Universidad Complutense de Madrid

Facultad de Informática

Aprendizaje Automático y Big Data

Memoria Proyecto Final:

Indicadores Personales de Problemas Cardiovasculares

Profesor:

- Alberto Díaz Esteban.

Alumnos:

- Marina de la Cruz López.

- Diego Alejandro Rodríguez Pereira.

Tabla de Contenidos

[Introducción 2](#_Toc104324098)

[Descripción de las variables. 2](#_Toc104324099)

[Preprocesamiento 4](#_Toc104324100)

[Adecuación de las variables para su uso en los algoritmos. 4](#_Toc104324101)

[Distribución de valores 5](#_Toc104324102)

[Correlaciones. 9](#_Toc104324103)

[Correlaciones con la variable de salida 11](#_Toc104324104)

[Eliminación de outliers 13](#_Toc104324105)

[Nuestro dataset se encuentra muy disperso y esto hace que la eliminación de los outliers sea difícil porque hay muchos valores que eliminar. Hemos probado a quitar los outliers usando la técnica del rango interquartil (IQR) donde eliminamos todos aquellos valores que se separen 1.5 veces por arriba del tercer quartil o 1.5 veces del primer cuartil. Esto se ha realizado para todas las columnas numéricas. El resultado es que reducimos la cantidad de personas con HeartDisease a casi la mitad, No: 292422 - Yes:  27373 a No: 215159 - Yes: 15861. 13](#_Toc104324106)

[Normalizado de los datos 13](#_Toc104324107)

[Balanceado de datos 13](#_Toc104324108)

[División training, validación y test. 14](#_Toc104324109)

[Función de coste. 14](#_Toc104324110)

[Regresión logística no regularizada. 15](#_Toc104324111)

[Regresión logística regularizada. 16](#_Toc104324112)

[Overfitting del modelo añadiendo grados. 18](#_Toc104324113)

[Variables Limitadas (10 variables con más correlación) 20](#_Toc104324114)

[Eliminación de outliers 21](#_Toc104324115)

[Eliminación de outliers y 10 variables 22](#_Toc104324116)

[Redes Neuronales 24](#_Toc104324117)

[Overfitting 24](#_Toc104324118)

[Dataset con 10 variables 27](#_Toc104324119)

[Eliminación de outliers 29](#_Toc104324120)

[Eliminación de outliers y dataset con 10 variables 32](#_Toc104324121)

[Support Vector Machines 35](#_Toc104324122)

[Kernel Lineal 35](#_Toc104324123)

[**Todas las variables** 35](#_Toc104324124)

[**Dataset con 11 variables** 37](#_Toc104324125)

[**Eliminación de outliers** 38](#_Toc104324126)

[**Eliminación de Outliers y dataset con 11 variables** 39](#_Toc104324127)

[Kernel Gaussiano 40](#_Toc104324128)

[**Todo el Dataset** 40](#_Toc104324129)

[**Eliminación de Outliers** 41](#_Toc104324130)

[**DataSet Reducido a 11 Variables** 42](#_Toc104324131)

[**Eliminación de outliers y dataset con 10 variables** 43](#_Toc104324132)

[Tiempos de ejecución 44](#_Toc104324133)

[Resumen Resultados 45](#_Toc104324134)

[Otros modelos 45](#_Toc104324135)

[Gradient Boosting 46](#_Toc104324136)

[Random Forest Clasificador 46](#_Toc104324137)

[BaggingClassifier 46](#_Toc104324138)

[Gaussian Naive Bayes 47](#_Toc104324139)

[Redes Neuronales (Scikit-Learn) 47](#_Toc104324140)

[Conclusiones 48](#_Toc104324141)

[Código 48](#_Toc104324142)

# **Introducción**

El objetivo de este proyecto es la predicción temprana de enfermedades cardiovasculares mediante factores personales obtenidos con una serie de preguntas básicas y sencillas en forma de cuestionario.  
 Las enfermedades cardiovasculares es una de las mayores causas de muertes. En Estados Unidos, donde ha sido realizada la encuesta, la mitad de la población tiene al menos uno de los tres factores claves de riesgo cardiovascular: alta presión arterial, alto colesterol y fumar. Otros factores principales incluyen: diabetes, obesidad, no hacer ejercicio y tomar mucho alcohol. Por lo que detectar y prevenir los factores que tienen un mayor impacto en enfermedades cardiovasculares es importante para la salud pública.

La información utilizada en este proyecto se ha obtenido originalmente de una encuesta realizada por la CDC con el Behavioral Risk Factor Survelliance System (BRFSS), la cual realiza una encuesta telefónica anual para recabar datos personales de los residentes de EE.UU. Este sistema de encuestas de salud es el mayor realizado en el mundo, obteniendo en 2020 resultados de alrededor de 400.000 personas junto con 279 variables, que corresponde a las preguntas realizadas. Esta cantidad de datos se redujo a 319.796 personas y unas 18 variables, las cuales se consideraron más relevantes.

[Personal Key Indicators of Heart Disease | Kaggle](https://www.kaggle.com/datasets/kamilpytlak/personal-key-indicators-of-heart-disease)

# **Descripción de las variables.**

El dataframe consiste en 18 columnas y un total de 319.796 filas, donde cada variable es un dato del paciente. Dentro de las 18 columnas tenemos como información, por un lado, la variable de salida, que es si el paciente ha tenido o no una enfermedad cardiovascular, que se llama: “HeartDisease” y es una variable binaria “yes” “no” (1,0).

Por otro lado, tenemos las variables de entrada que nuestro algoritmo usará para realizar la predicción:

Tabla 1. Variables del dataset

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Tipo | Rango | Descripción |
| BMI | Numérica continua | 12.02 - 94.85 | Representa el valor de BMI de la persona. |
| Smoking | Categórica | “yes”, “no” (1,0) | Representa si la persona ha fumado más de 100 cigarrillos (5 paquetes) durante su vida. |
| AlcoholDrinking | Categórica | “yes”, “no” (1,0) | Representa si la persona toma alcohol en exceso. Hombres más de 14 copas a la semana y en mujeres más de 7. |
| Stroke | Categórica | “yes”, “no” (1,0) | Representa si la persona ha tenido un accidente cerebrovascular. |
| PhysicalHealth | Numérica discreta | 0- 30 | Define cuántos días de los últimos 30 ha tenido la persona una enfermedad física o una herida. |
| MentalHealth | Numérica discreta | 0- 30 | Define cuántos días de los últimos 30 considera la persona que su salud mental fue mala. |
| DiffWalking | Categórica | “yes”, “no” (1,0) | Representa si la persona tiene dificultades severas al andar o subir escaleras. |
| Sex | Categórica | “Female”, “Male” | Sexo biológico de la persona, masculino o femenino. |
| AgeCategory | Categórica de rango | '18-24', '25-29' '30-34', '35-39' '40-44', '45-49' '50-54', '55-59' '60-64',   '65-69', '70-74' '75-79', '80 or older' | Rango de edad en el que se encuentra la persona dentro de los 14 niveles mostrados previamente. |
| Race | Categórica | 'American Indian/Alaskan Native', 'Asian' 'Black', 'Hispanic', 'Other,'   'White' | Raza de la persona. |
| Diabetic | Categórica | 'No'  'No, borderline diabetes'  'Yes'  'Yes (during pregnancy)' | Representa si alguna vez le han dicho a la persona que tenía diabetes. |
| PhysicalActivity | Categórica | “yes” “no” (1,0) | Representa si la persona ha realizado ejercicio (fuera de su trabajo) en los últimos 30 días. |
| GenHealth | Categórica | 'Excellent' 'Fair' 'Good' 'Poor' 'Very good' | Representa la percepción personal de cada una de las personas sobre cómo describiría su salud general. |
| SleepTime | Numérica  continua | 1 - 24 | Cantidad de horas de sueño en un periodo de 24 horas |
| Asthma | Categórica | “yes” “no” (1,0) | Representa si alguna vez le han dicho a la persona que tenía asma. |
| KidneyDisease | Categórica | “yes” “no” (1,0) | Representa si alguna vez le han dicho a la persona que tenía una enfermedad de riñones sin incluir cálculos renales, infección de la vejiga o incontinencia. |
| SkinCancer | Categórica | “yes” “no” (1,0) | Representa si alguna vez le han dicho a la persona que tenía cáncer de piel. |

# **Preprocesamiento**

## Adecuación de las variables para su uso en los algoritmos.

Lo primero que se ha realizado en el procesamiento es comprobar y eliminar cualquier fila que contuviese un valor nulo, es decir, un ‘nan’. Para esta operación se puede realizar una llamada a la función dropna() sobre el dataframe. No obstante, debido a que el dataframe utilizado ha sido previamente limpiado, es decir, se han eliminado los pacientes (filas) que tuviesen algún valor nulo, entonces la operación no ha tenido efecto sobre el dataframe.

De la tabla anterior, se puede observar que la mayoría de las variables con las que se trabaja en el proyecto son categóricas. A su vez, gran parte de los valores de estas variables son binarias, del estilo de sí o no. Por lo que, para este dataset se ha decidido transformar estos valores para que puedan funcionar en los algoritmos que se van a ejecutar. Para esto se ha transformado simplemente de los valores, de yes a 1 y de no a 0, es decir, “yes”: 1 “no”:0. Esto también ocurre para el parámetro “Sex”, ya que esta columna solo contiene dos valores, lo cuales son “female” y “male”, los lo que estos valores se pueden transformar en: “female”: 1, “male”:0.

En el caso de la variable “GenHealth”, la cual indica la salud general de la persona, también es una variable categórica, pero que contiene 5 valores distintos: 'Excellent', 'Very good', 'Good', 'Fair', 'Poor'. A su vez, y como se puede observar en la tabla anterior, estos posibles valores de la variable tienen un orden entre ellos, el cual se debe de preservar. Es decir: 'Excellent' > 'Very good' > 'Good' > 'Fair' > 'Poor'. Por lo que, al realizar la transformación de los valores de la variable, simplemente les vamos a dar valores de entre 0 y 4, dejando de esta manera el orden que existe de por sí entre las variables:   'Excellent': 4, 'Very good':3, 'Good':2, 'Fair':1, 'Poor':0.

En el caso de la variable “Race”, es al igual, una variable categórica la cual contiene 8 valores distintos, pero al contrario que en la variable “GenHealth”, no existe ningún tipo de orden entre ellos. Por lo que para hacer la transformación de esta variable hemos optado por la representación one-hot de los valores. Es decir, para cada una de las razas que aparecen metemos una nueva columna donde el 0 representa no pertenecer a esa raza y el 1 que sí pertenece.

Para la variable “Diabetic” es un poco más difícil, ya que 3 de los 4 valores sí que se les podría asumir un orden: 'No' , 'No, borderline diabetes',  'Yes'; pero al último valor, que es: 'Yes (during pregnancy)', no se le puede otorgar un orden. Finalmente hemos optado por la representación one-hot, igual que en el caso anterior, porque incluso en los valores en los que sí se puede inferir un orden se necesitarían más datos para realmente poder ordenar a los individuos (ej.: HAB1c) y nos ha parecido más correcto no hacerlo.

La variable “AgeCategory” es una variable categórica nominal donde cada valor representa un rango de edades. Para transformar esta variable hemos generado un diccionario con valores del 0 al 12 que representan los valores de los rangos de edades y los hemos asignado a cada rango de menor a mayor.

Las variables que nos faltan por procesar son las variables numéricas, las cuales en este punto no tenemos que adecuar porque ya se encuentran de forma numérica.

## Distribución de valores

La distribución de los valores numéricos los podemos ver en esta gráfica (Figura 1):

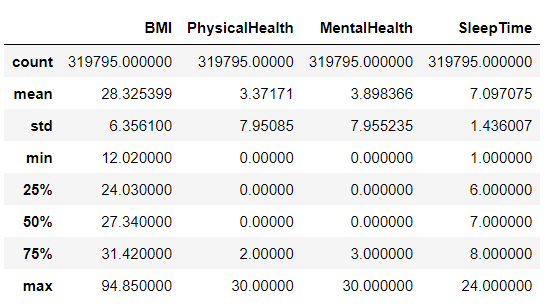


Figura 1. Distribución de las variables numéricas

De esta gráfica podemos sacar bastante información. Por ejemplo, la media del BMI de los participantes es de 28.32, valor que se encuentra dentro del rango de lo que se considera sobrepeso. Más aún, es que el 50% de los datos se encuentra en 27.34, lo cual nos indica que los participantes del estudio en su gran mayoría se encuentran dentro de este rango de sobrepeso.

Para las variables de “PhysicalHealth” y “MentalHealth” observamos que la mayoría de los participantes no han tenido problemas físicos o mentales en los últimos 30 días pero que existen una pequeña cantidad de outliers que hace que el máximo para ambas variables sea de 30.

Lo mismo ocurre con las horas de sueño, el 50% se encuentra en 7 horas pero algún individuo del estudio respondió 24 poniendo el máximo a este valor.

La distribución de las variables categóricas se puede observar en las siguientes gráficas:

Tabla 2. Distribución de las variables categóricas

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Vemos que para la mayoría de las variables la distribución no es 50-50, siendo la más equilibrada la del sexo. Las variables de salida, “HeartDisease” tiene una distribución muy desequilibrada, como resultado para hacer el entrenamiento de los datos vamos a tener que balancear los datos y usar otras métricas distintas del Accuracy para determinar lo bueno que es nuestro modelo.

## Correlaciones.

Para poder visualizar las variables que tienen mayor relevancia, para luego elegir aquellas que se van a utilizar en nuestros algoritmos, miraremos las correlaciones entre cada par de variables. Estas correlaciones nos ayudarán y nos proporcionarán pistas acerca de qué variables son las mejores para realizar la predicción con nuestros modelos.

Por ejemplo, en este dataset podemos visualizar la correlación que existe entre la variable de salida “HeartDisease” con el resto de las variables del conjunto de datos, y seleccionar aquellas variables/columnas que estén más correlacionadas con la variable de salida.

A su vez, también podemos visualizar las correlaciones de entre todas las variables, y buscar aquellas variables cuya correlación es muy alta, lo que significaría que para la predicción de nuestro modelo no tendrán relevancia alguna utilizar ambas, sino que bastaría con utilizar una de ellas, por ejemplo, quedarnos con aquella que tenga una correlación un poco mayor con la variable de salida y descartar la otra variable. En el caso de este dataset, esto ocurre con las variables “Diabetes” y “No\_Diabetes”, ya que ambas variables determinan lo mismo, que una persona tenga o no diabetes, es decir, una variable podría ser deducida a través de la otra, por lo que una de estas variables puede ser descartada mientras que la otra variable se mantenga.

Para el cálculo de las correlaciones entre cada par de variables se ha utilizado el método “spearman”, el cual se puede obtener utilizando la función de la librería de Pandas: df.corr(method='spearman'). El método “spearman” ha sido escogido porque este no solo determina las correlaciones lineales de las variables, sino que también las correlaciones monótonas entre ellas, por lo que lo convierte en un método más potente que el método de “pearson” que es otro de los métodos disponibles para el cálculo de las correlaciones.

