Aprendizaje Automático

Problomeas de Clasificación y Regresión

Práctica 3

Juan José Herrera Aranda - juanjoha@correo.ugr.es

5 de junio de 2022

Índice

1. Introducción

2. Problema 1: Bank Marketing

En este caso de estudio tenemos que los datos están relacionados con campañas de marketing (llamadas telefónicas) de una entidad bancaria portuguesa. El objetivo de la clasificación es predecir si el cliente se subscribirá a un depósito a plazos. El dataset consta de un número de 49732 instancias con 17 atributos contando la clase. Los datos se pueden agrupar en cuatro categorías: datos bancarios el cliente, relacionados con el último contacto de la campaña actual, otro tipo de atributos y atributos sociales y económicos.

- Datos bancarios del cliente
  - Age: (Numérico)
  - Job: (Categórico) Tipo de trabajo
  - Marital: (Categórico) Estado civil
  - Education: (Categórico) Educación
  - Default: (Categórico) Si tiene crédito en mora ('yes','no','unknown')
  - Housing: (Categórico) Si tiene préstamos de la vivienda ('yes","no','unknown')
  - Loan: (Categórico) Si tiene préstamos personales ('ye','no','unknown')
- Relacionados con el último contacto de la campaña actual
  - Contact: (Categórico) tipo de comunicación del contacto ('Celular','Telephone')
  - Month: (Categórico) mes del año del último contacto
  - Day_of_the_week: (Categórico) día de la semana del último contacto
  - Duration: duración del último contacto, en segundos (numérico). Nota importante: este atributo afecta en gran medida al objetivo de salida (por ejemplo, si la duración=0 entonces y='no'). Sin embargo, la duración no se conoce antes de realizar una llamada. Además, después de la finalización de la llamada la clase 'y' es obviamente conocido. Por lo tanto, esta entrada sólo debería incluirse con fines de referencia y debería descartarse si la intención es tener un modelo de predicción realista.
- Otros atributos
  - Campaing: (Numérico, incluye último contacto) número de contactos realizados durante esta campaña y para este cliente
  - pdays: (Numérico; 999 significa que el cliente no fue contactado previamente) número de días transcurridos desde que el cliente fue contactado por última vez en una campaña anterior
  - previous: (Numérico) número de contactos realizados antes de esta campaña y para este cliente
  - Outcome: (Categórico, 'fracaso', 'inexistente', 'éxito') resultado de la campaña de marketing anterior
  - Balance: (Numérico) No se especifica nada sobre este atributo
- Variable Salida
  - y: (Categórico) Se ha subscrito el cliente al depósito? (ɀes","no")

Estamos ante un problema de aprendizaje supervisado en el que el conjunto $\mathcal{X}$ está formado por una matriz, cuyas filas son las instancias y cuyas columnas son las características de los datos. El conjunto de etiquetas $\mathcal{Y}$ está formado por un vector en el que cada casilla solo puede tomar dos valores *si* y *no*. Aunque más adelante convertiremos esas opciones en numéricas ($\{1, -1\}$. La función objetivo f será la que, dado un vector de características del espacio $\mathcal{X}$, determina la clase, esto es, especifica si el cliente se ha subscrito a la campaña o no.

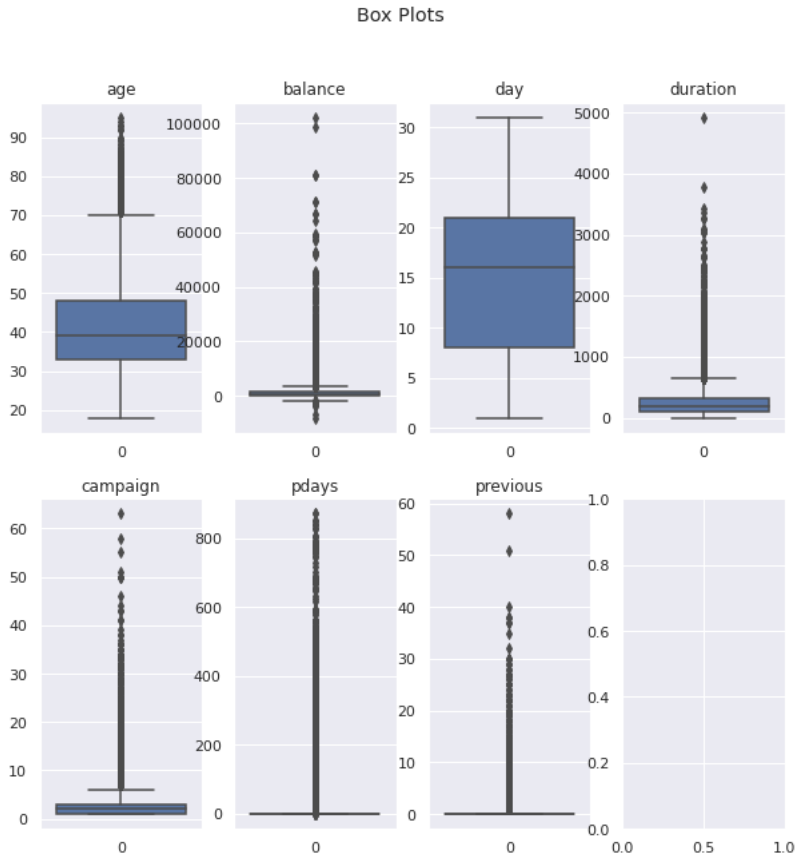

Figura 1: Boxplots para las variables numéricas

Más adelante comentaremos el procedimiento realizado para la división en conjuntos de training y test. Pero ahora vamos a centrarnos en la visualización del conjunto de entrenamiento. Primeramente vamos a comentar el desbalanceo entre clases que tenemos presente, lo podemos apreciar en el countplot de la figura 2 .En la figura 1 presentamos los boxplots de las variables numéricas. Podemos apreciar que por ejemplo, el atributo *age* tiene una media de 40 años y la mitad de la población se sitúan en personas de entre aproximadamente 30 y 50 años. Los más jóvenes rondan los 20 y podemos apreciar algunos valores atípicos en edades. como por ejemplo, personas de 70 para arriba. De entre todos podemos observar un comportamiento especialmente raro en *balance, duration, campaign, pdays y previous*, ya que presentan demasiados valores atípicos.

