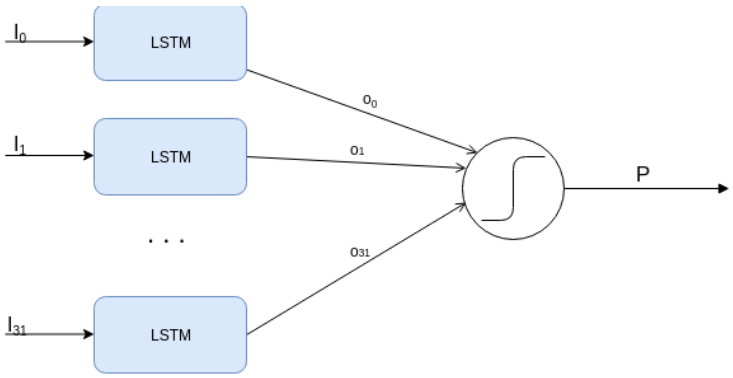
A partir de esta medida se puede medir el rendimiento de un clasificador, si el valor de AUC es 1, entonces el clasificador es perfecto. Si un clasificador hiciera las predicciones de forma aleatoria, su valor de AUC sería 0.5 y por lo tanto cualquier clasificador con valores cercanos o por debajo se considera que el clasificador es malo; para valores a partir de 0.7-0.75 se puede decir que el clasificador es aceptable.

6.3. Parámetros de la red

En este apartado se va a describir la estructura de la red utilizada en este estudio.

Como se comentó anteriormente, la red usada está formada por un conjunto de LSTMs para procesar la información de las series temporales.

La red básica utilizada está formada por dos capas; una capa formada por 32 LSTM y otra formada por una única neurona conectada a todas las LSTM usada para calcular la clase; para ello utiliza una función sigmoide como salida de la neurona. La estructura tendría la siguiente forma.



Cuando se define la capa de LSTM, se debe especificar el tamaño de la entrada que tendrá la LSTM; como en el caso de este estudio se utilizan diferentes tamaños de ventana, el tamaño de la entrada para la LSTM se especifica como $(1, tamventana * nseries)$, de esta forma cada LSTM procesa una ventana de información en cada momento.

Por último, la red utiliza 50 epochs para entrenar y un tamaño para el batch de 16.

Capítulo 7

Resultados y discusión

Capítulo 8

Conclusiones y trabajos futuros