Para la visualización de las correlaciones hemos creado un gráfico de heatMap que se muestra en la siguiente figura:

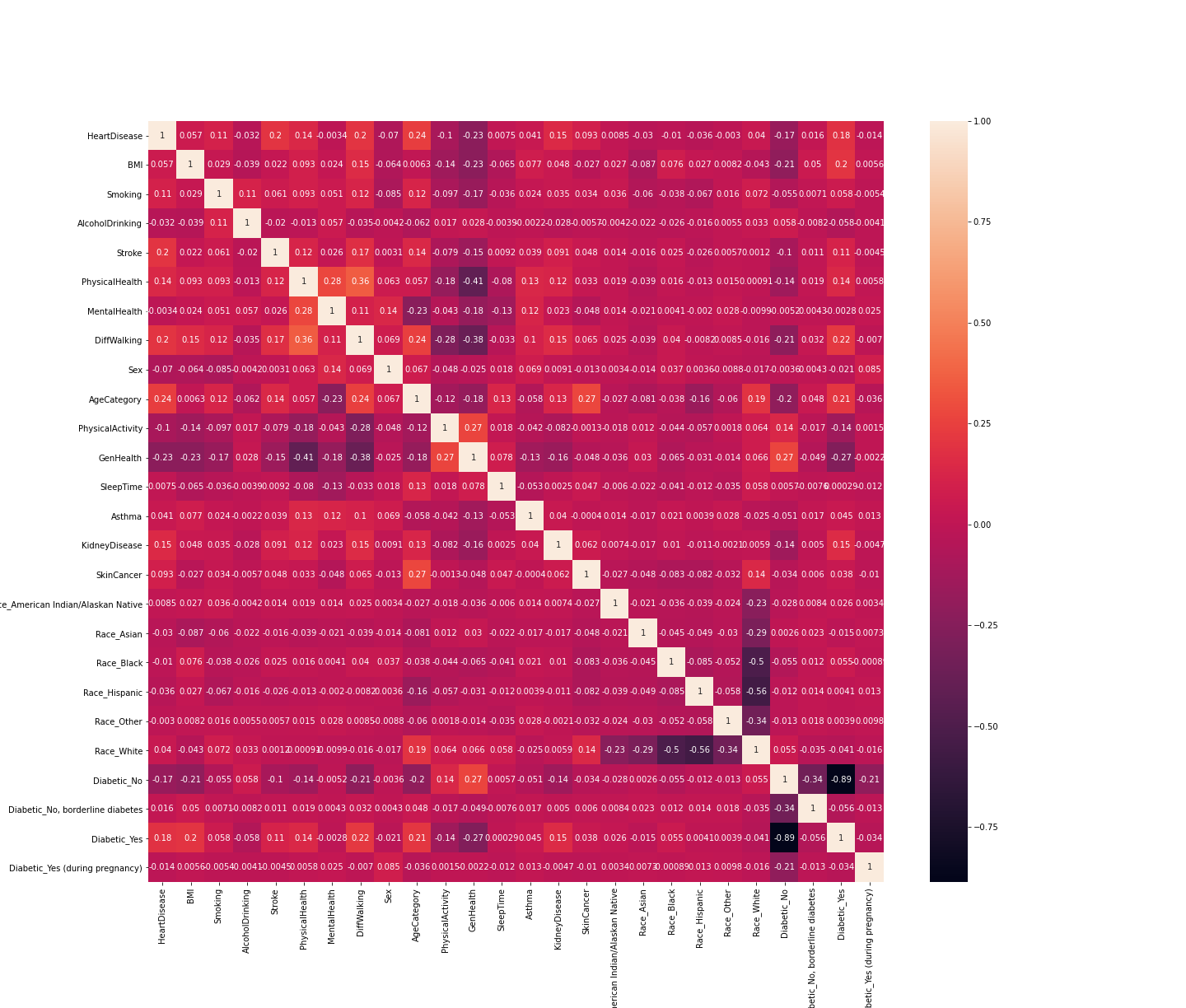


Figura 2. Gráfico de correlación

En este gráfico se puede observar que las correlaciones entre la mayoría de las variables del conjunto de datos son muy pequeñas.

Para un mejor entendimiento, si a partir de esta table de correlaciones sacamos únicamente aquellas variables que tienen una correlación cuyo valor es mayor a 0.5 entonces se obtendrían solamente 3 pares de variables que son las siguientes:

['Race\_Black && Race\_White: -0.5040402992221741',

'Race\_Hispanic && Race\_White: -0.5555711160223381' ,

'Diabetic\_No && Diabetic\_Yes: -0.886841227138652']

Estos tres pares de variables son las columnas que han sido obtenidas mediante la transformación one\_hot, por lo que tiene sentido que sean estas variables las que estén más correlacionadas entre sí, debido a que originalmente pertenecían a la misma variable/columna. Por ejemplo, para las variables “Diabetic\_No” y “Diabetic\_Yes”, en la mayor parte de los casos, si un individuo no es diabético tendrá en la variable “Diabetic\_No” un 1, y a su vez, en la variable “Diabetic\_Yes” tendrá un 0, y viceversa, porque a pesar de que existen otras dos categorías, que son “Diabetic\_No, bordeline diabetes” y “Diabetic\_Yes (during pregnancy)” son clases minoritarias de la variable original.

Si continuamos, y limitamos el valor de la correlación entre variables a 0.4 en valor absoluto obtenemos adicionalmente a las anteriores par de variables, un cuarto par que sería 'PhysicalHealth && GenHealth' con una correlación de -0.41.

['PhysicalHealth && GenHealth: -0.40792639880573317'

'Race\_Black && Race\_White: -0.5040402992221741',

'Race\_Hispanic && Race\_White: -0.5555711160223381' ,

'Diabetic\_No && Diabetic\_Yes: -0.886841227138652']

En el caso de “PhysicalHealth” y “GenHealth”, a pesar de que no provienen de una transformación de one\_hot, tienen mucha similitud entre sí, ya que la salud general (“GenHealth”) está muy correlacionada al esfuerzo físico que realice la persona (“PhysicalHealth”). Por lo que una persona que realice mucho ejercicio seguramente tenga una salud general buena, como a su vez, una persona que realice poco ejercicio es propensa a tener una salud general peor. Por esta razón es que existe una cierta relación entre este par de variables.

En general lo que podemos obtener de este estudio es que las variables de entrada de este dataset no se encuentran prácticamente correlacionadas entre sí excepto por algunos casos donde se ha producido la transformación one\_hot o en el caso de PhysicalHealth && GenHealth donde ambas están relacionadas con la salud de la persona (física o general).

## Correlaciones con la variable de salida

Una vez realizado la correlación entre cada par de variables, vamos a observar cuál es la correlación lineal de las variables de entrada con la variable de salida. Esto nos puede ayudar y nos puede dar una pista acerca de qué variables son más relevantes y cuáles escoger para realizar la predicción con los algoritmos y también nos puede indicar si la predicción va a ser sencilla o no de realizar.

El heatMap con todas las variables frente a la de salida es:

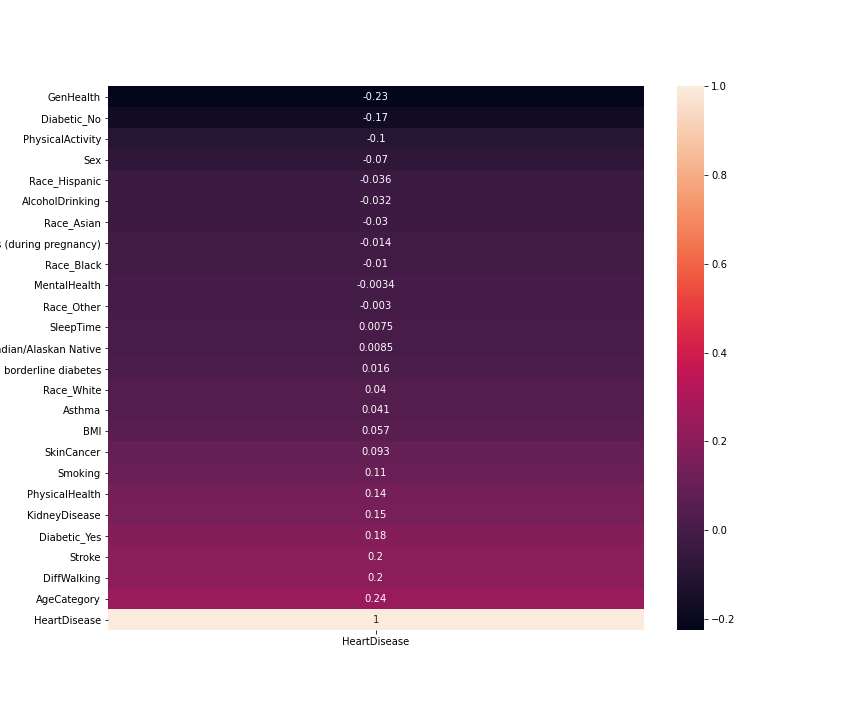


Figura 3. HeatMap de la correlación respecto a la variable de salida

En la gráfica se puede observar que la correlación lineal de las variables frente a la de salida es muy baja, siendo la variable con mayor correlación un 0.24. Debido a esto parece que hacer una selección de variables no nos va a ser de mucha ayuda porque no existe ningún subconjunto de variables que sean mucho mejores para realizar la predicción que otro. No obstante, podemos probar a usar sólo aquellas cuya correlación sea mayor que 0.1 en valor absoluto, obteniendo así:

GenHealth                             -0.225173

Diabetic\_No                           -0.170977

PhysicalActivity                      -0.100030

Smoking                                0.107764

PhysicalHealth                         0.143064

KidneyDisease                          0.145197

Diabetic\_Yes                           0.183072

Stroke                                 0.196835

DiffWalking                            0.201258

AgeCategory                            0.239298

Obtenemos así un subconjunto de 10 variables de las 26 originales. Luego podemos comparar los valores obtenidos en la predicción y los tiempos de ejecución del algoritmo.

## Eliminación de outliers

Nuestro dataset se encuentra muy disperso y esto hace que la eliminación de los outliers sea difícil porque hay muchos valores que eliminar. Hemos probado a quitar los outliers usando la técnica del rango interquartil (IQR) donde eliminamos todos aquellos valores que se separen 1.5 veces por arriba del tercer quartil o 1.5 veces del primer cuartil. Esto se ha realizado para todas las columnas numéricas. El resultado es que reducimos la cantidad de personas con HeartDisease a casi la mitad, No: 292422 - Yes:  27373 a No: 215159 - Yes: 15861.

## Normalizado de los datos

Dado que nuestros datos tienen rangos distintos, algunos se encuentran entre 0 y 30, otros entre 0 y 1, previamente a la ejecución de los algoritmos vamos a realizar un escalado de los datos. Para ello hacemos uso de StandardScaler() el cual realiza esta fórmula de transformación: X - mean(X) / std\_dev(X), que es la que hemos visto en clase para escalar datos.

## Balanceado de datos

Finalmente, previamente a ejecutar los algoritmos, hemos realizado un balanceado de nuestros datos, ya que cómo hemos visto previamente la cantidad de valores de salida que tienen “HeartDisease” a 1 son mucho menores que los que están a 0 (proporción: No: 91% Yes: 8%)

Como tenemos una gran cantidad de datos podemos hacer el balanceo mediante: Underbalancing, nos quedamos con todos los datos de la clase minoritaria y un subconjunto de la misma cantidad de datos de la clase mayoritaria, hay varias técnicas está la forma aleatoria que elige los datos de forma aleatoria y el NearMiss que realiza el balanceado quedándose con los valores de que clase mayoritaria que más se alejan de los de la clase minoritaria.

El resultado del Underbalancing será un dataset con un total de 54746 datos de entrada balanceado al 50% ([27373, 27373]).

Overbalancing, en este caso lo que obtenemos al final es un dataset con todos los datos de la clase mayoritaria y datos extra generados de forma sintética de la clase minoritaria. Se puede hacer de varias formas, por ejemplo, duplicando datos, o con el método SMOTE. En este método los ejemplos sintéticos se crean haciendo medias con otros ejemplos de la clase que existen en el propio dataset.

El resultado de hacer Overbalancing en nuestro dataset será un dataset con un total de 584844 datos de entrada balanceado al 50% ([292422, 292422]).

Ya que tenemos bastantes datos de entrada para nuestro algoritmo consideramos que haciendo un Underbalancing de los datos no vamos a perder información en exceso y obtenemos un subconjunto para no usar todos los datos, que son muchos.

## División training, validación y test.

Hemos realizado una división en training, validación y test para realizar las ejecuciones de nuestros algoritmos. Dado que tenemos bastantes datos de entrada hemos considerado que no hace falta realizar k-fold validación cruzada, que además nos añade tiempo de ejecución. La división será 60% training, 20% validación y 20% test, la hemos realizado usando *train\_test\_split* dos veces con stratify para la variable de salida para mantener la proporción original del dataframe. En los datos en entrenamiento hacemos el balanceado de datos, mientras que para hacer la validación y el test usamos la distribución original de los datos, ya que es lo que se va a encontrar el algoritmo y por lo tanto para asignar los parámetros es lo más correcto.

El resultado para por ejemplo Underbalancing es:

Training (array([0., 1.]), array([16424, 16424], dtype=int64))

Validación: (array([0., 1.]), array([58484, 5475], dtype=int64))

Test: (array([0., 1.]), array([58485, 5474], dtype=int64))

## Función de coste.

Como estamos tratando de predecir, a partir de unos datos obtenidos mediante encuestas de salud, si las personas van a padecer de problemas cardiovasculares o no, entonces, para la elección de nuestra función de coste primero debemos plantearnos qué es lo que queremos medir. Lo más importante es el poder reconocer personas que puedan padecer problemas cardiovasculares, es decir, cuán cierto es que van a tener problemas vasculares todas aquellas personas que los tienen, mientras que la precisión en la predicción no es tan importante. Ya que si predecimos que una persona va a tener problemas vasculares y luego resulta que la persona no los tiene, entonces lo único que puede suceder es que la persona tendrá un aviso para hacerse pruebas médicas para comprobar la predicción y comprobar su estado de salud a nivel médico, ya que el conjunto de datos proviene puramente de respuestas de una encuesta, pero no es información médica precisa de la persona.

Dado que la precisión y el recall suelen ser medidas opuestas para testear lo bueno que es nuestro algoritmo en validación y test usaremos el valor de recall de la clase 1 (aquella que predice que la persona va a tener problemas cardiovasculares.)

# **Regresión logística no regularizada.**

Una vez tenemos los datos preparados ya podemos hacer la regresión logística sobre el dataframe. Ya que nuestra variable de salida solo tiene 2 valores 0 y 1 podemos usar regresión logística normal mediante la función *opt.fmin\_tnc* a la cual le hemos pasado nuestra función de coste y gradiente programadas en prácticas anteriores.

Los primeros tests los vamos a realizar con el término de regularización a 0 (lmb=0), con esto lo que vamos a tratar de visualizar es si el dataset con las variables dadas tiende al sobreajuste, para ello visualizamos la curva de aprendizaje donde en el eje Y tenemos el valor del recall del modelo y en el eje X cantidad de datos de entrenamiento.

Los datos de entrenamiento empiezan en 10 y se van sumando de 1000 en 1000. Ya que los datos devueltos del *RandomUnderSampler* se encuentran ordenados vamos a hacer un shuffle previo de la matriz X de training y la salida Y (con el mismo orden).

El resultado obtenido en las curvas de aprendizaje para lambda= 0 es.

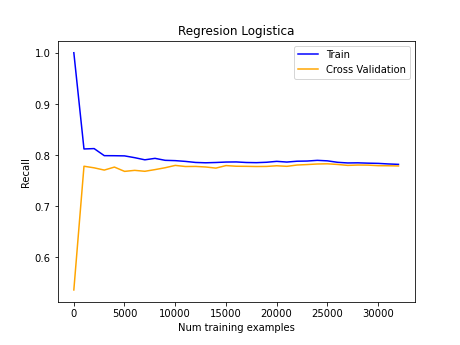


Figura 4. Curva de aprendizaje para lambda = 0

Es decir, no parece que el dataset tienda a sobreaprender, especialmente con la cantidad total de ejemplos de entrenamiento considerados.

Para comprobar esto vamos a generar las PolynomialFeatures de las 26 variables hasta el grado 3, dándonos en total 3276 variables, para lambda= 0 generamos la curva de aprendizaje con el recall frente a la cantidad de ejemplos. El resultado en el siguiente:

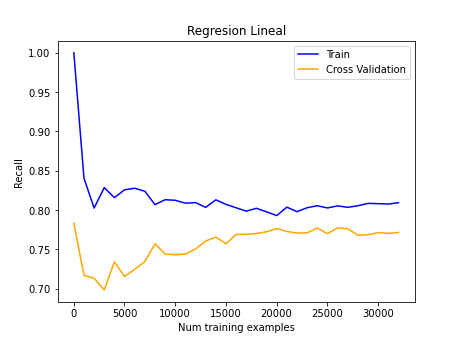


Figura 5. Curva de aprendizaje para polinomio de grado 3

Observamos que, aunque las curvas se separan un poco más que para el caso donde solo tenemos las 26 variables la diferencia no es muy grande, considerando la cantidad de variables que estamos introduciendo, 3276, la diferencia en el valor del Recall para entrenamiento y validación es de alrededor de un 3%.

El recall en test para todos los ejemplos de entrenamiento es:

             precision    recall  f1-score   support

          no       0.97      0.73      0.84     58485

          si       0.21      0.77      0.33      5474

    accuracy                           0.74     63959

   macro avg       0.59      0.75      0.58     63959

weighted avg       0.91      0.74      0.79     63959

Recall para la clase 1: 0.7709170624771647

# **Regresión logística regularizada.**

En este apartado vamos a explicar la implementación y mostrar resultados de la regresión logística regularizada.

En primer lugar, vamos a probar a ejecutar la regresión logística regularizada, con distintos valores para lambda.

Hemos ejecutado el algoritmo inicialmente con un rango muy amplio de lmb: lmb = [0.1,0.25, 0.5,0.75,1,1.5, 2,3,4,5,6,7,10,15,20,25,30,50,100,150,200] para ver qué valor es el mejor en validación y con todas las columnas del dataframe sin editar.

Esto nos va a dar una idea del valor de Recall que obtenemos con el dataframe tal cual y posteriormente podemos añadir o quitar variables.