En cuanto a las variables categóricas, se han empleado *countplots* para visualizarlas. Podemos observar que la variable *housing* es la única cuya proporción entre las modalidades (valores que toma la variable) es la más similar. El resto tiene una proporción muy desigual entre los distintos valores que toma. Por ejemplo, la característica *default* tiene dos modalidades, *yes* y *no*, pero en la mayoría de los casos vemos que casi todas las instancias tienen *no*, de hecho, aproximadamente algo más del 98 % de los datos de entrenamiento tienen la modalidad *no*. Estudiando un poco

más esta variable (Fig. 4), notamos que todas las instancias que tienen la modalidad *yes* tienen la etiqueta *-1*.

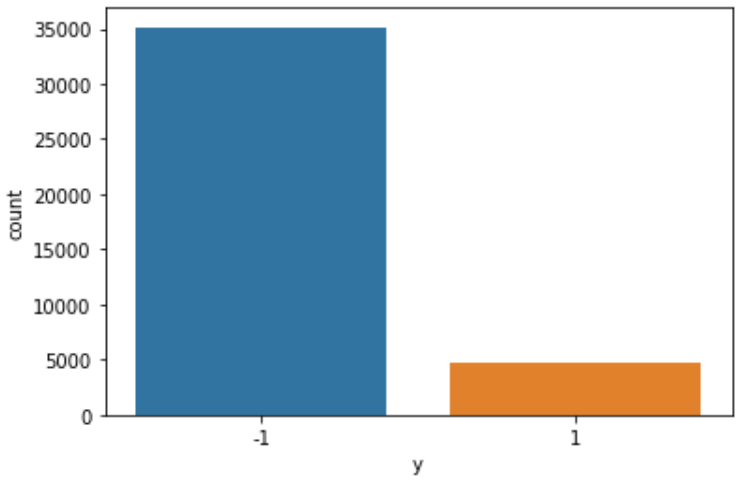

Figura 2: Count plot que muestra el desbalanceo de la de clases

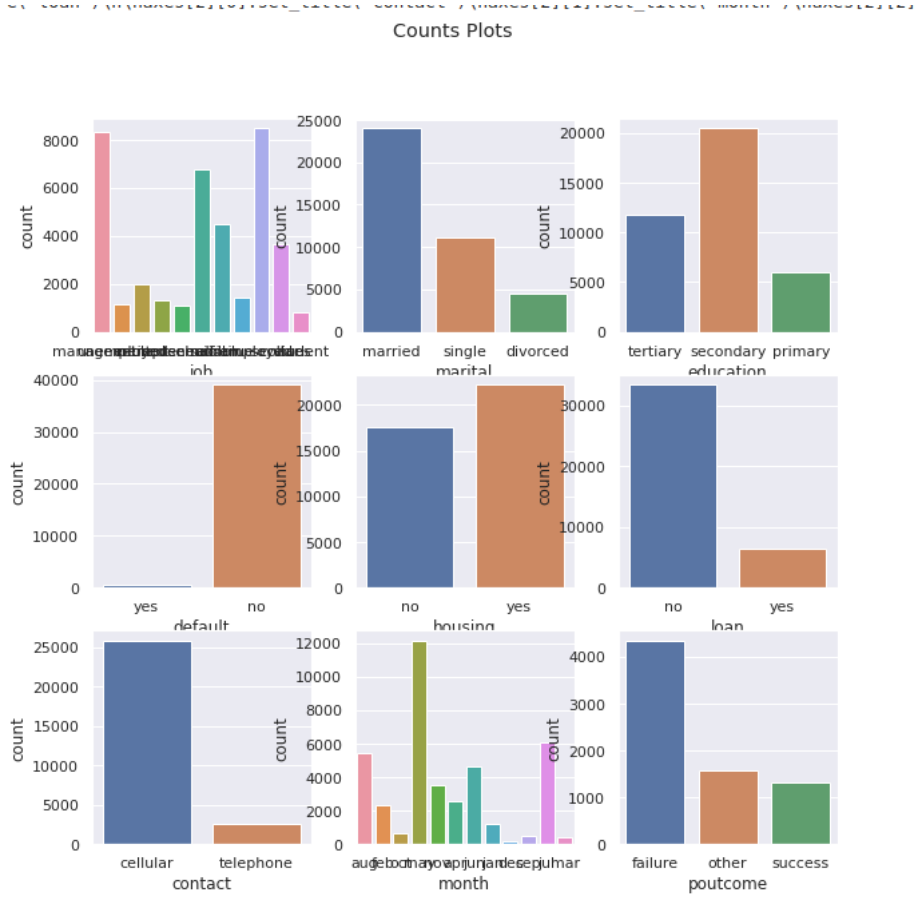

Figura 3: Counts plots para las variables categóricas.

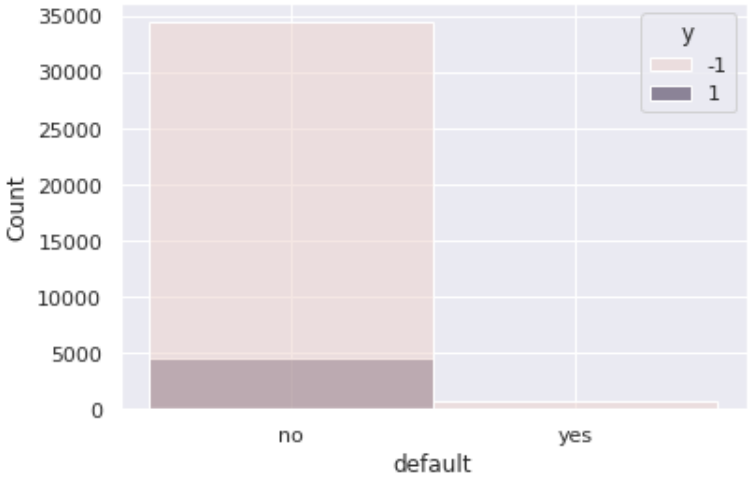

Figura 4: displot para la característica default

2.1. Preprocesado

El preprocesado es una de las partes más importantes antes de aplicar las técnicas de Machine Learning. Para obtener resultados de calidad necesitamos datos de calidad y para tener datos de calidad debemos de hacer un preprocesado para ponerlos a puesta en escena de la mejor manera. El preprocesado es la parte que lleva más tiempo a la hora de enfrentarse con un problema de Machine Learning y no es nada fácil. El preprocesado que realizaremos será uno más o menos básico ya que no es el principal objeto de esta práctica. El orden de las distintas partes del preprocesado se puede seguir en las siguientes subsecciones.