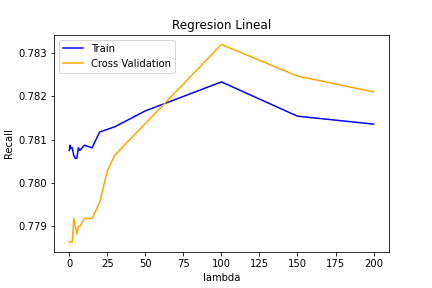


Figura 6. Recall respecto a lambda

Obtenemos que el mejor valor nos da para lambda: 100. Los resultados de la matriz de confusión son:

             precision    recall  f1-score   support

          no       0.97      0.75      0.84     58485

          si       0.22      0.78      0.35      5474

    accuracy                           0.75     63959

   macro avg       0.60      0.76      0.59     63959

weighted avg       0.91      0.75      0.80     63959

Recall para la clase 1: 0.7776762879064669

Una vez sabemos el mejor valor para lambda vamos a intentar probar con valores más parecidos para ver si existe un valor mejor de entre los cercanos a 100.

Probamos con lmb = [50,60,70,80,90,95,100,105,110,120]



Figura 7. Conjunto de lambdas distinto

El mejor valor nos da para lambda= 90

             precision    recall  f1-score   support

          no       0.97      0.75      0.84     58485

          si       0.22      0.78      0.35      5474

    accuracy                           0.75     63959

   macro avg       0.60      0.76      0.59     63959

weighted avg       0.91      0.75      0.80     63959

[[43589 14896]

 [ 1219  4255]]

Recall para la clase 1: 0.7773109243697479

Se puede observar en el gráfico que la diferencia de los valores de recall para las diferentes lambdas es bastante pequeña, alrededor de un 0.2%.

## Overfitting del modelo añadiendo grados.

Una vez que tenemos estos resultados base vamos a realizar un overfitting de los parámetros, metiendo variables extra (elevadas a un número) y después hacemos la regularización normal.

Para las variables elevadas a 3 estamos tratando con 3277 variables en total, si ejecutamos el algoritmo con la lista de valores de lambda original obtenemos:

lmb = [0,0.1,0.25,0.5,1,1.5, 2,3,4,5,6,7,10,15,20,25,30,50,100,150,200]

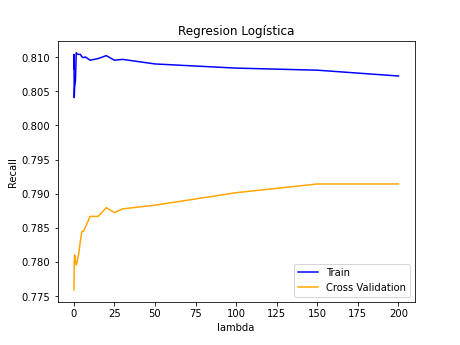


Figura 8. Overfitting del modelo

Donde en este caso el mejor valor nos da para lambda es lmb = 150.

Valores en test

Para lambda: 150

              precision    recall  f1-score   support

          no       0.97      0.73      0.84     58485

          si       0.22      0.79      0.34      5474

    accuracy                           0.74     63959

   macro avg       0.60      0.76      0.59     63959

weighted avg       0.91      0.74      0.79     63959

[[42938 15547]

 [ 1165  4309]]

Recall para la clase 1: 0.7871757398611618

Vamos a probar ahora con valores de lambda cercanos a 150 para ajustar al mejor valor posible.

lmb = [145,150,155,160]

Obtenemos el mismo valor en validación para lambda 145, 150 y 155 que es el que mostramos antes

## Variables Limitadas (10 variables con más correlación)

Vamos a considerar solo las 10 variables con mayor correlación entre ellas y la variable de salida para realizar la predicción. Si repetimos el proceso que realizamos en el dataset con todas las variables elevándose al cubo, para obtener un total de 286 variables, los resultados del Recall son:



Figura 9. Usando variables limitadas

Con lambda = 50 la mejor en validación.

Valores en test

Para lambda: 50

              precision    recall  f1-score   support

          no       0.97      0.70      0.82     58485

          si       0.20      0.80      0.32      5474

    accuracy                           0.71     63959

   macro avg       0.59      0.75      0.57     63959

weighted avg       0.91      0.71      0.77     63959

[[41163 17322]

 [ 1097  4377]]

Recall para la clase 1: 0.7995981001096091

Si probamos con valores cercanos para ajustar el resultado:

lmb = [40,45, 50,55,60,70]

Nos sigue dando que el mejor valor es lambda = 50

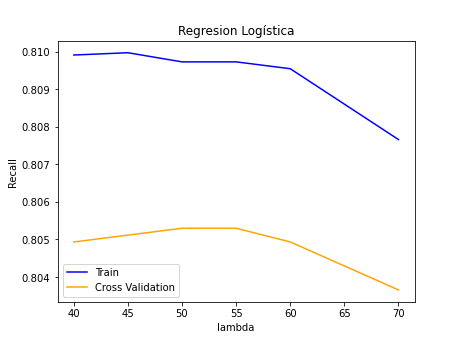


Figura 10. Distinto conjunto de lambdas

Es decir, los resultados obtenidos son ligeramente mejores (en un 0.9%) que cuando considerábamos todas las variables para entrenar nuestro algoritmo. Esto prueba que podemos usar el subset de 10 variables para realizar la predicción, porque funciona bastante similar.

## Eliminación de outliers

Usando el dataset con los outliers eliminados y las variables elevadas al cubo obtenemos:

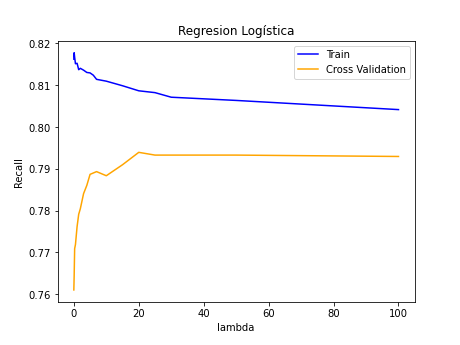


Figura 11. Eliminación de outliers

Para lmb = [0,0.1,0.25,0.5,1,1.5, 2,3,4,5,6,7,10,15,20,25,30,50,100] el mejor valor en validación lo obtenemos con lambda 20

Para lambda: 20

              precision    recall  f1-score   support

          no       0.98      0.73      0.84     42105

          si       0.17      0.79      0.28      3039

    accuracy                           0.73     45144

   macro avg       0.58      0.76      0.56     45144

weighted avg       0.93      0.73      0.80     45144

[[30763 11342]

 [  652  2387]]

Recall para la clase 1: 0.7854557420204015

## Eliminación de outliers y 10 variables

Resultados obtenidos de ejecutar regresión logística quitando los outliers y limitando las variables a las 10 más relevantes, dado que ha sido con lo que nos ha dado mejores resultados previamente. La cantidad de variables es de 287 porque las elevamos al cubo como antes.

lmb = [0,0.1,0.25,0.5,1,1.5, 2,3,4,5,6,7,10,15,20,25,30,50,100]

Valores en test

Para lambda: 30

              precision    recall  f1-score   support

          no       0.98      0.71      0.82     49503

          si       0.16      0.79      0.27      3506

    accuracy                           0.71     53009

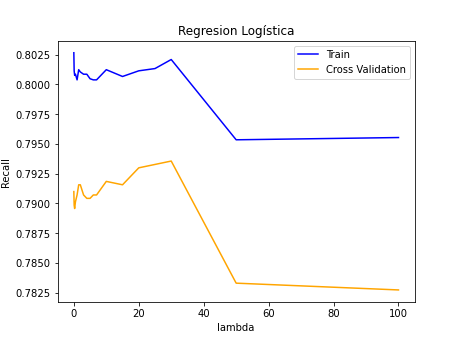
   macro avg       0.57      0.75      0.54     53009

weighted avg       0.93      0.71      0.79     53009

[[35033 14470]

 [  728  2778]]

Recall para la clase 1: 0.7923559612093554



Nuevamente podemos observar que efectivamente el dataset con menos variables funciona ligeramente mejor, aunque la eliminación de outliers no parece mejorar el resultado.

# **Redes Neuronales**

Usando el código ya programado en las prácticas anteriores lo hemos adecuado para que entrene la red con los datos de entrenamiento y posteriormente guarde el valor de la predicción del entrenamiento y la validación en una lista, para poder mostrarlos por pantalla. Se queda con los valores de Theta que dan mejores resultados en validación y los prueba en Test, estos son los resultados que luego muestra por pantalla.

Hemos probado primero a ejecutar la red con una cantidad elevada de neuronas en la capa intermedia para hacer overfitting de los datos. Posteriormente, hemos probado a realizar la predicción con varios valores del parámetro de regularización.

## Overfitting

Hemos probado distintos valores para la cantidad de neuronas de la capa intermedia “l= [5,10,20,50,100,150,200,250 ,300,400, 500,600,700]”. Para los valores bajos el Recall de la clase 1 para training y test son muy parecidos y empieza a empeorar ligeramente a partir de 250 neuronas, después de 250 a 700 no podemos observar visualmente que cambie mucho, por lo que hemos elegido ese valor para hacer las pruebas con los distintos valores de regularización.

Los resultados se obtienen realizando un incremento de los datos de 5000 en 5000, empezando desde 10:

Gráfica para 10 Neuronas:

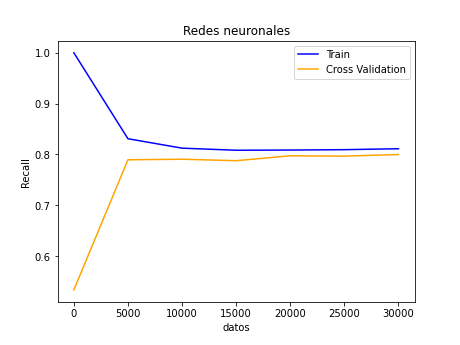


Figura 12. 10 neuronas

Gráfica para 50 Neuronas:

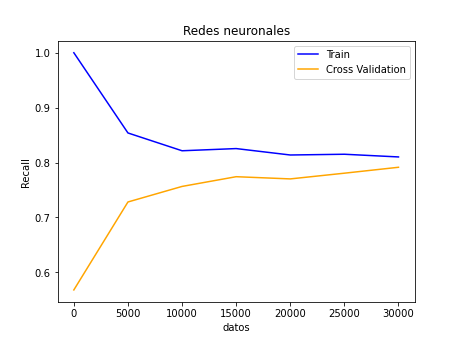


Figura 13. 50 neuronas

Gráfica para 250 Neuronas:

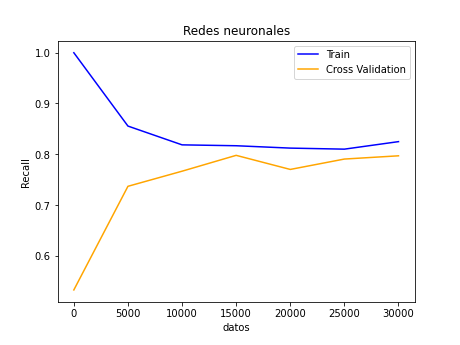


Figura 14. 250 neuronas

Gráfica para 700 Neuronas:



Figura 15. 700 neuronas

A partir de 250 neuronas no vemos que los valores en validación empeoren por lo que hemos decidido usar ese valor para hacer las predicciones.

Ahora entrenamos la red con distintos valores para lambda de entre: “lmb = [0,0.1,0.25,0.5,1,1.5, 2,3,4,5,6,7,10,15,20,25,30,50,100]” y obtenemos las mejores thetas en validación que nos dan los resultados `para el Test.

El mejor resultado que se ha obtenido usando 250 neuronas es para un valor de lambda igual a 1.5, como se puede observar en el siguiente resultado.

Best lambda value is: 1.5

              precision    recall  f1-score   support

          no       0.98      0.73      0.83     58485

          si       0.22      0.80      0.34      5474

    accuracy                           0.73     63959

   macro avg       0.60      0.77      0.59     63959

weighted avg       0.91      0.73      0.79     63959

Recall de la Clase1: 0.8021556448666423

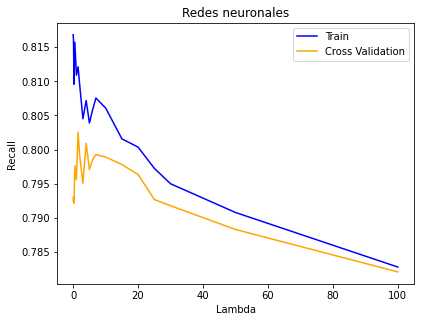


Figura 16. Curva de aprendizaje

Esto se encuentra un poco por encima de los obtenido con Regresión Logística.

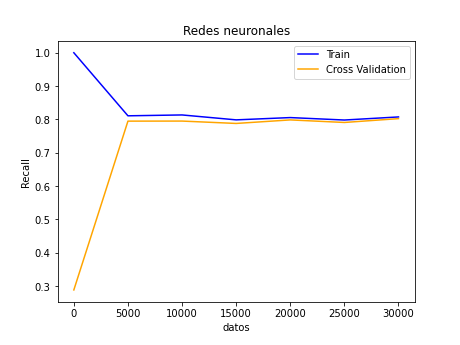
## Dataset con 10 variables

Hemos realizado las pruebas que las redes neuronales también sobre el dataset que solo contiene a las 10 variables con correlación superior a 0.1.

Número de neuronas utilizadas:

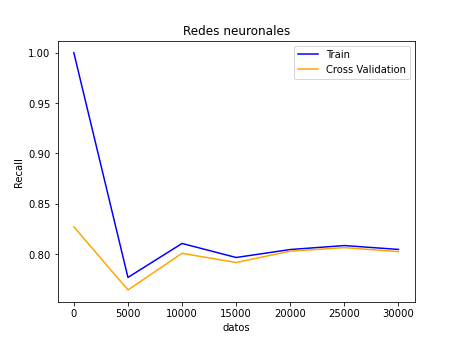
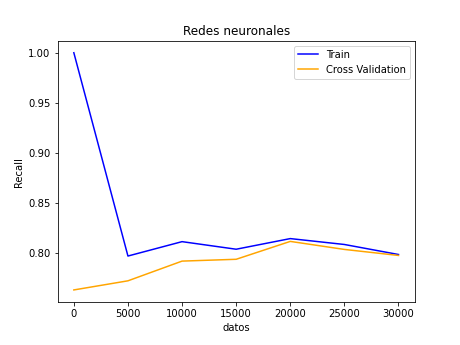
Vemos que ocurre lo mismo que cuando tenemos todas las variables, no realiza demasiado overfitting por la cantidad de datos. Por lo que hemos usado la red con 250 neuronas para sacar los resultados.

* 250 y 500

Chart, line chart

Description automatically generated

* 700 y 1000



Hemos probado distintos valores dentro de la lista de lambda: lmb = [0,0.1,0.25,0.5,1,1.5, 2,3,4,5,6,7,10,15,20,25,30,50,100] y nos ha dado como mejor valor en validación lmb = 6.

Best lambda value is: 6

              precision    recall  f1-score   support

          no       0.97      0.69      0.81     58485

          si       0.20      0.80      0.32      5474

    accuracy                           0.70     63959

   macro avg       0.59      0.75      0.56     63959

weighted avg       0.91      0.70      0.77     63959

Recall de la Clase1: 0.8036170990135184

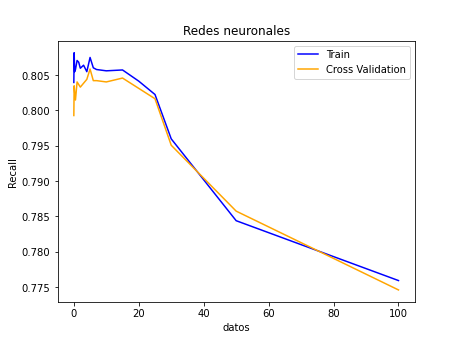


Figura 17

## Eliminación de outliers

Resultados obtenidos con Redes Neuronales al eliminar aquellas filas que contenían outliers:

Gráfico para 150 neuronas:

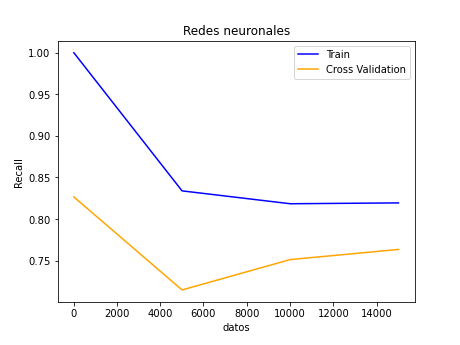


Figura 18. 150 neuronas

Gráfico para 250 neuronas:

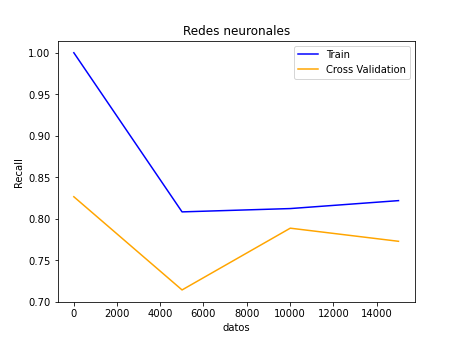


Figura 19. 250 neuronas

Gráfico para 500 neuronas:

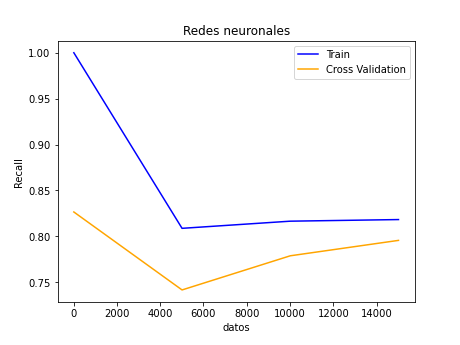


Figura 20. 500 neuronas

Valores de Lambda que se han utilizado para la ejecución:

Lambda = [0, 0.1, 0.25, 0.5, 1, 1.5, 2, 3, 4, 5, 6, 7, 10, 15, 20, 25, 30, 50, 100]

Para 250 neuronas se ha obtenido que la mejor lambda es 4.