2.1.1. Reducción de la dimensión

Primeramente estudiamos las correlaciones entre las variables numéricas. Si dos variables están correladas o tienen un fuerte grado de correlación, quiere decir que la información que explica una se podría explicar con la otra y vivecersa. Se ha usado el coeficiente de correlación de Pearson. Este coeficiente está comprendido en el intervalo $[-1, 1]$. El valor 0 significa que no hay correlación lineal entre las variables. el valor 1 significa que hay una correlación positiva perfecta y cuando es -1 indica que hay una correlación negativa perfecta. Cuánto más alejado esté del 0 más correlación habrá entre las variables. Obviamente, una variable tiene una correlación positiva perfecta consigo misma.

En la figura 5 vemos que no hay variables numéricas correladas, la que más es una correlación de 0.5 entre *pdays* y *previous* pero no es lo suficiente como para decir que hay correlación entre ese par.

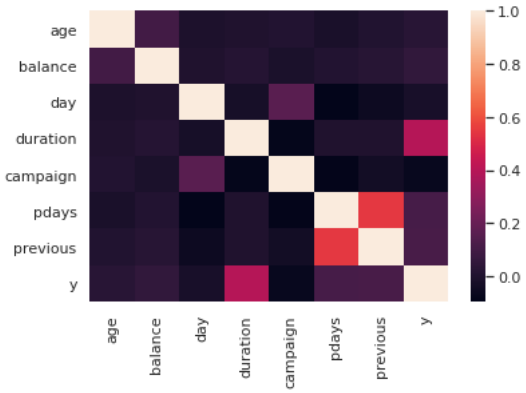

Figura 5: Mapa de calor que muestra la correlación entre las variables numéricas. Hemos usado la variable 'y' que es la etiqueta para ver si hay correlación entre ella y alguna característica. Si la hubiera significa que existe una variable que explica bastante bien la salida y le haríamos especial hincapié. Obviamente, sería criminal eliminar una característica que estuviera bastante correlación con la etiqueta.

Como antes se comentó, la variable *poutcome* tiene un porcentage de datos perdidos algo más del 98 % por lo que vamos a eliminar esta variable del conjunto de datos de training y test, ya que no aporta prácticamente nada de información.

2.1.2. Valores perdidos

En la variables numéricas no tenemos valores perdidos pero sí los tenemos en las categóricas. Ocurre que en la variable *job* hay 260 instancias con valores perdidos, la variable *education* tiene 1651 instancias con valores perdidos y la variable *contact* 11492.

Las dos primeras variables tienen pocos valores perdidos y los hemos imputado por la moda, esto es, por la categoría mayoritaria, ya que si la mayoría de valores están en una categoría, hay más probabilidad de pertenecer a esa categoría. El valor más frecuente en la variable *job* es *blue-collar* y en la variable *education* es *secondary*. En cambio, la variable *contact* tiene demasiado valores perdidos y si lo imputamos por la moda estaríamos introduciendo sesgo. Por tanto, vamos a considerar todos esos valores perdidos como una nueva categoría llamada *new_type*.

2.1.3. Codificación

En este dataset contamos con características de tipo categórico, por lo que hemos empleado la técnica llamada **One Hot Encoder** disponible en *sklearn*. Su funcionamiento es sentillo, si tenemos n categorías en la variable, se crea un vector de longitud n donde cada posición está asociada a una categoría. Todos los valores del vector son cero excepto un uno en la posición de la categoría e indica que el vector pertenece a esa categoría. Ocurre que por cada variable que tengamos vamos a añadir tantas columnas en la matriz de datos como número de categorías tenga la variable para poder codificarla, por lo que va a aumentar el número de variables en términos generales, pasamos de tener 15 características a 45 (sin contar la etiqueta). Las características a codificar son: job, marital, education, default, housing, loan, contact y month.

2.1.4. Tratamiento de outliers.

Los outliers o datos anómalos son observaciones extremadamente grandes o pequeñas en comparación con el resto de la distribución de la varaible a la que pertencen. Son un arma de

doble filo y deben ser tratados correctamente. Siguen siendo datos, por lo que aportan información y eliminarlos puede suponer no obtener información sobre el suceso que modelen obteniendo así un modelo que no llegue a modelar todo lo que se pretende. Pero por otra parte, no tratarlo y dejarlos, puede provocar distorsiones en el aprendizaje de los modelos llegando a obtener modelos malos. En general los outliers pueden deberse a errores de mediciones en cuyo caso sí sería convenientes eliminarlos, o que sean observaciones que realmente tengan importancia en la población en cuyo caso deben de ser tratados como el resto de datos. El correcto tratamiento de los outliers es bastante complejo ya que hay que estudiar variable por variable aplicando test de hipótesis los cuales nos dicen en términos de probabilidad si éstos son importantes o no. Sin embargo, este estudio se escapa de los objetivos de esta práctica y haremos uno más laxo.

Detectar outliers es muy sencillo e intuitivo si graficamos la distribución de los datos por ejemplo en un boxplot como el de la figura 1, en donde podemos apreciar que las variables con outliers son: *age, pdays, duration, campaign, pdays y previous.* Sin embargo, vamos a recurrir a una técnica algo más avanzada para la detección de outliers pues nuestro objetivo no es deshacernos de todos ellos.

Vamos a emplear un algoritmo conocido como Local Outlier Factor (LOF). Es un algoritmo de aprendizaje automático semi-supervisado y calcula la desviación de la densidad local de cada punto con respecto a sus vecinos más cercanos. Considera como outliers aquellos ejemplos que presentan, con batante diferencia, una menor densidad que sus vecinos. La densidad local se obtiene de los k vecinos más cercanos. El valor del LOF de una observación es igual al ratio de la media de la densidad local de sus k vecinos más cercanos y su propia densidad local. Se espera que una instancia normal tenga una densidad local similar a sus vecinos, mientras que para un outliers se espera una menor densidad.