Best lambda value is: 4

              precision    recall  f1-score   support

          no       0.98      0.73      0.83     42105

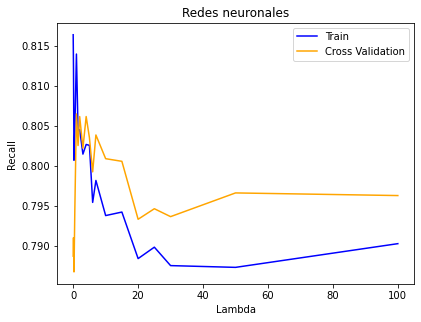
          si       0.17      0.80      0.29      3039

    accuracy                           0.73     45144

   macro avg       0.58      0.76      0.56     45144

weighted avg       0.93      0.73      0.80     45144

Recall de la Clase1: 0.7966436327739388



## Eliminación de outliers y dataset con 10 variables

Gráfico para 100 neuronas:

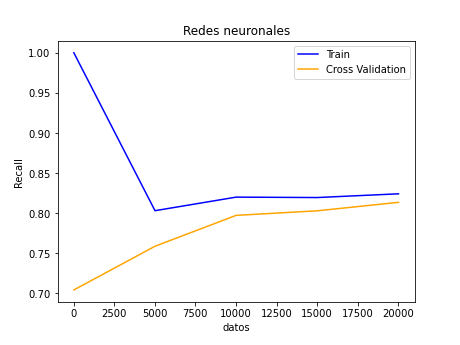


Figura 21. 100 neuronas

Gráfico para 250 neuronas:

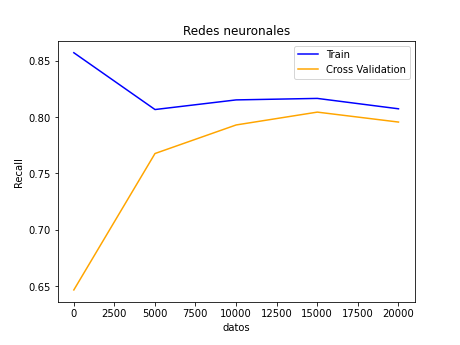


Figura 22. 250 neuronas

Gráfico para 500 neuronas:

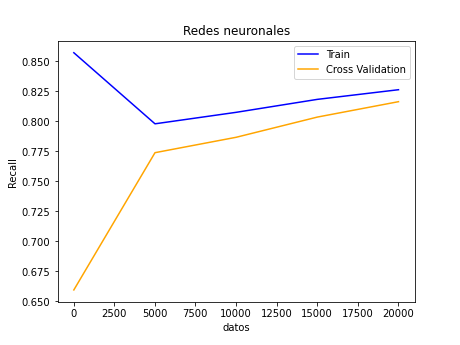


Figura 23. 500 neuronas

Resultado en test para 250 neuronas con los valores de lambda:

Se ha obtenido mejor resultado para lambda=2

Best lambda value is: 2

              precision    recall  f1-score   support

          no       0.98      0.73      0.84     42105

          si       0.18      0.80      0.29      3039

    accuracy                           0.73     45144

   macro avg       0.58      0.76      0.56     45144

weighted avg       0.93      0.73      0.80     45144

Recall de la Clase1: 0.8005923000987167

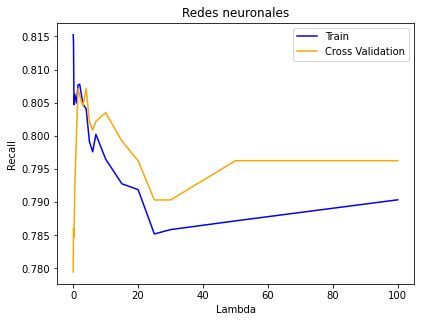


Figura 24. Resultado test 250 neuronas

# **Support Vector Machines**

En el caso de Support Vector Machines hemos probado a realizar la predicción para tanto el kernel lineal como el gaussiano. En general los resultados se asemejan bastante a los obtenidos previamente, aunque mejoran ligeramente en el porcentaje de Recall de la clase 1 (entre un 1 y un 2%).

## Kernel Lineal

### **Todas las variables**

En nuestro caso este Kernel no debería darnos los mejores resultados ya que aunque nuestra n es bastante grande con 26 variables en el dataset original tenemos una cantidad importante de datos, por lo que la m no es pequeña. Aún así los resultados obtenidos están al nivel de los obtenidos en regresión logística.

Para el kernel lineal hemos usado svm.SVC(kernel='linear' , C=i) con distintos valores para C de los que la lista: C = [0.01,0.02, 0.03,0.05, 0.1, 0.3, 0.5,1]

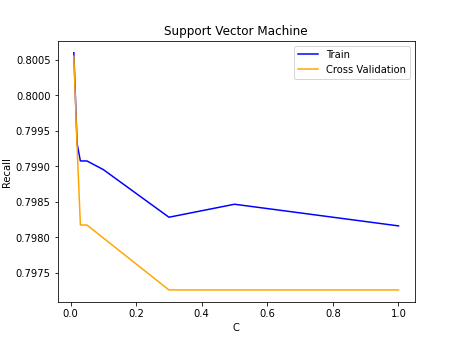


Figura 25. Test con todas las variables del dataset

El mejor valor entre ellos ha sido para C = 0.01 con un recall en Test de:

Test

Mejor valor de C : 0.01

Finished fit

Finished predicting

              precision    recall  f1-score   support

          no       0.97      0.73      0.83     58485

          si       0.22      0.80      0.34      5474

    accuracy                           0.73     63959

   macro avg       0.59      0.76      0.59     63959

weighted avg       0.91      0.73      0.79     63959

[[42606 15879]

 [ 1115  4359]]

Recall clase1: 0.7963098282791378

Este valor se asemeja a los obtenido con Regresión Logística, aunque el tiempo de ejecución es mucho mayor.

Si probamos con otros valores cercanos (C = [0.005, 0.01,0.015]) obtenemos que el mejor valor sigue siendo para C=0.01

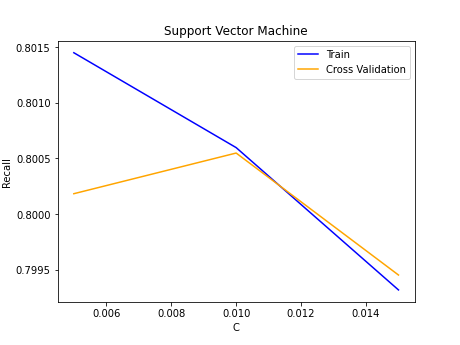


Figura 26. Distintos valores de C

### **Dataset con 11 variables**

Para esta ejecución del SVM con kernel lineal se ha utilizado el dataset con 10 variables unicamente en vez del dataset completo.

Se ha obtenido mejores valores usando C = 0.03

Mejor valor de C : 0.03

Finished fit

Finished predicting

              precision    recall  f1-score   support

          no       0.97      0.69      0.80     58485

          si       0.19      0.81      0.31      5474

    accuracy                           0.70     63959

   macro avg       0.58      0.75      0.56     63959

weighted avg       0.91      0.70      0.76     63959

[[40098 18387]

 [ 1048  4426]]

Recall de la Clase1: 0.8085495067592254

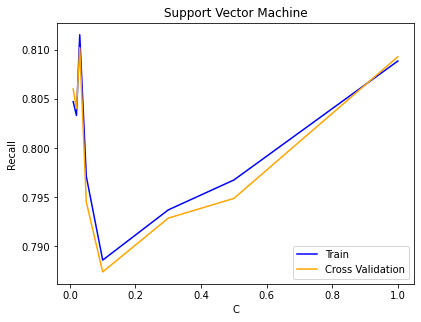


Figura 27. Dataset con 11 variables

### **Eliminación de outliers**

Para esta ejecución se ha utilizado el SVM con kernel lineal eliminando del dataset aquellas filas que contienen valores outliers.

El resultado ha sido que el mejor valor obtenido es usando C = 0.01

Test

Mejor valor de C : 0.01

Finished fit

Finished predicting

              precision    recall  f1-score   support

          no       0.98      0.71      0.83     42105

          si       0.17      0.80      0.28      3039

    accuracy                           0.72     45144

   macro avg       0.57      0.76      0.55     45144

weighted avg       0.93      0.72      0.79     45144

[[30033 12072]

 [  594  2445]]

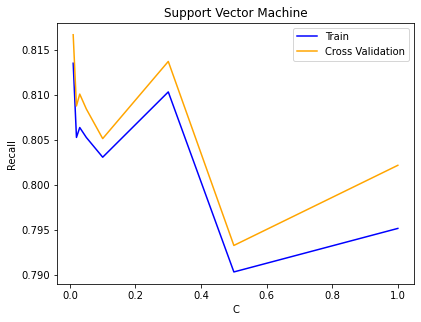


Figura 28. Con eliminación de outliers

### **Eliminación de Outliers y dataset con 11 variables**

Para esta ejecución se ha utilizado el dataset con 10 variables y se ha eliminado las filas que contenían valores outliers.

El mejor valor se ha obtenido con el parámetro C = 0.01

Mejor valor de C : 0.01

Finished fit

Finished predicting

              precision    recall  f1-score   support

          no       0.98      0.68      0.80     49503

          si       0.15      0.81      0.26      3506

    accuracy                           0.69     53009

   macro avg       0.57      0.75      0.53     53009

weighted avg       0.93      0.69      0.77     53009

[[33679 15824]

 [  654  2852]]

Recall de la Clase1: 0.8134626354820308

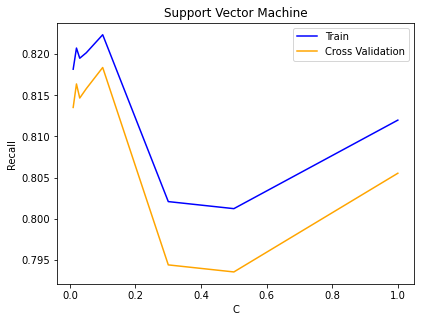


Figura 29. Con eliminación de outliers y dataset de 11 variables

## Kernel Gaussiano

Hemos probado también a ejecutar nuestro algoritmo con el kernel gaussiano ya que es más adecuado para el problema. Algo a notar es que en todas las pruebas anteriores habíamos considerado únicamente el Recall de la clase 1 porque el valor obtenido para cada clase estaba bastante equilibrado por lo que lo que nos parecía más importante era clasificar correctamente a los enfermos, en el caso del Kernel Gaussiano, para algunas ejecuciones se puede ver que empieza a existir un poco más de diferencia  en la separación por lo que quizás sería conveniente usar la media entre las dos clases en vez de solo el valor de la clase “si”.

Los valores de C y Sigma probados han sido los siguientes, con estos valores el problema de la separación de las clases no se ve tan acentuado:

C = [ 0.1, 0.5, 1, 10,20, 40]

sigma = [1, 5, 10,20,30, 40]

### **Todo el Dataset**

El mejor valor en Test lo obtenemos con C = 0.1 y Sigma = 1

Test results for best C: 0.1 and best sigma 1

Finished fit

Finished predicting

              precision    recall  f1-score   support

          no       0.98      0.63      0.77     58485

          si       0.18      0.86      0.30      5474

    accuracy                           0.65     63959

   macro avg       0.58      0.74      0.53     63959

weighted avg       0.91      0.65      0.73     63959

[[37098 21387]

 [  791  4683]]

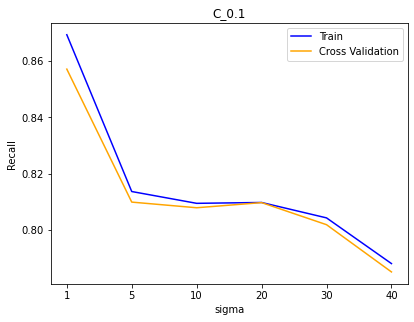


Figura 30. Con todo el dataset

### **Eliminación de Outliers**

En la ejecución del SVM con kernel Gaussiano, haciendo una eliminación de aquellas filas que contuviesen valores outliers, se ha obtenido que los parámetros que dan mejores resultados es para C = 0.1 y Sigma = 30

Test results for best C: 0.1 and best sigma 30

Finished fit

Finished predicting

              precision    recall  f1-score   support

          no       0.98      0.63      0.77     42105

          si       0.14      0.86      0.25      3039

    accuracy                           0.65     45144

   macro avg       0.56      0.74      0.51     45144

weighted avg       0.93      0.65      0.73     45144

[[26571 15534]

 [  431  2608]]

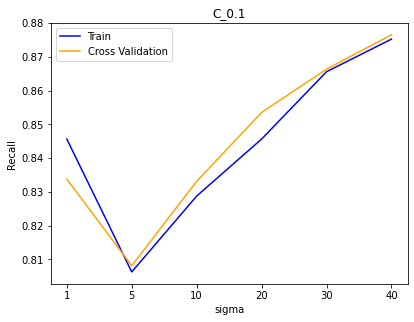
****

Figura 31. Eliminación de outliers

### **DataSet Reducido a 11 Variables**

Si probamos a ejecutar el Kernel Gaussiano con el Dataset reducido con varios valores para C y sigma:

C = [ 0.1, 0.5, 1, 10,20, 40]

sigma = [1, 5, 10,20,30, 40]

El mejor valor en Test lo obtenemos con C = 10 y Sigma = 10

Test results for best C: 10 and best sigma 10

Finished fit

Finished predicting

              precision    recall  f1-score   support

          no       0.98      0.68      0.80     58485

          si       0.19      0.82      0.31      5474

    accuracy                           0.69     63959

   macro avg       0.58      0.75      0.55     63959

weighted avg       0.91      0.69      0.76     63959

[[39504 18981]

 [  993  4481]]

Best Recall clase 1: 0.8185970040189989

Cuya gráfica de evolución asociada es:

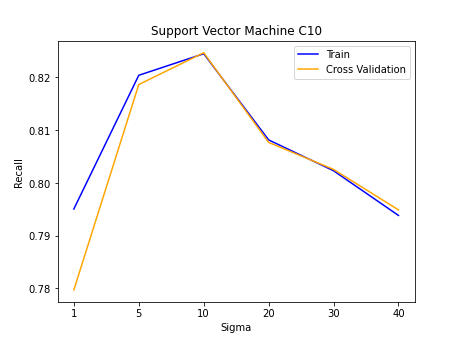


Figura 32. Con dataset de 11 variables

### **Eliminación de outliers y dataset con 10 variables**

Para esta ejecución del SVM con kernel Gaussiano, se ha utilizado el dataset con las 10 variables más relevantes del problema, y a partir de este dataset se ha eliminado aquellas filas que contenían valores outliers.

Se ha obtenido que los mejores valores de los parámetros son C = 0.1 y Sigma = 40.

Test results for best C: 0.1 and best sigma 40

Finished fit

Finished predicting

              precision    recall  f1-score   support

          no       0.98      0.61      0.76     49503

          si       0.14      0.87      0.24      3506

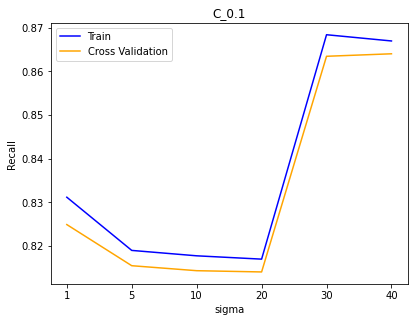
    accuracy                           0.63     53009

   macro avg       0.56      0.74      0.50     53009

weighted avg       0.93      0.63      0.72     53009

[[30349 19154]

 [  471  3035]]



# **Tiempos de ejecución**

|  |  |
| --- | --- |
| Algoritmo | Tiempo |
| Regresión Logística | 22.2 segundos |
| Redes Neuronales | 56.5 segundos |
| SVM Lineal | 2 minutos, 12 segundos |
| SVM Guassiano | 5 minutos, 30 segundos |

De media tarda unos 2 minutos por ejecución desde google collaborate, aunque depende del valor de C, cuanto más elevado más tarda en ejecutarse, por ejemplo para C=50 tarda unos 10 minutos.