Este método tiene como parámetros el número de vecinos más cercanos, hemos considerado 20 que es el valor que tiene por defecto y suele funcionar bastante bien. También podemos elegir el algoritmo que usará la función por dentro para calcular los vecinos más cercanos, y como valor hemos seleccionado la opción *auto*, esto es, la propia función se encarga de decidir qué algoritmo irá mejor en función de los datos. Como distancia hemos elegido la euclídea, esto se consigue estableciendo los parámetros *metric='mikonski'* y *p=2*. Por último, podemos indicar el nivel de contaminación del datset, esto es, la proporción de outliers en los datos. Cuando el algoritmo se entrena esto se usa para definir un umbral influye en los resultados. Hemos usado la opción *auto* para que el propio algoritmo calcule la contaminación.

Este algoritmo usa métodos de cálculo basados en distancias, por lo que sería recomendable tener los datos normalizados previamente para que el factor de la escala influya lo menos posible. Notamos que este método solo detecta outliers pero no los elimina. La eliminación la hacemos a mano en el código. La cantidad de outliers encontrada la presentamos en la tabla 1. Vamos a proceder a eliminarlos, pero no los vamos a eliminar todos, vamos a tener en cuenta el desbalanceo enorme que tenemos entre clases, y solo eliminaremos aquellas instancias con outliers que pertenezcan a la clase mayoritaria, es decir a la clase negativa. En la tabla 2 mostramos las instancias con outliers pertenecientes a cada clase.

Variable	Número de Outliers
age	56
balance	1735
duration	519
campaign	28
pdays	519
previous	13

Tabla 1: Outliers enccontrados por el método LOF en los datos de entrenamiento

Total Outliers	2799
Outliers class yes	368
Outliers class no	2461

Tabla 2: (Primera Columna) Número total de outliers. Si una instancia tiene maś de una característica con outliers solo la contabilizamos una vez. (Segunda columna) Número total de instancias con outliers pertenecientes a la clase positiva. (Tercer Columna) Número total de instanccias con outliers perteneccientes a la clase negativa.

2.1.5. Mutual Information

Pueden existir características que no aporten información a la etiqueta, es importante detectar cuales son y eliminarlas en dicho caso ya que si no, se introduce ruido. Para ello, usamos lo que se conoce como información mutua que nos dice la cantidad de información que obtenemos de una variable observando la otra y no tiene por qué ser de manera lineal.

Vamos a usar la función *mutual_info_classif* de *sklearn* para determinar el valor de MI entre dos variables. Cuanto más cercano esté el valor a cero tenemos que las variables son independientes y cuando más dependencia haya, más grande será. Este método funciona por dentro haciendo uso de los vecinos más cercanos para estimar el valor. En este caso hemos tomado como parámetro *n_neighbors=3* que es el valor por defecto y según se comenta en el paper del método, suele ir bien.

Hay que tener en cuenta que este método trabaja con atributos numéricos, por tanto no vamos a tener en cuenta a los categóricos por dos motivos. El primero es que tendríamos que darle un valor numérico a los atributos categóricos sin usar one hot encoding, lo que originaría una relación de orden que realmente no existe y estaríamos contaminando el método. El segundo es que al aplicar one hot encoding, vamos a tener tantas variables auxiliares como cardinalidad tenga la variable original, por lo que si alguna de las variables auxiliares no aportara información, sería un error eliminarla, pues estaríamos quitando información de la variable original.

Tras aplicar el método no se ha observado una dependencia significativa respecto alguna variable, de hecho, la gran mayoría aportan poca información tal y como se puede observar en la tabla 3.

age	0.012
balance	0.023
day	0.0096
duration	0.081
campaign	0.005
pdays	0.029
previous	0.016

Tabla 3: Información mutua de las variables de tipo numérico. Se han obviado las categóricas.

2.1.6. Normalización

Un paso fundamental en los problemas en los que se vayan a usar modelos con variables numéricas es la normalización de los datos. Esta parte influye mucho en los modelos ya que por ejemplo, SVM usa distancias de los puntos al hiperplano separador y es necesario que los datos estén normalizados para que la solución sea correcta, ya que en caso contrario, tendríamos unas características con más peso que otras en función de la escala. Algo similar ocurre con las redes neuronales y otros métodos de Machine Learning, aunque no en todos.

Hay muchos tipos de normalización, y en este caso hemos optado por transformar los datos para que tengan desviación típica uno y media cero. Este tipo de normalziación se conoce como *z-score*. Si X es la variable a normalizar, entonces la variable normalizada quedaría como

$$Z = \frac{X - \mu}{\sigma}$$

donde μ y σ son la media y la desviación típica de la variable respectivamente.

El procedimiento para normalizar se puede hacer de diversas maneras, usando scipy.stats.zscore, sklearn.preprocessing.StandardScale con los parametros adecuados o simplemente aplicar la fórmula columna por columna. En la realización de la práctica he optado por esta tercera opción.

2.2. Selección del conjunto de hipótesis a usar

Para este problema hemos usado modelos lineales que incluyen, el Perceptron, PLA Pocket, regresión logística, y también se ha usado SVM con kernel lineal. Cada uno de estos modelos tienen una clase de funciones, pues al fijar un modelo, se fija su propia clase de hipótesis.