# **Resumen Resultados**

Tabla con los mejores resultados obtenidos para cada test de los Realizados

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | DataSet Completo | 10 Variables más correladas | Sin outliers | Sin Outliers y 10 Variables | Mejor |
| Regresión Logística No Reg | 0.7709 “Sí”  0.73 “No” | —- | —-- | —-- | 0.7709 “Sí” |
| Regresión Logística Reg | 0.79 “Sí”  0.74 “No” | 0.7995 “Sí”  0.70 “No” | —-- | 0.7923 “Sí”  0.71 “No” | 0.7995 “Sí” |
| Redes Neuronales | 0.8021 “Sí”  0.73 “No” | 0.8036 “Sí”  0.69 “No” | 0.7966 “sí”  0.73 “No” | 0.8005 “Sí”  0.73 “No” | 0.8036 “Sí” |
| SVM Lineal | 0.7963 “Sí”  0.73 “No” | 0.81 “Sí”  0.69 “No” | 0.80 “Sí”  0.71 “No” | 0.8134 “Sí”  0.68 “No” | 0.8134 “Sí” |
| SVM Gausiano | 0.86 “Sí”  0.63 “No” | 0.82 “Sí”  0.68 “No” | 0.86 “Sí”  0.63 “No” | 0.87 “Sí”  0.61 “No” | 0.87 “Sí” |
|  |  |  |  |  |  |

# **Otros modelos**

Hemos realizado pruebas con otros algoritmos de aprendizaje automático de Sklearn para comprobar que los resultados obtenidos se encuentran en el mismo rango de los que hemos obtenido con nuestros modelos. Todos los algoritmos considerados tienen los parámetros asignados al default y no hemos realizado validación para ajustarlos, por lo que simplemente los hemos entrenado con los datos de entrenamiento balanceado del dataset original y hemos sacado el resultado en Test.

## Gradient Boosting

modelo = GradientBoostingClassifier(loss='deviance', random\_state=0)

Correct classification rate: 0.7365656123454087

              precision    recall  f1-score   support

         0.0       0.97      0.73      0.84     58485

         1.0       0.22      0.80      0.34      5474

    accuracy                           0.74     63959

   macro avg       0.60      0.76      0.59     63959

weighted avg       0.91      0.74      0.79     63959

Normalized confusion matrix

[[0.73088826 0.26911174]

 [0.20277676 0.79722324]]

## Random Forest Clasificador

modelo = RandomForestClassifier(criterion='gini', random\_state=0)

Correct classification rate: 0.7197423349333166

              precision    recall  f1-score   support

         0.0       0.97      0.72      0.82     58485

         1.0       0.20      0.76      0.32      5474

    accuracy                           0.72     63959

   macro avg       0.59      0.74      0.57     63959

weighted avg       0.90      0.72      0.78     63959

Normalized confusion matrix

[[0.71573908 0.28426092]

 [0.2374863  0.7625137 ]]

## BaggingClassifier

modelo=BaggingClassifier(  n\_estimators = 250 , max\_samples = 1.0 ,

                           bootstrap = True , bootstrap\_features = False ,

                         oob\_score = False , warm\_start = False , n\_jobs = None , random\_state = 0 , verbose = 0 )

Correct classification rate: 0.7157241357744806

              precision    recall  f1-score   support

         0.0       0.97      0.71      0.82     58485

         1.0       0.20      0.76      0.31      5474

    accuracy                           0.72     63959

   macro avg       0.58      0.74      0.57     63959

weighted avg       0.90      0.72      0.78     63959

Normalized confusion matrix

[[0.71165256 0.28834744]

 [0.24077457 0.75922543]]

## Gaussian Naive Bayes

modelo = BernoulliNB()

             precision    recall  f1-score   support

         0.0       0.96      0.76      0.85     58485

         1.0       0.22      0.70      0.33      5474

    accuracy                           0.76     63959

   macro avg       0.59      0.73      0.59     63959

weighted avg       0.90      0.76      0.81     63959

[[44605 13880]

 [ 1644  3830]]

## Redes Neuronales (Scikit-Learn)

modelo=MLPClassifier()

             precision    recall  f1-score   support

         0.0       0.98      0.70      0.81     58485

         1.0       0.20      0.81      0.32      5474

    accuracy                           0.71     63959

   macro avg       0.59      0.76      0.57     63959

weighted avg       0.91      0.71      0.77     63959

[[40861 17624]

 [ 1029  4445]]

A primera vista se puede observar que los resultados son parecidos, un poco peores, pero esto puede ser debido a que no estamos realizando ningún tipo de ajuste para los modelos, por lo que están en desventaja.

# **Conclusiones**

En conclusión, los resultados son bastante buenos, aunque no varían mucho con los distintos modelos. El mejor valor de recall para la clase 1 o “Sí” ha sido obtenido con Support Vector Machines aunque es a costa de un recall menor en la otra clase, por lo que realmente si hacemos la media no existe mucha diferencia.

La variación del porcentaje de los resultados obtenidos no es muy alta y está entre un 5% y un 7% para el recall de la clase 1 y entre un 1% y un 2% para la media de los dos Recall. El dataset nos ha parecido bastante complicado ya que al no usar datos médicos las predicciones no eran tan sencillas y porque se encontraba muy disperso, probablemente porque se trataban de respuestas de encuestas.

# **Código**

#Importacion de librerias al proyecto

import pandas as pd

import numpy as np

import seaborn as sns

%matplotlib inline

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.compose import make\_column\_transformer

from sklearn.preprocessing import OneHotEncoder

from sklearn import metrics

from sklearn.metrics import confusion\_matrix

from sklearn.utils.multiclass import unique\_labels

from sklearn.preprocessing import StandardScaler

from sklearn.preprocessing import MinMaxScaler

import scipy.optimize as opt

from sklearn.model\_selection import train\_test\_split

from imblearn.under\_sampling import NearMiss

from imblearn.over\_sampling import RandomOverSampler

from imblearn.under\_sampling import RandomUnderSampler

from imblearn.over\_sampling import SMOTE

from imblearn.ensemble import BalancedBaggingClassifier

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import classification\_report

from pylab import \*

import missingno as msno

from sklearn.preprocessing import PolynomialFeatures

from sklearn.utils import shuffle

#SVM

from sklearn import svm

from sklearn.model\_selection import cross\_val\_score

import matplotlib.pyplot as plt

from pyod.models.knn import KNN

import scipy.stats as stats

from scipy.stats import iqr

Lectura del dataset y primera visualización de los datos

#df = pd.read\_csv('heart\_dataset.csv')

df = pd.read\_csv('heart\_2020\_cleaned.csv')

print(df['HeartDisease'].value\_counts()/len(df))

print(df['HeartDisease'].value\_counts())

print(df.shape)

df

Descripción Variables

print(unique\_labels(df['Smoking']))

print(unique\_labels(df['AlcoholDrinking']))

print(unique\_labels(df['Stroke']))

print(unique\_labels(df['DiffWalking']))

print(unique\_labels(df['Sex']))

print(unique\_labels(df['AgeCategory']))

print(unique\_labels(df['Race']))

print(unique\_labels(df['Diabetic']))

print(unique\_labels(df['PhysicalActivity']))

print(unique\_labels(df['GenHealth']))

print(unique\_labels(df['Asthma']))

print(unique\_labels(df['KidneyDisease']))

print(unique\_labels(df['SkinCancer']))

df.describe()

df.describe().T.style.set\_properties(\*\*{'background-color': 'grey','color': 'white','border-color': 'white'})

Gráficos

i = 0

Labels = df.columns

Labels = ['HeartDisease', 'Smoking', 'AlcoholDrinking', 'Stroke', 'DiffWalking', 'Sex', 'AgeCategory','Race', 'Diabetic', 'PhysicalActivity', 'GenHealth','Asthma', 'KidneyDisease', 'SkinCancer']

i = 0

df.groupby(Labels[i]).size().plot(kind='pie', autopct='%.2f', label= Labels[i])

plt.savefig(Labels[i]+".png")

i = 1

df.groupby(Labels[i]).size().plot(kind='pie', autopct='%.2f', label= Labels[i])

plt.savefig(Labels[i]+".png")

i = 2

df.groupby(Labels[i]).size().plot(kind='pie', autopct='%.2f', label= Labels[i])

plt.savefig(Labels[i]+".png")

i = i+1

i = 3

df.groupby(Labels[i]).size().plot(kind='pie', autopct='%.2f', label= Labels[i])

plt.savefig(Labels[i]+".png")

i = 4

df.groupby(Labels[i]).size().plot(kind='pie', autopct='%.2f', label= Labels[i])

plt.savefig(Labels[i]+".png")

i = 5

df.groupby(Labels[i]).size().plot(kind='pie', autopct='%.2f', label= Labels[i])

plt.savefig(Labels[i]+".png")

i = 6

df.groupby(Labels[i]).size().plot(kind='pie', autopct='%.2f', label= Labels[i])

plt.savefig(Labels[i]+".png")

i = 7

df.groupby(Labels[i]).size().plot(kind='pie', autopct='%.2f', label= Labels[i])

plt.savefig(Labels[i]+".png")

i = 8

df.groupby(Labels[i]).size().plot(kind='pie', autopct='%.2f', label= Labels[i])

plt.savefig(Labels[i]+".png")

i = 9

df.groupby(Labels[i]).size().plot(kind='pie', autopct='%.2f', label= Labels[i])

plt.savefig(Labels[i]+".png")

i = 10

df.groupby(Labels[i]).size().plot(kind='pie', autopct='%.2f', label= Labels[i])

plt.savefig(Labels[i]+".png")

i = 11

df.groupby(Labels[i]).size().plot(kind='pie', autopct='%.2f', label= Labels[i])

plt.savefig(Labels[i]+".png")

i = 12

df.groupby(Labels[i]).size().plot(kind='pie', autopct='%.2f', label= Labels[i])

plt.savefig(Labels[i]+".png")

i = 13

df.groupby(Labels[i]).size().plot(kind='pie', autopct='%.2f', label= Labels[i])

plt.savefig(Labels[i]+".png")

Asignar variables categóricas con números

dic = {'No': 0, 'Yes': 1}

df['HeartDisease'] = df['HeartDisease'].map(dic)#.astype('category')

df['Smoking'] = df['Smoking'].map(dic)#.astype('category')

df['AlcoholDrinking'] = df['AlcoholDrinking'].map(dic)#.astype('category')

df['Stroke'] = df['Stroke'].map(dic)#.astype('category')

df['DiffWalking'] = df['DiffWalking'].map(dic)#.astype('category')

df['PhysicalActivity'] = df['PhysicalActivity'].map(dic)#.astype('category')

df['Asthma'] = df['Asthma'].map(dic)#.astype('category')

df['KidneyDisease'] = df['KidneyDisease'].map(dic)#.astype('category')

df['SkinCancer'] = df['SkinCancer'].map(dic)#.astype('category')

dic = {'Male': 0, 'Female': 1}

df['Sex'] = df['Sex'].map(dic)#.astype('category')

df

#Aplicacion de la transformacion one\_hot

y = pd.get\_dummies(df.Race, prefix='Race')

df = pd.concat([df, y], axis=1)

df

#Eliminacion de la columna Race

df.drop('Race', inplace=True, axis=1)

df

#AgeCategory

#A cada rango se le asigna un número

dic = {}

dic["18-24"] = 0

j = 1

for i in np.arange(25, 80, 5):

dic[str(i)+"-"+str(i+4)] = j

j = j+1

dic["80 or older"] = j

dic

#Cambiando los valores de AgeCategory de rango a valores discretos

df['AgeCategory'] = df['AgeCategory'].map(dic)#.astype('category')

#Aplicacion de la transformacion one\_hot a la columna Diabetic

x = pd.get\_dummies(df.Diabetic, prefix='Diabetic')

df = pd.concat([df, x], axis=1)

df.drop('Diabetic', inplace=True, axis=1)

df

df['Diabetic\_No'] = df['Diabetic\_No']#.astype('category')

df['Diabetic\_No, borderline diabetes'] = df['Diabetic\_No, borderline diabetes']#.astype('category')

df['Diabetic\_Yes'] = df['Diabetic\_Yes']#.astype('category')

df['Diabetic\_Yes (during pregnancy)'] = df['Diabetic\_Yes (during pregnancy)']#.astype('category')

#Distintos valores de la variable GenHealth

unique\_labels(df['GenHealth'])

#Cambiando los valores de string a enteros con orden

dic = {'Poor': 0, 'Fair': 1, 'Good': 2, 'Very good': 3, 'Excellent': 4}

df['GenHealth'] = df['GenHealth'].map(dic)#.astype('category')

#Descripcion del dataset de aquellas variables que son numericas

dfNumerical = pd.DataFrame(df, columns=["BMI","PhysicalHealth","MentalHealth","SleepTime"])

dfNumerical.describe()

#Descripcion de todo el dataset

df.describe()

Eliminación de NaNs

print(len(df))

df.dropna()

print(len(df))

Correlaciones

Como se puede observar en la tabla de debajo ninguna variable numérica se encuentra correlacionada a más de 0.5 en valor absoluto con otra variable numérica del dataset.

#Cálculo de las correlaciones usando el método Spearman

correl=df.corr(method='spearman')

#correl[abs(correl) >= 0.5]

correl

df.iloc[:, 0]

correl2 = correl[(abs(correl) != 1)]

correl2 = correl2[(abs(correl2) >= 0.5)]

correl2.dropna()

correl2

l =[]

for c in correl:

for r in correl:

#print(correl[c][r])

if abs(correl[c][r]) > 0.4 and correl[c][r] != 1:

l.append(c+" && " + r +": " + str(correl[c][r]))

print(l)

#Gráfico de las correlaciones de par de variables

plt.rcParams["figure.figsize"] = (20, 17)

heatMap = sns.heatmap(correl,

xticklabels=correl.columns,

yticklabels=correl.columns, annot = True)

plt.savefig("corr\_heatMap.png")

plt.rcParams["figure.figsize"] = plt.rcParamsDefault["figure.figsize"]

#Correlación de las varibles respecto a la variable de salida

plt.rcParams["figure.figsize"] = (12, 10)

order = correl["HeartDisease"].sort\_values();

print(order)

heatMap = sns.heatmap(order[:, np.newaxis], xticklabels=["HeartDisease"],

yticklabels=order.index, annot = True)

plt.savefig("corr\_heatMap\_HeartDisease.png")

plt.rcParams["figure.figsize"] = plt.rcParamsDefault["figure.figsize"]

Eliminar Outliers

i = 0

Labels = ["BMI","PhysicalHealth","MentalHealth","SleepTime"]

fig7, ax7 = plt.subplots()

ax7.set\_title(Labels[i])

ax7.boxplot(df[Labels[i]])

plt.savefig("BoxPlot.png")

# encontrar Q1, Q3 y rango intercuartílico para cada columna

Q1 = df[["BMI", "PhysicalHealth", "MentalHealth", "SleepTime"]].quantile (q = .25)

Q3 = df[["BMI", "PhysicalHealth", "MentalHealth", "SleepTime"]].quantile (q = .75)

IQR = df.apply (stats.iqr)

# solo mantenga las filas en el marco de datos que tengan valores dentro de 1.5 \* IQR de Q1 y Q3

data\_clean\_no\_outliers = df [~ ((df <(Q1-1.5 \* IQR)) | (df> (Q3 + 1.5 \* IQR))). any (axis = 1)]

# encontrar cuántas filas quedan en el marco de datos

data\_clean\_no\_outliers.shape

data\_clean\_no\_outliers["HeartDisease"].value\_counts()

Normalizar los datos

#Normalización de los datos usando estandarización

scaler\_x = StandardScaler()

#Tambien se puede aplicar MinMaxScaler, pero queremos mantener la distribución original de las variables

#scaler\_x = MinMaxScaler()

scaler\_x.fit(df)

data\_scaled = scaler\_x.transform(df)

df\_scaled = pd.DataFrame(data\_scaled, columns=df.columns)

print(df\_scaled)

#A las variables númericas se le aplica la normalización de los datos

df[['BMI', 'PhysicalHealth', 'MentalHealth', 'AgeCategory', 'GenHealth', 'SleepTime']] = df\_scaled[['BMI', 'PhysicalHealth', 'MentalHealth', 'AgeCategory', 'GenHealth', 'SleepTime']]

df.describe()

Correlación del dataset normalizado

#Reducción del dataset a 10 variables, usando únicamente las que están más relacionadas a la variable de salida

df\_copia = df[['HeartDisease', 'AgeCategory', 'DiffWalking', 'Diabetic\_Yes', 'KidneyDisease', 'PhysicalHealth', 'Smoking', 'GenHealth', 'Diabetic\_No', 'PhysicalActivity', 'Stroke']]

df\_copia

Dataset con 11 variables y eliminación de Outliers

# encontrar Q1, Q3 y rango intercuartílico para cada columna

Q1 = df\_copia[['PhysicalHealth']].quantile (q = .25)