Para PLA, PLA Pocket tenemos como conjunto de hipótesis $\mathcal{H}_{pla} = \mathcal{H}^1_{pla} \cup \mathcal{H}^2_{pla}$ donde $\mathcal{H}^1_{pla} = \{signo(p(x_1, ..., x_n)) : p \in \mathbf{P}^1[x]\}$ son los polinomios de orden uno y $\mathcal{H}^2_{pla} = \{signo(p(x_1, ..., x_n)) : p \in \mathbf{P}^2[x]\}$ son los polinomios de orden dos. Para que se entienda de manera más simple, podemos imaginarnos que tenemos un conjunto de datos con tres características, llamémoslas x_1, x_2, x_3, entonces los polinomios de orden uno serían de la forma $p(x) = w_1 \cdot x_1 + w_2 \cdot x_2 + w_3 \cdot x_3$ y los de orden dos serían de la forma $p(x) = w_1 \cdot x_1^2 + w_2 \cdot x_2^2 + w_3 \cdot x_3^2 + x_1x_2 \cdot w_4 + x_1x_3 \cdot w_5 + x_2x_3 \cdot w_6 + w_7 \cdot x_1 + w_8 \cdot x_2 + w_9 \cdot x_3$, donde w_i son los pesos. Esto se ha llevado a cabo usando el método *PolynomialFeatures* implementado en *sklearn*. No hemos incluido el término de sesgo *include_bias=False* por ser una constante igual a todas las instancias. También hemos usado regresión logística y LinearSVC, obviamente cada modelo tiene su propia clase de hipótesis. A la regresión logística le hemos pasado las variables tal cual, sin transformaciones polinómicas por problemas de tiempo de cómputo, por lo que el conjunto de hipótesis será el mismo que $\mathcal{H}^1_{pla}$ pero sin la función signo. Para el modelo LinearSVC se han

usado tanto las variables del problema como la transformación polinómica de orden dos. Por tanto tendremos también dos conjuntos de hipótesis a los que denotaremos $\mathcal{H}^1_{svm}$ y $\mathcal{H}^2_{svm}$

La elección de este conjunto de hipótesis ha sido porque nos enfrenamos a un problema de ML algo más real que las prácticas anteriores y por ende, mucho más complejo, entonces se ha pensado que el añadir características cuadráticas puede ayudar bastante a combatir la dificultad del problema. Hay que tener en cuenta que el número de variables en la clase $\mathcal{H}^2_{pla}$ es el cuadrado del número de variables en $\mathcal{H}^1_{pla}$. Así que la dimensión de los datos aumenta bastante y puede suponer un contratiempo con respecto al tiempo de ejecución, ya que se disponen de unos recursos hardware limitados.

En teoría se vieron tres criterios de selección para la mejor hipótesis. El primero es el criterio ERM, que consiste en seleccionar la hipótesis con menor riesgo empírico, es decir, aquella que obtenga un menor E_{in}. En nuestro caso, descartamos esta primera opción ya que no tiene en cuenta el sobreajuste y nosotros al considerar la clase de polinomios de grado dos, estamos aumentando la complejidad aumentando así la posibilidad de sobreajuste. La segunda es el criterio SRM que es pragmático cuando tenemos como conjunto de hipótesis una unión encajada y consiste en ajustar una función dentro de cada clase seleccionando como ganadora aquella que tenga menor cota de error E_{out}, es decir, $E_{in} + O(d_{vc})$. Esta segunda opción también la descartamos pues no se tiene en cuenta la regularización cosa que nosotros si tenemos en cuenta. La última consiste en aplicar SRM más regularización, se elige dentro de cada clase la función que minimiza E_{aug}, en cuyo caso se estaría minimizando lo que se conoce como sesgo-varianza. Vamos a usar esta última pero en vez de quedarnos con la hipótesis que tiene menor E_{aug}, nos vamos a quedar con la hipótesis que tiene mejor E_{val} en validación cruzada, esto es, E_{cv}

2.3. Conjuntos de training, test y validación

El dataset nos presentaba dos conjuntos de datos separados, uno para training y otro para test, sin embargo los hemos juntado en uno y hemos realizado nuestra propia partición de datos. La partición ha consistido en dejar un 80 % de los datos para training y el resto para test. Como tenemos un notorio desbalanceo de clases, la partición se ha hecho de manera estratificada con respecto a la etiqueta, esto quiere decir que se ha mantenido la misma proporción de los valores de las etiquetas en ambos conjuntos. Para esto se ha usado la función *train_test_split* implementada en *sklearn*. El argumento *stratify* es el que se encarga de hacer el muestreo estratificado, y basta con indicarle la columna de la etiqueta para que lo realice.

Para el tuneamiento de los parámetros hemos usado el método *GridSearchCV* de *sklearn*, dentro de este método se usa cross-validation de manera estratificada por defecto (si la etiqueta es binaria, que en nuestro caso así es). Como conjunto de validación hemos usado el 20 % del conjunto de training.

Tabla 4: Particion del conjunto de datos

	Training	**Test**	**Val**
Yes	4183	1162	465
No	29687	8785	3299
Total	33870	9947	3764

2.4. Métrica de error

En este problema queremos predecir si el cliente se ha subscrito o no al depósito. Teniendo en cuenta que las clases están desbalanceadas a favor de la clase negativa y que a la empresa le interesa más la clase positiva, debemos de elegir una métrica que tenga en cuenta esta situación. Los falsos positivos son perjudiciales para la empresa ya que si se predice que un cliente se va a subscribir pero luego no lo hace, es dinero que se pierde, por otra parte, los falsos negativos son también perjudiciales ya que si se predice que un cliente no se va a subscribir pero luego sí lo hace, se le sigue bombardeando con publicidad para convencerlo pudiendo provocar un enfado y que al final no se subscriba de verdad. Así que de alguna manera hay que penalizar esto. Para ello, usamos la métrica conocida como *F1-score* la cual viene dada por la expresión,

$$F1 = \frac{2TP}{TP + FP + FN}$$

La métrica por antonomasia usada es la precisión o *accuracy* pero en este caso no la usaremos ya que si por ejemplo, todas las instancias se predijeran de la clase negativa tendríamos una accuracy igual a la proporción de elementos de la clase negativa, que al ser muchos, sería un valor elevado y se podría pensar que tendríamos buenos resultados siendo falso.

No hay que confundir la métrica de error, que sirve para medir el rendimiento del modelo, con la función de pérdida que es la que se usa para optimizar el modelo. La función de pérdida que vayamos a usar en el caso de la clasificacións será la propia de cada modelo. Para RL será la entropía cruzada, para PLA será la el número de instancias mal clasificadas,...

2.5. Regularización

Tenemos como clase de funciones la unión de otras dos clases y como vamos a usar el criterio SRM vamos a seleccionar de cada clase la mejor hipótesis y nos quedaremos con la mejor todas. Para la clase de los polinomios de grado 2, al aumentar el número de variables aumenta la varianza provocando así un aumento del ruido determinista, por lo que el sobreajuste es probable, así que resulta conveniente incluir algún tipo de regularización.

En general hay tres tipos de regularización, la del tipo Lasso (L1), la del tipo ridge (L2) y una mezcla de ambas (ElasticNet). La regularización L1 nos va a servir de ayuda cuando sospechamos que varios de los atributos de entrada son irrelevantes. Al usar esta regularización estamos favoreciendo que algunos de los coeficientes acaben valiendo cero. Es útil para describir cuáles de los atributos de entrada son relevantes y en general para obtener un modelo que generalice mejor. Nos puede ayudar en hacer la selección de los atributos de entrada y funciona bien cuando los atributos no están muy correlados entre ellos. La regularización L2 nos va a venir bien cuando sospechemos que varios de los atributos de entrada están correlados entre ellos. Esta regularización hace que los coeficientes acaben siendo más pequeños minimizando así el efecto de la correlación entre los atributos y hace que el modelo generalice mejor. Esta regularización viene bien cuando la mayoría de los atributos son relevantes. La regularizacion ElasticNet es cuando tengamos un gran número de atributos. Algunos de ellos serán irrelevantes y otros estarán correlados entre ellos. En estos casos no hemos usado ElasticNet porque es incompatible con otros parámetros y no todos los modelos lo tienen implementado.

Sabemos que los atributos numéricos no están apenas correlados, sin embargo, los categóricos fueron codificamos con one hot encoding y se añadieron bastantes variables a los datos por lo que puede ser posible que existan correlaciones. Además, gracias a la tabla 3 vemos que todos los atributos (en este caso los numéricos) son igual de importantes. Así que en principio podemos usar tanto L1 como L2. Según el modelo que usemos vamos a incluir la regularización como

hiperparámetro, por ejemplo, el caso de PLA. En otros modelos, como la regresión logística vamos a fijar la regularización de tipo L2, ya que algunos de los hiperparámetros que usamos son incompatibles con otras regularizaciones.

2.6. Modelos empleados e hiperparámetros a tunear

2.6.1. PLA y Regresión Logística

Hemos empleado como modelos lineales el PLA, PLA Pocket y la regresión logistica. No vamos a entrar en detalle con la explicación de estos métodos por estar explicados en prácticas anteriores. Solo comentar que *sklearn* no tiene implementado PLA Pocket y nosotros lo hemos simulado empleando la técnica Early Stopping aplicada al PLA. Vamos a presentar ahora los parámetros que hemos usado en los algoritmos anteriores tanto para el tuneamiento como en la declaración del método.

Los parámetros usados en PLA han sido:

- *penalty*: hace referencia a la regularización
- *alpha*: constante que multiplica el término de regularización
- *l1_ratio*: Es un parámetro relativo a elasticnet, pero como no se ha usado no le damos importancia.
- *validation_fraction*: Tamaño para la partición de validación si se usa early_stopping. Hemos seleccionado 0.1 que es el valor por defecto ya que los datos se diezmaron cuando le quitamos el conjunto de test y preferimos no disminuir mucho más el conjunto de entrenamiento.
- *Early Stopping*: indica si aplicamos o no dicha técnica
- *max_iter*: Número máximo de iteraciones, hemos usado 100000
- *tol*: Otro criterio de parada. Si la diferencia entre una iteración y la anterior no mejora una tolerancia, entonces para. Hemos usado una tolerancia de $10e-5$ ya que una tolerancia grande hace que el algoritmo realice muy pocas iteraciones.
- *n_iter_no_change*: Número de iteraciones que continua en caso de que no mejore estando el parámetro *early_stopping* activado.
- *random_state*: Especificar la semilla
- *shuffle:* Indica si se desordena o no los ejemplos de entrenamiento. Lo hemos puesto a True

de los cuales hemos usado para el tuneamiento *penalty*, *alpha* y *early_stopping* por ser los más importantes.

Los parámetros usados para Regresión Logística han sido:

- *penalty*, *max_iter*, *tol*, *random_state*, *class_weight* explicados anteriormente y cuyos valores se han mantenido a excepción de *penalty*. Se ha usado la regularización L2 porque presenta compatibilidad con todos los algoritmos de optimización, L1 no la presenta.
- *multi_class*: Parámetro para indicar que use la técnica one-versus-rest. Como estamos en un problema de clasificación binario, es indiferente.

- *solver*: Algoritmo usado para resolver el problema de optimización.
  - *newton-cg*: Es el método de Newton que usa la Hessiana
  - *lbfgs*: Es una modificación del método de Newton.
  - *Sag (Stochastic Average Gradient)*: Es una variación del gradiente descendentes estocástico donde se usa información de las iteraciones previas para aumentar la rapidez de la convergencia
  - *liblinear*: Aplica un algoritmo llamado descenso coordinado
- *fit_intercept* Dejamos el valor por defecto ya que este indica que se añada un término de sesgo igual a uno.
- *warm_start*: Este parámetro solo sirve para el solver liblinear y hemos decidido dejarlo a False (por defecto) ya que vamos a usar mas solvers.
- C: Especifica el efecto de la regularización, mayores valores implican un menor efecto y valores mas pequeños implican un mayor efecto.

de los cuales se han usado para el tuneamiento C y *solver*. Hemos usado estos parámetros, ya que en términos generales son los más importantes del algoritmo.

2.6.2. SVM

Esta técnica se basa en la construcción de un hiperplano separador entre los datos con el fin de clasificarlos. Se pretende que el hiperplano sea lo más ancho o gordo posible con el objetivo de maximizar la distancia entre los puntos más cercanos de dos clases diferentes. Esta técnica tiene dos enfoques, la primal y la dual. El primero es el que vamos a usar nosotros ya que el segundo enfoque se recomienda cuando el número de características sea mayor al número de ejemplos, cosa que no ocurre en nuestro conjunto de datos.

En este modelo hemos usado la regularización L2 y comentamos que es muy importante el papel que juega el parámetro C, que es el peso de la penalización porque un punto quede dentro del margen de separación o esté clasificado erróneamente. Este parámetro se describe en la documentación de *sklearn* como la inversa de la influencia de la regularización, tal y como explicamos en RL.