Q3 = df\_copia[['PhysicalHealth']].quantile (q = .75)

IQR = df.apply(stats.iqr)

# solo mantenga las filas en el marco de datos que tengan valores dentro de 1.5 \* IQR de Q1 y Q3

data\_clean\_11var\_noOutliers = df\_copia[~ ((df\_copia <(Q1-1.5 \* IQR)) | (df\_copia> (Q3 + 1.5 \* IQR))). any (axis = 1)]

# encontrar cuántas filas quedan en el marco de datos

data\_clean\_11var\_noOutliers.shape

data\_clean\_11var\_noOutliers["HeartDisease"].value\_counts()

Balanceado de datos

df\_array = np.array(df)

X = df\_array[:, 1:]

m = np.shape(X)[0]

#Obtencion de la primera columna de la tabla, la de salida

Y = df\_array[:, 0]

n = np.shape(X)[1]

print(Y)

#Funciones para el balanceado de datos

#Smote hace un Oversampling

#Mientras que RandomSampler y NearMiss hacen un Undersampling

smote = SMOTE(random\_state=1000)

rus = RandomUnderSampler(random\_state=1000)

nm = NearMiss()

División en Training, Validación y Test

def train\_validation\_test(X, Y, sampler):

RANDOM\_STATE = 7777

#Dividimos los datos en training y test con 0.6 training y 0.4 test

X\_train, X\_temp, Y\_train, Y\_temp = train\_test\_split(X, Y, test\_size=0.4, random\_state=RANDOM\_STATE, stratify=Y)

X\_train, Y\_train = sampler.fit\_resample(X\_train,Y\_train)

#Los datos de test los volvemos a dividir en 0.2 validación y 0.2 test

X\_val , X\_test, Y\_val, Y\_test = train\_test\_split(X\_temp, Y\_temp, test\_size=0.5, random\_state=RANDOM\_STATE, stratify=Y\_temp)

return X\_train,Y\_train, X\_val,Y\_val, X\_test,Y\_test

#Comprobación y visualización de la cantidad de datos que obtenemos luego de hacer la división

#de entrenamiento, test y validación

X\_train,Y\_train, X\_val,Y\_val, X\_test,Y\_test = train\_validation\_test(X, Y, rus)

print(np.unique(Y\_train, return\_counts=True))

print(np.unique(Y\_val, return\_counts=True))

print(np.unique(Y\_test, return\_counts=True))

Curva de aprendizaje para visualizar underfiting y overfiting

#Función para pintar las gráficas de las curvas de aprendizaje

def draw\_learning\_curve(err1, err2, title, label, lmb):

b = err1

plt.title(title)

plt.xlabel(label)

plt.ylabel("Recall")

plt.plot(lmb, b, c="blue", label="Train")

d = err2[0:len(err1)]

plt.plot(lmb, d, c="orange", label="Cross Validation")

plt.legend()

Regresión Logística

#Función Sigmoide

def sigmoid\_funct(z):

return 1 / (1 + np.exp(-z))

#Función de coste regularizada

def func\_coste\_reg(Thetas, X, Y, lmb, m):

return cost\_funct(Thetas, X, Y) + regularizacion(Thetas[1:], lmb, m)

#Función de regularización

def regularizacion(Thetas, lmb, m):

return (lmb/(2\*m))\*np.sum(Thetas\*\*2)

#Función de gradiente regularizado

def func\_grad\_reg(Thetas, X, Y, lmb, m):

return np.add(gradient(Thetas, X, Y), reg\_grad(Thetas[1:], lmb, m))

#Regularización del gradiente

def reg\_grad(Thetas, lmb, m):

return np.insert(lmb/m\*Thetas, 0, values=[0])

#Función de coste sin regularización

def cost\_funct(Theta, X, Y):

g = sigmoid\_funct(np.matmul(X, Theta))

m = np.shape(X)[0]

J = (np.matmul(np.transpose(np.log(g)), Y)) + (np.matmul(np.transpose(np.log(1-g)), (1 - Y)))

return np.sum(-J)/m

#Función de gradiente sin regularización

def gradient(Theta, X, Y):

m = np.shape(X)[0]

g = sigmoid\_funct(np.matmul(X,Theta))

J = np.dot(np.transpose(X), (g - Y))

return J/m

#Función que calcula el porcentaje de aciertos para la regresión logística

def porcentaje\_aciertos\_RL(Theta, X, Y, mostrar):

# Calculamos los valores estimados segun la theta que hemos obtenido

sigmoid = sigmoid\_funct(np.matmul(X, Theta))

if mostrar:

print(classification\_report(Y,(sigmoid >= 0.5), target\_names=["no", "si"]))

print(confusion\_matrix(Y,(sigmoid >= 0.5)))

#Obtenemos la especificidad y la sensibilidad

recall\_0 = confusion\_matrix(Y,(sigmoid >= 0.5), normalize = "true")[0][0]

if(len(confusion\_matrix(Y,(sigmoid >= 0.5), normalize = "true")) < 2):

if(np.sum(Y) == 0):

recall\_1 = 1

else:

recall\_1 = 0

else:

recall\_1 = confusion\_matrix(Y,(sigmoid >= 0.5), normalize = "true")[1][1]

#print(recall\_0)

#print(recall\_1)

#print(len(Y))

#harm\_recall = 2/((1.0/recall\_0) + (1.0/recall\_1))

# Devolvemos el porcentaje

#return harm\_recall

return recall\_1

def mainRL\_Data(X\_train, Y\_train, X\_val, Y\_val, X\_test, Y\_test, lmb, mul, name):

#Obtencion de todas las columnas de la tabla menos la de salida la primera

m = np.shape(X\_train)[0]

print((Y\_train== 0).sum())

print( (Y\_train== 1).sum() )

#Inicializamos las Thetas a 0

#print(Thetas)

#Si la representación que le pasamos al algoritmo no tiene la primera columna ya puesta como 1 se la ponemos dentro

if(mul == False):

# Agregamos una columna de 1s

X\_train = np.hstack([np.ones([m, 1]), X\_train])

m\_val = np.shape(X\_val)[0]

print(m\_val)

# Agregamos una columna de 1s

X\_val = np.hstack([np.ones([m\_val, 1]), X\_val])

m\_t = np.shape(X\_test)[0]

print(m\_t)

# Agregamos una columna de 1s

X\_test = np.hstack([np.ones([m\_t, 1]), X\_test])

print(np.shape(X\_train))

errorT = []

errorV = []

n = np.shape(X\_train)[1]

print(n)

Thetas = np.zeros(n)

Max = 0

LambdaF= 0

rangeVals= [];

for data in range(10,m,1000):

rangeVals.append(data)

result = opt.fmin\_tnc(func=func\_coste\_reg, x0=Thetas, fprime=func\_grad\_reg, args=(X\_train[0:data],Y\_train[0:data],lmb,m), messages=0)

errorT.append(porcentaje\_aciertos\_RL(result[0], X\_train[0:data], Y\_train[0:data], False))

porcentaje = porcentaje\_aciertos\_RL(result[0], X\_val, Y\_val, False)

errorV.append(porcentaje)

if(porcentaje > Max):

Max = porcentaje

LambdaF = i

draw\_learning\_curve(errorT, errorV, "Regresion Logistica", "Num training examples", rangeVals)

plt.savefig(name)

#Test

print("Valores en test")

print("Para lambda: " + str(LambdaF))

result = opt.fmin\_tnc(func=func\_coste\_reg, x0=Thetas, fprime=func\_grad\_reg, args=(X\_train,Y\_train,LambdaF,m), messages=0)

print(result[0])

porcentaje = porcentaje\_aciertos\_RL(result[0], X\_test, Y\_test, True)

print("Recall para la clase 1: " + str(porcentaje))

def mainRL(X\_train, Y\_train, X\_val, Y\_val, X\_test, Y\_test, lmb, mul, name):

#Obtencion de todas las columnas de la tabla menos la de salida la primera

m = np.shape(X\_train)[0]

print((Y\_train== 0).sum())

print( (Y\_train== 1).sum() )

#Inicializamos las Thetas a 0

#Si la representación que le pasamos al algoritmo no tiene la primera columna ya puesta como 1 se la ponemos dentro

if(mul == False):

# Agregamos una columna de 1s

X\_train = np.hstack([np.ones([m, 1]), X\_train])

m\_val = np.shape(X\_val)[0]

print(m\_val)

# Agregamos una columna de 1s

X\_val = np.hstack([np.ones([m\_val, 1]), X\_val])

m\_t = np.shape(X\_test)[0]

print(m\_t)

# Agregamos una columna de 1s

X\_test = np.hstack([np.ones([m\_t, 1]), X\_test])

print(np.shape(X\_train))

errorT = []

errorV = []

n = np.shape(X\_train)[1]

print(n)

Thetas = np.zeros(n)

Max = 0

LambdaF= 0

#Para cada lambda calculamos las Thetas óptimas, pintamos la función y sacamos el porcentaje de aciertos

for i in lmb:

result = opt.fmin\_tnc(func=func\_coste\_reg, x0=Thetas, fprime=func\_grad\_reg, args=(X\_train,Y\_train,i,m), messages=0)

print("Lambda: " , i)

errorT.append(porcentaje\_aciertos\_RL(result[0], X\_train, Y\_train, False))

#print(result[0])

porcentaje = porcentaje\_aciertos\_RL(result[0], X\_val, Y\_val, False)

print("Resultado en validación: " + str(porcentaje))

errorV.append(porcentaje)

if(porcentaje > Max):

Max = porcentaje

LambdaF = i

draw\_learning\_curve(errorT, errorV, "Regresion Logística", "lambda", lmb)

plt.savefig(name)

#Test

print("Valores en test")

print("Para lambda: " + str(LambdaF))

result = opt.fmin\_tnc(func=func\_coste\_reg, x0=Thetas, fprime=func\_grad\_reg, args=(X\_train,Y\_train,LambdaF,m), messages=0)

#print(result[0])

porcentaje = porcentaje\_aciertos\_RL(result[0], X\_test, Y\_test, True)

print("Recall para la clase 1: " + str(porcentaje))

df\_array = np.array(df)

X = df\_array[:, 1:]

m = np.shape(X)[0]

#Obtencion de la primera columna de la tabla, la de salida

Y = df\_array[:, 0]

n = np.shape(X)[1]

#No regularizamos

lmb = 0

#Obtenemos los sets de test training y validación

X\_train,Y\_train, X\_val,Y\_val, X\_test,Y\_test = train\_validation\_test(X, Y, rus)

X\_train, Y\_train = shuffle(X\_train, Y\_train)

#Metemos más variables si es necesario, con esto estamos intentando que se produzca un overfitting

#Combinaciones de 2

poly = PolynomialFeatures(3)

X\_train = poly.fit\_transform(X\_train)

X\_val = poly.fit\_transform(X\_val)

X\_test = poly.fit\_transform(X\_test)

mainRL\_Data(X\_train,Y\_train, X\_val,Y\_val, X\_test,Y\_test, lmb, True, "Lmb\_0\_Pol2.png")

df\_array = np.array(df)

X = df\_array[:, 1:]

m = np.shape(X)[0]

#Obtencion de la primera columna de la tabla, la de salida

Y = df\_array[:, 0]

n = np.shape(X)[1]

#lmb = [0,0.1,0.25,0.5,1,1.5, 2,3,4,5,6,7,10,15,20,25,30,50,100,150,200]

lmb = [145,150,155,160,170]

#lmb = [15,16,17,18,19,20,21,22,23,24,25]

#lmb = [50,60100,150,200]

#lmb = [0,0.05, 0.1, 0.15, 0.2, 0.25]

#lmb = [50,60,70,80,90,95,100,105,110,120]

#lmb = [1]

#lmb = [0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5 ]

#X\_train,Y\_train, X\_val,Y\_val, X\_test,Y\_test = train\_validation\_test(Xrus, Yrus)

#Obtenemos los sets de test training y validación

X\_train,Y\_train, X\_val,Y\_val, X\_test,Y\_test = train\_validation\_test(X, Y, rus)

#Metemos más variables si es necesario, con esto estamos intentando que se produzca un overfitting

#Combinaciones de 3

poly = PolynomialFeatures(3)

X\_train = poly.fit\_transform(X\_train)

X\_val = poly.fit\_transform(X\_val)

X\_test = poly.fit\_transform(X\_test)

mainRL(X\_train,Y\_train, X\_val,Y\_val, X\_test,Y\_test, lmb, False, "RL\_reg\_0\_200\_pol3.png")

Dataset con 11 Variables

#Dataset con solo 10 variables, prueba de overfitting

df\_array = np.array(df\_copia)

X = df\_array[:, 1:]

m = np.shape(X)[0]

#Obtencion de la primera columna de la tabla, la de salida

Y = df\_array[:, 0]

n = np.shape(X)[1]

#print(Y)

#No regularizamos

lmb = 0

#Obtenemos los sets de test training y validación

X\_train,Y\_train, X\_val,Y\_val, X\_test,Y\_test = train\_validation\_test(X, Y, rus)

X\_train, Y\_train = shuffle(X\_train, Y\_train)

#Metemos más variables si es necesario, con esto estamos intentando que se produzca un overfitting

#Combinaciones de 3

poly = PolynomialFeatures(3)

X\_train = poly.fit\_transform(X\_train)

X\_val = poly.fit\_transform(X\_val)

X\_test = poly.fit\_transform(X\_test)

mainRL\_Data(X\_train,Y\_train, X\_val,Y\_val, X\_test,Y\_test, lmb, True, "10Var\_Lmb\_0\_Pol3.png")

#DataFrame solo con 10 variables

df\_array = np.array(df\_copia)

X = df\_array[:, 1:]

m = np.shape(X)[0]

#Obtencion de la primera columna de la tabla, la de salida

Y = df\_array[:, 0]

n = np.shape(X)[1]

#lmb = [0,0.1,0.25,0.5,1,1.5, 2,3,4,5,6,7,10,15,20,25,30,50,100,150,200]

lmb = [40,45, 50,55,60,70]

#Obtenemos los sets de test training y validación

X\_train,Y\_train, X\_val,Y\_val, X\_test,Y\_test = train\_validation\_test(X, Y, rus)

#Metemos más variables si es necesario, con esto estamos intentando que se produzca un overfitting

#Combinaciones de 2

poly = PolynomialFeatures(3)

X\_train = poly.fit\_transform(X\_train)

X\_val = poly.fit\_transform(X\_val)

X\_test = poly.fit\_transform(X\_test)

mainRL(X\_train,Y\_train, X\_val,Y\_val, X\_test,Y\_test, lmb, True, "RL\_reg\_10Var\_Tlmb.png")

Eliminación de Outliers

#Dataset entero eliminando outliers, prueba de overfitting

df\_array = np.array(data\_clean\_no\_outliers)

X = df\_array[:, 1:]

m = np.shape(X)[0]

#Obtencion de la primera columna de la tabla, la de salida

Y = df\_array[:, 0]

n = np.shape(X)[1]

#print(Y)

#No regularizamos

lmb = 0

#Obtenemos los sets de test training y validación

X\_train,Y\_train, X\_val,Y\_val, X\_test,Y\_test = train\_validation\_test(X, Y, rus)

X\_train, Y\_train = shuffle(X\_train, Y\_train)

#Metemos más variables si es necesario, con esto estamos intentando que se produzca un overfitting

#Combinaciones de 3

poly = PolynomialFeatures(3)

X\_train = poly.fit\_transform(X\_train)

X\_val = poly.fit\_transform(X\_val)

X\_test = poly.fit\_transform(X\_test)

mainRL\_Data(X\_train,Y\_train, X\_val,Y\_val, X\_test,Y\_test, lmb, True, "RL\_NO\_OUTLIERS\_DATA.png")

#DataFrame eliminando outliers

df\_array = np.array(data\_clean\_no\_outliers)

X = df\_array[:, 1:]

m = np.shape(X)[0]

#Obtencion de la primera columna de la tabla, la de salida

Y = df\_array[:, 0]

n = np.shape(X)[1]

lmb = [0, 0.1, 0.25, 0.5, 1, 1.5, 2, 3, 4, 5, 6, 7, 10, 15, 20, 25, 30, 40, 45, 50, 55, 60, 60, 100, 150, 200]

#lmb = [40, 45, 50, 55, 60, 70]

#Obtenemos los sets de test training y validación

X\_train,Y\_train, X\_val,Y\_val, X\_test,Y\_test = train\_validation\_test(X, Y, rus)

#Metemos más variables si es necesario, con esto estamos intentando que se produzca un overfitting

#Combinaciones de 2

poly = PolynomialFeatures(3)

X\_train = poly.fit\_transform(X\_train)