Los parámetros usados en este algoritmo son similares a lo de regresión logística por lo que no vamos a comentarlos todos. Solo decir que al tratarse del método *SVCLinear* se ha usado como kernel el lineal. La función de perdida hemos usado *squared_hinge* por ser una función de pérdida diferenciable. La otra alternativa era *hinge* pero no es diferenciable. Hemos indicado *dual=False* para usar el enfoque primal. El algoritmo de optimización usado (parámetro *solver*) es *liblinear* el cual ya hemos hablado de él antes. Solo vamos a usar para el tuneamiento el valor de la constante de complejidad, C por ser un hiperparámetro crucial en la regularización.

2.7. Análisis y Selección de las mejores hipótesis

Primeramente vamos a seleccionar los modelos con los mejores hiperparámetros. Para ello vamos a emplear el error de validación aplicando la técnica *Cross Validation* con 5 particiones. El error de validación final es la media de los cinco errores que tengamos, pero no vamos a usar el error tal cual para seleccionar el mejor modelo, si no que emplearemos la métrica que venimos usando en el conjunto de validación.

Como antes se ha comentado, hemos simulado PLA Pocket con el parámetro *early_stopping=True*, pero vamos a tratar estos dos algoritmos como uno solo. Los hiperparámetros con los que obtenemos el mejor modelo lo podemos apreciar en la tabla 5. Observamos que la mejor hipótesis en $\mathcal{H}^1_{pla}$ tiene una F1 score un poco pobre, no tan alta por ejemplo que la mejor hipótesis en la clase $\mathcal{H}^2_{pla}$ (Tabla 6). Estos resultados son bastante significativos ya al aumentar la complejidad del modelo hemos obtenido buenos resultados. Al aumentar la complejidad estamos a la vez aumentando la varianza y disminuyendo el sesgo. En este caso resulta beneficioso ya que obtenemos mejores resultados y no hay indicios de sobreajuste, que es lo que puede ocurrir al aumentar la complejidad del modelo.

Tabla 5: Tuneamiento para el PLA en el conjunto de hipótesis H^1_{pla}

alpha	**early_stopping**	**penalty**	**F1**	**rank_test_score**
0,0001	False	l1	0,36359	3
0,0001	False	l2	0,32616	6
0,0001	True	l1	0,35102	4
0,0001	True	l2	0,38917	2
0,001	False	l1	0,39201	1
0,001	False	l2	0,23872	8
0,001	True	l1	0,29625	7
0,001	True	l2	0,33129	5

Tabla 6: Tuneamiento para el PLA en el conjunto de hipótesis H^2_{pla}. En este caso no hemos usado early stopping por temas de tiempo de cómputo

alpha	**penalty**	**F1**	**rank_val_score**
0,0001	l1	0,5059	1
0,0001	l2	0,2999	4
0,001	l1	0,4213	2
0,001	l2	0,3949	3

Usando el modelo LinearSVC vemos que los mejores parámetros los obtenemos empleando *C=10000* (Tabla 7). Este valor tan elevado del parámetro C nos viene a decir que la regularización apenas ha tenido efecto por ser inversamente proporcional al efecto de ésta. Cuando aumentamos la complejidad del modelo (Tabla 8) observamos algo un tanto extraño. El mejor resultado se obtiene usando $C = 1$, lo que indica que la regularización ha jugado un papel importante. Sin embargo, si quitamos importancia a ésta aumentando el valor de C vemos que los resultados siguen siendo buenos y al disminuir el valor de C, vemos que los resultados empeoran. Así que tampoco podemos afirmar con certeza la mucha o poca importancia de la regularización en este modelo.

Tabla 7: Tuneamiento para el LinearSVC en el conjunto de datos

C	F1	rank_val_score
0.0001	0.20780706001175558	5
0.01	0.34583326064397435	4
1.0	0.3605655854115314	2
100.0	0.3605655854115314	2
10000.0	0.36081196251741265	1

Tabla 8: Tuneamiento para el LinearSVC usando como entrada características polinómicas de orden dos

param_C	F1	rank_test_score
0.0001	0.34380820876208795	5
0.01	0.5010389765471155	4
1.0	0.5060896734895192	1
100.0	0.5050788630984132	2
10000.0	0.5048827590552911	3

Presentamos ahora el tuneamiento de los parámetros para el modelo de regresión logística, como ya se comentó, no se han usado transformaciones polinómicas. Hemos empleado una gran cantidad de hiperparámetros ya que la enjundia radica en el tipo de algoritmo usado para la optimización. Los resultados en términos generales no son buenos, y observamos que usando $C = 10$ obtenemos los mismos resultados sea cual sea el algoritmo de optimización. Puede parecer un tanto extraño, sin embargo, podemos elucubrar que por esa zona se llega a un mínimo local y se quedan atrapados en parte debido al efecto de las restricciones en los pesos a causa de la regularización impuesta.

Tabla 9: Tuneamiento de los hiperparámetros para la regresión logística usando como clase de hipótesis la de los polinomios de grado 1

C	solver	F1	rank_test_score
0.1	newton-cg	0.4038831166957772	11
0.1	lbfgs	0.4038831166957772	11
0.1	liblinear	0.40364977227918813	15
0.1	sag	0.40382817690326434	13
0.1	saga	0.40382817690326434	13
1	newton-cg	0.41274278712167967	6
1	lbfgs	0.41274278712167967	6
1	liblinear	0.41262222336066207	10
1	sag	0.41274278712167967	6
1	saga	0.41268083369761166	9
10	newton-cg	0.41339327105390417	1
10	lbfgs	0.41339327105390417	1
10	liblinear	0.41339327105390417	1
10	sag	0.41339327105390417	1
10	saga	0.41339327105390417	1

Presentamos en la tabla de más abajo las mejores hipótesis de cada modelo, observamos que la mejor es la del modelo LinearSVC usando como características de entrada una transformación polinómica de orden dos.