X\_val = poly.fit\_transform(X\_val)

X\_test = poly.fit\_transform(X\_test)

mainRL(X\_train,Y\_train, X\_val,Y\_val, X\_test,Y\_test, lmb, True, "RL\_NO\_OUTLIERS\_LAMBDA.png")

Dataset con 11 variables y eliminación de outliers

#Dataset entero eliminando outliers, prueba de overfitting

df\_array = np.array(data\_clean\_11var\_noOutliers)

X = df\_array[:, 1:]

m = np.shape(X)[0]

#Obtencion de la primera columna de la tabla, la de salida

Y = df\_array[:, 0]

n = np.shape(X)[1]

#print(Y)

#No regularizamos

lmb = 0

#Obtenemos los sets de test training y validación

X\_train,Y\_train, X\_val,Y\_val, X\_test,Y\_test = train\_validation\_test(X, Y, rus)

X\_train, Y\_train = shuffle(X\_train, Y\_train)

#Metemos más variables si es necesario, con esto estamos intentando que se produzca un overfitting

#Combinaciones de 3

poly = PolynomialFeatures(3)

X\_train = poly.fit\_transform(X\_train)

X\_val = poly.fit\_transform(X\_val)

X\_test = poly.fit\_transform(X\_test)

mainRL\_Data(X\_train,Y\_train, X\_val,Y\_val, X\_test,Y\_test, lmb, True, "RL\_NO\_OUTLIERS\_11VAR\_DATA.png")

#DataFrame eliminando outliers

df\_array = np.array(data\_clean\_11var\_noOutliers)

X = df\_array[:, 1:]

m = np.shape(X)[0]

#Obtencion de la primera columna de la tabla, la de salida

Y = df\_array[:, 0]

n = np.shape(X)[1]

lmb = [0, 0.1, 0.25, 0.5, 1, 1.5, 2, 3, 4, 5, 6, 7, 10, 15, 20, 25, 30, 40, 45, 50, 55, 60, 60, 100, 150, 200]

#lmb = [40, 45, 50, 55, 60, 70]

#Obtenemos los sets de test training y validación

X\_train,Y\_train, X\_val,Y\_val, X\_test,Y\_test = train\_validation\_test(X, Y, rus)

#Metemos más variables si es necesario, con esto estamos intentando que se produzca un overfitting

#Combinaciones de 2

poly = PolynomialFeatures(3)

X\_train = poly.fit\_transform(X\_train)

X\_val = poly.fit\_transform(X\_val)

X\_test = poly.fit\_transform(X\_test)

mainRL(X\_train,Y\_train, X\_val,Y\_val, X\_test,Y\_test, lmb, True, "RL\_NO\_OUTLIERS\_11VAR\_LAMBDA.png")

Redes Neuronales

#Función de coste regularizado

def func\_coste\_reg(Thetas, X, Y, lmb):

m = np.shape(X)[0]

return cost\_funct(Y, X, m) + regularizacion(Thetas, lmb, m)

#Función de coste sin regularización

def cost\_funct(Y, g, m):

J = np.sum(-1.0\*Y\* np.log(g) -1.0\*(1 - Y)\* np.log(1-g))

return J/m

#Función de regularización

def regularizacion(Thetas, lmb, m):

suma = 0

for i in Thetas:

i = i[:,1:]

suma += (np.sum(i\*\*2))

return (lmb/(2\*m))\*suma

#Función que implementa la propagación hacia adelante

def forward\_propagation(X, theta1, theta2, m, Y):

#Capa entrada asignamos la X con los unos incluidos

a1 = np.hstack([np.ones([m, 1]), X])

#capa intermedia (hidden) calculamos las ecuaciones de la anterior, aplicamos la sigmoide e incluimos los unos de la neurona 0

z2 = np.dot(a1, theta1.T)

a2 = np.hstack([np.ones([m, 1]), sigmoid\_funct(z2)])

#Capa salida calcuamos las ecuaciones con theta2 y aplicamos la sigmoide, nos devuelve la matriz de salida 5000x10

z3 = np.dot(a2, theta2.T)

a3 = sigmoid\_funct(z3)

return a1, z2, a2, z3, a3

#Función que genera los pesos aleatorios

def RandomWeights(entradas, salidas, ini):

Theta = np.random.uniform(-ini, ini, size = (salidas,entradas+1))

return Theta

#Función que implementa la propagación hacia atrás

def backprop(params\_rn, num\_entradas, num\_ocultas, num\_etiquetas, X, Y, reg):

#Recostruimos las Thetas

Theta1 = np.reshape(params\_rn[:num\_ocultas \*(num\_entradas + 1)],(num\_ocultas, (num\_entradas+1)))

Theta2 = np.reshape(params\_rn[num\_ocultas\*(num\_entradas + 1): ], (num\_etiquetas,(num\_ocultas+1)))

m = X.shape[0]

y\_onehot = Y

a1, z2, a2, z3, a3 = forward\_propagation(X, Theta1, Theta2, m, y\_onehot)

#Calculamos el coste

coste = func\_coste\_reg([Theta1, Theta2], a3, y\_onehot, reg)

d3 = a3 - y\_onehot

d2 = np.matmul(Theta2.T,d3.T)\*(a2\*(1-a2)).T

#Calculamos los gradientes no regularizados

G1 = np.matmul(d2[1:,:], a1)/m

G2 = np.matmul(d3.T,a2)/m

#Calculamos los gradientes regularizados

G1 = G1 + ((reg\*1.0)/m)\*np.insert(Theta1[:, 1:], 0, 0, axis = 1)

G2 = G2 + ((reg\*1.0)/m)\*np.insert(Theta2[:, 1:], 0, 0, axis = 1)

#Ponemos los gradientes en forma de lista

gradientes = np.concatenate((G1, G2), axis = None)

return coste, gradientes

#Función que calcula el porcentaje de aciertos para Redes Neuronales

def porcentaje\_aciertos\_RN(a3, X, Y, show):

# Calculamos los valores estimados segun la theta que hemos obtenido

if(show):

print(classification\_report(Y,(a3 >= 0.5), target\_names=["no", "si"]))

#Obtenemos la especificidad y la sensibilidad

recall\_0 = confusion\_matrix(Y,(a3 >= 0.5), normalize = "true")[0][0]

if(len(confusion\_matrix(Y,(a3 >= 0.5), normalize = "true")) < 2):

if(np.sum(Y) == 0):

recall\_1 = 1

else:

recall\_1 = 0

else:

recall\_1 = confusion\_matrix(Y, (a3 >= 0.5) , normalize = "true")[1][1]

#print(recall\_0)

#print(recall\_1)

#print(len(Y))

#harm\_recall = 2/((1.0/recall\_0) + (1.0/recall\_1))

# Devolvemos el porcentaje

#return harm\_recall

return recall\_1

def optimize\_backprop\_and\_check\_test ( X\_test,Y\_test, Theta1, Theta2):

#Resultados para Test

m = X\_test.shape[0]

#Hacemos el forward propagation

a1, z2, a2, z3, a3 = forward\_propagation(X\_test, Theta1, Theta2, m, Y\_test)

#Sacamos los aciertos

Y\_test = Y\_test.ravel()

accT = porcentaje\_aciertos\_RN(a3, X\_test, Y\_test, True)

return accT

def optimize\_backprop\_and\_check (num\_entradas, num\_ocultas, num\_etiquetas, reg, X, laps, Y):

ini1 = 0.1

ini2 = 0.1

#Inicializamos los pesos y los ponemos en forma de lista

Theta1 = RandomWeights(num\_entradas, num\_ocultas, ini1)

Theta2 = RandomWeights(num\_ocultas, num\_etiquetas, ini2)

pesos = np.concatenate((Theta1, Theta2), axis=None)

m = X.shape[0]

#Optimizamos

out = opt.minimize(fun = backprop, x0 = pesos, args = (num\_entradas, num\_ocultas, num\_etiquetas, X, Y, reg), method='TNC', jac = True, options = {'maxiter': laps})

#Reconstruimos las Thetas

Theta1 = out.x[:(num\_ocultas\*(num\_entradas+1))].reshape(num\_ocultas,(num\_entradas+1))

Theta2 = out.x[(num\_ocultas\*(num\_entradas+1)):].reshape(num\_etiquetas,(num\_ocultas+1))

return Theta1, Theta2

def optimize\_backprop\_training\_validation (num\_entradas, num\_ocultas, num\_etiquetas, reg, X, laps, Y, X\_val, Y\_val):

Theta1, Theta2 = optimize\_backprop\_and\_check (num\_entradas, num\_ocultas, num\_etiquetas, reg, X, laps, Y)

#Resultados para val

m\_val = X\_val.shape[0]

#Hacemos el forward propagation

a1, z2, a2, z3, a3 = forward\_propagation(X\_val, Theta1, Theta2, m\_val, Y\_val)

#Sacamos los aciertos

Y = Y.ravel()

accV = porcentaje\_aciertos\_RN(a3, X\_val, Y\_val, False)

#Resultados para Training

m = X.shape[0]

#Hacemos el forward propagation

a1, z2, a2, z3, a3 = forward\_propagation(X, Theta1, Theta2, m, Y)

#Sacamos los aciertos

#Y = Y.ravel()

accT = porcentaje\_aciertos\_RN(a3, X, Y, False)

return accV, accT, Theta1, Theta2

def main\_RN\_data(X\_train, Y\_train, X\_val, Y\_val, X\_test, Y\_test, lmb, mul, name, num\_hiden\_layers):

num\_labels = 1

num\_entries = np.shape(X\_train)[1]

reg = 0

m = X\_train.shape[0]

y = np.array([Y\_train])

y = y.T

Y\_val = np.array([Y\_val])

Y\_val = Y\_val.T

errorV = []

errorT = []

Theta1F = []

Theta2F = []

rangeVals = []

Max = 0

#print(y)

for data in range(10,m,5000):

rangeVals.append(data)

acc\_val, acc\_training, Theta1, Theta2 = optimize\_backprop\_training\_validation(num\_entries, num\_hiden\_layers, num\_labels, reg, X\_train[:data], 250, y[:data], X\_val, Y\_val)

errorV.append(acc\_val)

errorT.append(acc\_training)

if(acc\_val > Max):

Max = acc\_val

Theta1F = Theta1

Theta2F = Theta2

LambdaF = i

draw\_learning\_curve(errorT, errorV, "Redes neuronales", "datos", rangeVals)

plt.savefig(name)

acc= optimize\_backprop\_and\_check\_test ( X\_test,Y\_test, Theta1F, Theta2F)

return acc

def main\_RN\_lambda(X\_train, Y\_train, X\_val, Y\_val, X\_test, Y\_test, lmb, name,num\_hiden\_layers):

#Check del gradiente con reg

#print(grad.checkNNGradients(backprop, reg))

num\_labels = 1

num\_entries = np.shape(X\_train)[1]

#num\_hiden\_layers = 5

#reg = 0.01

m = X\_train.shape[0]

n = X\_train.shape[1]

y = np.array([Y\_train])

y = y.T

Y\_val = np.array([Y\_val])

Y\_val = Y\_val.T

errorV = []

errorT = []

Theta1F = []

Theta2F = []

rangeVals = []

Max = 0

#print(y)

for reg in lmb:

print("Para lambda: " + str(reg))

rangeVals.append(reg)

acc\_val, acc\_training, Theta1, Theta2 = optimize\_backprop\_training\_validation(num\_entries, num\_hiden\_layers, num\_labels, reg, X\_train, 250, y, X\_val, Y\_val)

errorV.append(acc\_val)

errorT.append(acc\_training)

print("Recall en validacion: " + str(acc\_val))

if(acc\_val > Max):

Max = acc\_val

Theta1F = Theta1

Theta2F = Theta2

LambdaF = reg

print("Best lambda value is: " + str(LambdaF))

draw\_learning\_curve(errorT, errorV, "Redes neuronales", "lambda", rangeVals)

plt.savefig(name)

acc= optimize\_backprop\_and\_check\_test ( X\_test,Y\_test, Theta1F, Theta2F)

return acc

Dataset completo

name = "RedesNeuronalesData"

#DataFrame

df\_array = np.array(df)

X = df\_array[:, 1:]

#Obtencion de la primera columna de la tabla, la de salida

Y = df\_array[:, 0]

#Obtenemos los sets de test training y validación

X\_train,Y\_train, X\_val,Y\_val, X\_test,Y\_test = train\_validation\_test(X, Y, rus)

print(X\_train.shape)

X\_train, Y\_train = shuffle(X\_train, Y\_train)

neuronas = [5, 10, 20, 50, 100, 150, 200, 250, 400, 500]

for neurona\_i in neuronas:

plt.clf() #Para limpiar el grafico anterior

name\_i = name + str(neurona\_i)

main\_RN\_data(X\_train, Y\_train, X\_val, Y\_val, X\_test, Y\_test, lmb, True, name\_i, neurona\_i)

name = "RedesneuronalesLMB.png"

#DataFrame

df\_array = np.array(df)

X = df\_array[:, 1:]

m = np.shape(X)[0]

#Obtencion de la primera columna de la tabla, la de salida

Y = df\_array[:, 0]

n = np.shape(X)[1]

#lmb = [0,0.1,1,2]

lmb = [0, 0.05, 0.1, 0.15, 0.25, 0.5, 1, 1.5, 2, 3, 4, 5, 6, 7, 10, 15, 20, 25, 30, 50, 90, 95, 100, 105, 110, 120]

#lmb = [8,9,9.5,10,10.5,11]

#lmb = [0,0.05, 0.1, 0.15, 0.2, 0.25]

#lmb = [50,60,70,80,90,95,100,105,110,120]

#lmb =

#Obtenemos los sets de test training y validación

X\_train,Y\_train, X\_val,Y\_val, X\_test,Y\_test = train\_validation\_test(X, Y, rus)

#X\_train, Y\_train = shuffle(X\_train, Y\_train)

main\_RN\_lambda(X\_train, Y\_train, X\_val, Y\_val, X\_test, Y\_test, lmb, name, 250)

name = "RedesNeuronalesData\_11VAR\_"

#DataFrame

df\_array = np.array(df\_copia)

X = df\_array[:, 1:]

#Obtencion de la primera columna de la tabla, la de salida

Y = df\_array[:, 0]

#Obtenemos los sets de test training y validación

X\_train,Y\_train, X\_val,Y\_val, X\_test,Y\_test = train\_validation\_test(X, Y, rus)

print(X\_train.shape)

X\_train, Y\_train = shuffle(X\_train, Y\_train)

neuronas = [5, 10, 20, 50, 100, 150, 200, 250, 400, 500]

for neurona\_i in neuronas:

plt.clf() #Para limpiar el grafico anterior

name\_i = name + str(neurona\_i)

main\_RN\_data(X\_train, Y\_train, X\_val, Y\_val, X\_test, Y\_test, lmb, True, name\_i, neurona\_i)

name = "RedesneuronalesLMB\_11VAR.png"

#DataFrame

df\_array = np.array(df\_copia)

X = df\_array[:, 1:]

m = np.shape(X)[0]

#Obtencion de la primera columna de la tabla, la de salida

Y = df\_array[:, 0]

n = np.shape(X)[1]

#lmb = [0,0.1,1,2]

lmb = [0, 0.05, 0.1, 0.15, 0.25, 0.5, 1, 1.5, 2, 3, 4, 5, 6, 7, 10, 15, 20, 25, 30, 50, 90, 95, 100, 105, 110, 120]

#lmb = [8,9,9.5,10,10.5,11]

#lmb = [0,0.05, 0.1, 0.15, 0.2, 0.25]

#lmb = [50,60,70,80,90,95,100,105,110,120]

#lmb =

#Obtenemos los sets de test training y validación

X\_train,Y\_train, X\_val,Y\_val, X\_test,Y\_test = train\_validation\_test(X, Y, rus)

#X\_train, Y\_train = shuffle(X\_train, Y\_train)

main\_RN\_lambda(X\_train, Y\_train, X\_val, Y\_val, X\_test, Y\_test, lmb, name, 250)

Eliminación Outliers

name = "RedesNeuronalesData\_NO\_OUTLIERS\_"

#DataFrame

df\_array = np.array(data\_clean\_no\_outliers)

X = df\_array[:, 1:]

#Obtencion de la primera columna de la tabla, la de salida

Y = df\_array[:, 0]

#Obtenemos los sets de test training y validación

X\_train,Y\_train, X\_val,Y\_val, X\_test,Y\_test = train\_validation\_test(X, Y, rus)

print(X\_train.shape)

X\_train, Y\_train = shuffle(X\_train, Y\_train)

neuronas = [5, 10, 20, 50, 100, 150, 200, 250, 400, 500]

for neurona\_i in neuronas:

plt.clf() #Para limpiar el grafico anterior

name\_i = name + str(neurona\_i)

main\_RN\_data(X\_train, Y\_train, X\_val, Y\_val, X\_test, Y\_test, lmb, True, name\_i, neurona\_i)

name = "RedesneuronalesLMB\_NO\_OUTLIERS.png"

#DataFrame

df\_array = np.array(data\_clean\_no\_outliers)

X = df\_array[:, 1:]

m = np.shape(X)[0]

#Obtencion de la primera columna de la tabla, la de salida

Y = df\_array[:, 0]

n = np.shape(X)[1]

#lmb = [0,0.1,1,2]

lmb = [0, 0.05, 0.1, 0.15, 0.25, 0.5, 1, 1.5, 2, 3, 4, 5, 6, 7, 10, 15, 20, 25, 30, 50, 90, 95, 100, 105, 110, 120]

#lmb = [8,9,9.5,10,10.5,11]

#lmb = [0,0.05, 0.1, 0.15, 0.2, 0.25]

#lmb = [50,60,70,80,90,95,100,105,110,120]

#lmb =

#Obtenemos los sets de test training y validación

X\_train,Y\_train, X\_val,Y\_val, X\_test,Y\_test = train\_validation\_test(X, Y, rus)

#X\_train, Y\_train = shuffle(X\_train, Y\_train)

main\_RN\_lambda(X\_train, Y\_train, X\_val, Y\_val, X\_test, Y\_test, lmb, name, 250)

Dataset de 11 variables y eliminación de outliers

name = "RedesNeuronalesData\_11VAR\_NO\_OUTLIERS\_"

#DataFrame

df\_array = np.array(data\_clean\_11var\_noOutliers)

X = df\_array[:, 1:]

#Obtencion de la primera columna de la tabla, la de salida

Y = df\_array[:, 0]

#Obtenemos los sets de test training y validación

X\_train,Y\_train, X\_val,Y\_val, X\_test,Y\_test = train\_validation\_test(X, Y, rus)

print(X\_train.shape)

X\_train, Y\_train = shuffle(X\_train, Y\_train)

neuronas = [5, 10, 20, 50, 100, 150, 200, 250, 400, 500]

for neurona\_i in neuronas:

plt.clf() #Para limpiar el grafico anterior

name\_i = name + str(neurona\_i)

main\_RN\_data(X\_train, Y\_train, X\_val, Y\_val, X\_test, Y\_test, lmb, True, name\_i, neurona\_i)

name = "RedesneuronalesLMB\_11VAR\_NO\_OUTLIERS.png"

#DataFrame

df\_array = np.array(data\_clean\_11var\_noOutliers)

X = df\_array[:, 1:]

m = np.shape(X)[0]

#Obtencion de la primera columna de la tabla, la de salida

Y = df\_array[:, 0]

n = np.shape(X)[1]

#lmb = [0,0.1,1,2]

lmb = [0, 0.05, 0.1, 0.15, 0.25, 0.5, 1, 1.5, 2, 3, 4, 5, 6, 7, 10, 15, 20, 25, 30, 50, 90, 95, 100, 105, 110, 120]

#lmb = [8,9,9.5,10,10.5,11]

#lmb = [0,0.05, 0.1, 0.15, 0.2, 0.25]

#lmb = [50,60,70,80,90,95,100,105,110,120]

#lmb =

#Obtenemos los sets de test training y validación

X\_train,Y\_train, X\_val,Y\_val, X\_test,Y\_test = train\_validation\_test(X, Y, rus)

#X\_train, Y\_train = shuffle(X\_train, Y\_train)

main\_RN\_lambda(X\_train, Y\_train, X\_val, Y\_val, X\_test, Y\_test, lmb, name, 250)

SVM

Lineal

def porcentaje\_aciertos\_SVM(Y,Y\_pred, mostrar):

if mostrar:

print(classification\_report(Y,Y\_pred, target\_names=["no", "si"]))

print(confusion\_matrix(Y,Y\_pred))

#Obtenemos la especificidad y la sensibilidad

recall\_0 = confusion\_matrix(Y,Y\_pred, normalize = "true")[0][0]

if(len(confusion\_matrix(Y,Y\_pred, normalize = "true")) < 2):

if(np.sum(Y) == 0):

recall\_1 = 1

else:

recall\_1 = 0

else:

recall\_1 = confusion\_matrix(Y,Y\_pred, normalize = "true")[1][1]

#print(recall\_0)

#print(recall\_1)

#print(len(Y))

#harm\_recall = 2/((1.0/recall\_0) + (1.0/recall\_1))

# Devolvemos el porcentaje

#return harm\_recall

return recall\_1

def Main\_svm\_linearKernel(X\_train, Y\_train, X\_val, Y\_val, X\_test, Y\_test, C, name):

c = 0

Max = 0

BestC = 0

accAux = 0

errorV = []

errorT = []

for i in C:

print("Para C: " + str(i))

svm\_clf = svm.SVC(kernel='linear' , C=i)

svm\_clf.fit(X\_train, Y\_train)

T\_svm\_predict = svm\_clf.predict(X\_train)

errorT.append(porcentaje\_aciertos\_SVM(Y\_train, T\_svm\_predict, False))

print("Finished fit")

Y\_svm\_predict = svm\_clf.predict(X\_val)

print("Finished predicting")

porcentaje = porcentaje\_aciertos\_SVM(Y\_val, Y\_svm\_predict, False)

print("Recall en validación: "+ str(porcentaje))

errorV.append(porcentaje)

if porcentaje > Max:

Max = porcentaje

BestC = i

#Confusion Matrix

title\_img = "randomState\_linearKernel" + str(i) + ".png"

#plot\_confusion\_matrix(Y\_svm\_test, Y\_svm\_predict, ['normal', 'enfermo'], normalize=True, title=title\_img)

#print(confusion\_matrix(Y\_svm\_test, Y\_svm\_predict))

draw\_learning\_curve(errorT, errorV, "Support Vector Machine", "C", C)

plt.savefig(name)

svm\_clf = svm.SVC(kernel='linear' , C=BestC)

svm\_clf.fit(X\_train, Y\_train)

print("Test")

print("Mejor valor de C : " + str(BestC))

print("Finished fit")

Y\_svm\_predict = svm\_clf.predict(X\_test)

print("Finished predicting")

porcentaje = porcentaje\_aciertos\_SVM(Y\_test, Y\_svm\_predict, True)

print("Recall para la clase 1: " + str(porcentaje))

return porcentaje

name = "SupportVectorMachinesLinear\_0.01\_1.png"

#DataFrame

df\_array = np.array(df)

X = df\_array[:, 1:]

m = np.shape(X)[0]

#Obtencion de la primera columna de la tabla, la de salida

Y = df\_array[:, 0]

n = np.shape(X)[1]

#C = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30, 50]

C = [0.005, 0.01, 0.015, 0.02, 0.03, 0.05, 0.1, 0.3, 0.5, 1]

#C = [0.005, 0.01,0.015]

X\_train,Y\_train, X\_val,Y\_val, X\_test,Y\_test = train\_validation\_test(X, Y, rus)

Main\_svm\_linearKernel(X\_train, Y\_train, X\_val, Y\_val, X\_test, Y\_test, C, name)

Dataset con 11 variables

name = "SupportVectorMachinesLinear\_11VAR.png"

#DataFrame

df\_array = np.array(df\_copia)

X = df\_array[:, 1:]

m = np.shape(X)[0]

#Obtencion de la primera columna de la tabla, la de salida

Y = df\_array[:, 0]

n = np.shape(X)[1]

#C = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30, 50]

C = [0.005, 0.01, 0.015, 0.02, 0.03, 0.05, 0.1, 0.3, 0.5, 1]

#C = [0.005, 0.01,0.015]

X\_train,Y\_train, X\_val,Y\_val, X\_test,Y\_test = train\_validation\_test(X, Y, rus)

Main\_svm\_linearKernel(X\_train, Y\_train, X\_val, Y\_val, X\_test, Y\_test, C, name)

Eliminación de outliers

name = "SupportVectorMachinesLinear\_NO\_OUTLIERS.png"

#DataFrame

df\_array = np.array(data\_clean\_no\_outliers)

X = df\_array[:, 1:]

m = np.shape(X)[0]

#Obtencion de la primera columna de la tabla, la de salida

Y = df\_array[:, 0]

n = np.shape(X)[1]

#C = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30, 50]

C = [0.005, 0.01, 0.015, 0.02, 0.03, 0.05, 0.1, 0.3, 0.5, 1]

#C = [0.005, 0.01,0.015]

X\_train,Y\_train, X\_val,Y\_val, X\_test,Y\_test = train\_validation\_test(X, Y, rus)

Main\_svm\_linearKernel(X\_train, Y\_train, X\_val, Y\_val, X\_test, Y\_test, C, name)

Dataset con 11 variables y eliminación de outliers

name = "SupportVectorMachinesLinear\_11VAR\_NO\_OUTLIERS.png"

#DataFrame

df\_array = np.array(data\_clean\_11var\_noOutliers)

X = df\_array[:, 1:]

m = np.shape(X)[0]

#Obtencion de la primera columna de la tabla, la de salida

Y = df\_array[:, 0]

n = np.shape(X)[1]

#C = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30, 50]

C = [0.005, 0.01, 0.015, 0.02, 0.03, 0.05, 0.1, 0.3, 0.5, 1]

#C = [0.005, 0.01,0.015]

X\_train,Y\_train, X\_val,Y\_val, X\_test,Y\_test = train\_validation\_test(X, Y, rus)

Main\_svm\_linearKernel(X\_train, Y\_train, X\_val, Y\_val, X\_test, Y\_test, C, name)

SVM con Kernel Gaussiano

def Main\_svm\_gaussianKernel(X\_train, Y\_train, X\_val, Y\_val, X\_test, Y\_test, C, sigma, name):

c = 0

sig = 0

Max = 0

BestC = 0

Bestsigma = 0

accAux = 0

errorV = []

errorT = []

Vars = []

accAux = 0

for i in C:

Vars.clear()

errorT.clear()

errorV.clear()

for j in sigma:

Vars.append(str(j))

print("Execution:", i, j)

svm\_clf = svm.SVC(kernel='rbf' , C=i, gamma = (1 / (2 \* j\*\*2)))

svm\_clf.fit(X\_train, Y\_train)

print("Finish fit")

T\_svm\_predict = svm\_clf.predict(X\_train)

errorT.append(porcentaje\_aciertos\_SVM(Y\_train, T\_svm\_predict, False))

print("Finished fit")

Y\_svm\_predict\_val = svm\_clf.predict(X\_val)

porcentaje = porcentaje\_aciertos\_SVM(Y\_val, Y\_svm\_predict\_val, True)

errorV.append(porcentaje)

if porcentaje > Max:

Max = porcentaje

BestC = i

Bestsigma = j

plt.figure().clear()

plt.cla()

plt.clf()

draw\_learning\_curve(errorT, errorV, "C\_" + str(i), "sigma", Vars)

plt.savefig("C\_" + str(i) + ".png")

print("Finished predicting")

#plt.rcParams["figure.figsize"] = (20, 7)

print("Test results for best C: " + str(BestC) + " and best sigma " + str(Bestsigma))

#draw\_learning\_curve(errorT, errorV, "Support Vector Machine", "C", Vars)

#plt.savefig(name)

svm\_clf = svm.SVC(kernel='rbf' , C=BestC, gamma = (1 / (2 \* Bestsigma\*\*2)))

svm\_clf.fit(X\_train, Y\_train)

print("Finished fit")

#plt.rcParams["figure.figsize"] = plt.rcParamsDefault["figure.figsize"]

Y\_svm\_predict = svm\_clf.predict(X\_test)

print("Finished predicting")

porcentaje = porcentaje\_aciertos\_SVM(Y\_test, Y\_svm\_predict, True)

print("Recall para la clase 1: " + str(porcentaje))

return porcentaje

Dataset con 11 variables

name = "SupportVectorMachinesGaussian0.01\_40.png"

#DataFrame

df\_array = np.array(df)

X = df\_array[:, 1:]

m = np.shape(X)[0]

#Obtencion de la primera columna de la tabla, la de salida

Y = df\_array[:, 0]

n = np.shape(X)[1]

C = [0.1, 0.5, 1, 10, 20, 40]

sigma = [1, 5, 10, 20, 30, 40]

X\_train,Y\_train, X\_val,Y\_val, X\_test,Y\_test = train\_validation\_test(X, Y, rus)

Main\_svm\_gaussianKernel(X\_train, Y\_train, X\_val, Y\_val, X\_test, Y\_test, C, sigma, name)

name = "SupportVectorMachinesGaussian\_11VAR.png"

#DataFrame

df\_array = np.array(df\_copia)

X = df\_array[:, 1:]

m = np.shape(X)[0]

#Obtencion de la primera columna de la tabla, la de salida

Y = df\_array[:, 0]

n = np.shape(X)[1]

C = [0.1, 0.5, 1, 10, 20, 40]

sigma = [1, 5, 10, 20, 30, 40]

X\_train,Y\_train, X\_val,Y\_val, X\_test,Y\_test = train\_validation\_test(X, Y, rus)

Main\_svm\_gaussianKernel(X\_train, Y\_train, X\_val, Y\_val, X\_test, Y\_test, C, sigma, name)

Eliminación de outliers

name = "SupportVectorMachinesGaussian\_NO\_OUTLIERS.png"

#DataFrame

df\_array = np.array(data\_clean\_no\_outliers)

X = df\_array[:, 1:]

m = np.shape(X)[0]

#Obtencion de la primera columna de la tabla, la de salida

Y = df\_array[:, 0]

n = np.shape(X)[1]

C = [0.1, 0.5, 1, 10, 20, 40]

sigma = [1, 5, 10, 20, 30, 40]

X\_train,Y\_train, X\_val,Y\_val, X\_test,Y\_test = train\_validation\_test(X, Y, rus)

Main\_svm\_gaussianKernel(X\_train, Y\_train, X\_val, Y\_val, X\_test, Y\_test, C, sigma, name)

Dataset con 11 variables y eliminación de outliers

name = "SupportVectorMachinesGaussian\_NO\_OUTLIERS.png"

#DataFrame

df\_array = np.array(data\_clean\_no\_outliers)

X = df\_array[:, 1:]

m = np.shape(X)[0]

#Obtencion de la primera columna de la tabla, la de salida

Y = df\_array[:, 0]

n = np.shape(X)[1]

C = [0.1, 0.5, 1, 10, 20, 40]

sigma = [1, 5, 10, 20, 30, 40]

X\_train,Y\_train, X\_val,Y\_val, X\_test,Y\_test = train\_validation\_test(X, Y, rus)

Main\_svm\_gaussianKernel(X\_train, Y\_train, X\_val, Y\_val, X\_test, Y\_test, C, sigma, name)

Comparing other algorithms

#DataFrame

df\_array = np.array(df)

X = df\_array[:, 1:]

m = np.shape(X)[0]

#Obtencion de la primera columna de la tabla, la de salida

Y = df\_array[:, 0]

n = np.shape(X)[1]

X\_train,Y\_train, X\_val,Y\_val, X\_test,Y\_test = train\_validation\_test(X, Y, rus)

MLPClassifier

from sklearn.neural\_network import MLPClassifier

modelo=MLPClassifier()

modelo.fit(X\_train, Y\_train)

prediccion = modelo.predict(X\_test)

report = classification\_report(Y\_test, prediccion)

print(report)

cnf\_matrix = confusion\_matrix(Y\_test, prediccion)

print(cnf\_matrix)

Bagging Classifier

from sklearn.ensemble import BaggingClassifier

modelo=BaggingClassifier(n\_estimators = 250, random\_state = 1000)

modelo.fit(X\_train, Y\_train)

prediccion = modelo.predict(X\_test)

report = classification\_report(Y\_test, prediccion)

print(report)

cnf\_matrix = confusion\_matrix(Y\_test, prediccion)

print(cnf\_matrix)

from sklearn.naive\_bayes import BernoulliNB

modelo = BernoulliNB()

modelo.fit(X\_train, Y\_train)

prediccion = modelo.predict(X\_test)

report = classification\_report(Y\_test, prediccion)

print(report)

cnf\_matrix = confusion\_matrix(Y\_test, prediccion)

print(cnf\_matrix)

Random Forest Classifier

from sklearn.ensemble import GradientBoostingClassifier, RandomForestClassifier

modelo = RandomForestClassifier(criterion='gini', random\_state=0)

modelo.fit(X\_train, Y\_train)

prediccion = modelo.predict(X\_test)

report = classification\_report(Y\_test, prediccion)

print(report)

cnf\_matrix = confusion\_matrix(Y\_test, prediccion)

print(cnf\_matrix)

Gradient Boosting

#Gradient Boosting

modelo = GradientBoostingClassifier(loss='deviance', random\_state=1000)

modelo.fit(X\_train, Y\_train)

prediccion = modelo.predict(X\_test)

report = classification\_report(Y\_test, prediccion)

print(report)

cnf\_matrix = confusion\_matrix(Y\_test, prediccion)

print(cnf\_matrix)