Tabla 10: Mejores hipótesis de cada modelo

Modelos	**F1_cv**	**Ranking**
PLA-1	0,3900	4
PLA-2	0,5059	2
LinearSVC-1	0,3600	5
LinearSVC-2	0,5060	1
Regresión Logística	0,4133	3

2.8. Evaluación en el conjunto de test

Ya tenemos la mejor hipótesis seleccionada, *LinearSVC-2*, ahora lo que vamos a hacer es estimar E_{out} usando E_{test}.

$$E_{out}(g) \leq E_{test}(g) + \sqrt{\frac{1}{2N}\log\frac{2}{\delta}} = E_{test}(g) + \sqrt{\frac{1}{2 \times 33870}\log\frac{2}{0{,}05}} \approx 0{,}5041 + 0{,}0073 \approx 0{,}5114$$

con probabilidad al menos 0,95. A la hora de escoger el mejor modelo se usó la métrica $F_1 score$ por lo que elegimos el modelo que maximizaba esa métrica. Ahora usamos como error $1 - F_1 score$ para que el problema se convierta en uno de minimización y podamos aplicar correctamente la desigualdad de Hoeffding. Calculamos también las cotas del resto de modelos y las presentamos en la tabla de abajo. Ocurre que tenemos una mejor aproximación a E_{out} usando el modelo *PLA-2* que *LinearSVC-2* y podríamos ahora tomar el primero como mejor modelo en vez del que tenemos actualmente. Sin embargo, no podemos hacerlo, ya que el criterio empleado para la elección del mejor modelo ha sido otro cuyo ganador fue *LinearSVC-2*, si hubiéramos escogido como criterio el que mejor cota de E_{out} dé usando el conjunto de test, sí que habríamos escogido *PLA-2* como mejor modelo.

Tabla 11: Cotas de Eout usando la desigualdad de Hoeffding y el conjunto de test

Modelos	**E_out approx.**
PLA-1	0,7208
PLA-2	0,5067
LinearSVC-1	0,6567
LinearSVC-2	0,5114
Regresión Logística	0,6094

2.9. Curvas de aprendizaje

En la figura 6 presentamos la curva de aprendizaje del modelo ganador. La cual consta de dos curvas representadas en un plano 2-dimensional en el que el de abscisas representa el tamaño del conjunto de entrenamiento con el cual se ha entrenado el modelo y en el eje de ordenadas el valor de la métrica alcanzada con ese tamaño de entrenamiento. Tras el entrenamiento se

procede a evaluar el modelo con el conjunto de evaluación y con la parte del conjunto de entrenamiento con el que ha sido entrenado.

Tras la explicación anterior, podemos deducir que el valor de la métrica será mayor al evaluar el conjunto con los datos de entrenamiento que con los datos de validación, ya que el modelo ha sido entrenado con una parte de los primeros. También se puede deducir que al principio, cuando entrenamos con un conjunto reducido de datos, al entrenar y evaluar con el mismo conjunto, estamos sobreajustando por lo que el valor de la métrica en entrenamiento será muy elevada.

Todo lo anterior ocurre en la imagen de abajo. Además observamos que la curva de entrenamiento es estrictamente decreciente, es decir, conforme aumentamos el tamaño del conjunto de entrenamiento el valor de la métrica decrece, esto es debido a que al principio el modelo se ajusta a las particularidades de los datos y poco a poco va ajustándose menos al ruido, por así decirlo. En la curva de entrenamiento ocurre que pese a que se ha entrenado y evaluado en el mismo conjunto de entrenamiento, la métrica no es todo lo alta que uno podría desear, esto en parte se debe al ruido que hay en los datos y también a las limitaciones del propio modelo. Sin embargo, nos conformamos tal y como está ya que lo que hay que buscar no es un aumento de la métrica en la curva del entrenamiento, ya que de ese modo estaríamos sobreajustando el modelo si la separación con respecto a la curva de validación es muy significativa. En este caso, vemos que la distancia no es muy grande, pero tampoco es pequeña. Además, la curva de validación crece muy poco, el hecho de que crezca es positivo, ya que indica que el modelo está realmente aprendiendo de los datos, pero el incremento es algo negativo ya que no es todo lo alto que uno podría desear.

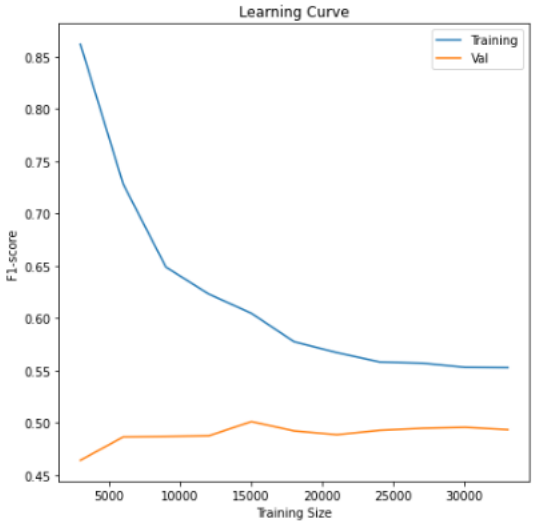

Figura 6: Curva de aprendizaje para el modelo *LinearSVG-2*. Se ha entrenado un total de 10 veces, ampliando cada vez el tamaño del conjunto de entrenamiento al 10 % de su capacidad total.

Con motivos de comparación, también hemos puesto la curva de aprendizaje del modelo *LinearSVC-1* (Fig. 7) que sabemos que usa las características de entrada originales frente a las transformaciones polinómicas de orden dos del modelo *LinearSVC-2*.

Lo primero y principal que debemos fijarnos y que puede pasar de manera inadvertida es el rango de valores que toma el eje de ordenadas, que es mucho menor que la gráfica del modelo *linearSVC-1*, lo que se traduce en peores resultados, como ya sabíamos. Otro aspecto, mucho más notorio, es que las curvas ya no son estrictamente monótonas, sino que hay unos trozos en las que son crecientes y otros en las que son decrecientes. Esto no se traduce en algo malo, de hecho es bastante frecuente, lo que habría que fijarse ahora es en la tendencia y vemos que la

tendencia de la curva de entrenamiento es decreciente y en la de validación creciente, y esto sí es positivo.

Hasta aquí no hay nada nuevo, lo interesante y el motivo por el que se ha puesto esta gráfica es que la diferencia existente entre la curva de entrenamiento y validación al entrenar con todos los datos es muy pequeña, es decir, el modelo no sobreajusta para nada. Aquí mostramos realmente la relevancia del concepto de variabilidad en una clase de funciones. La clase de funciones para este modelo tiene una variabilidad mucho menor que la del otro, lo que se traduce en una menor probabilidad de sobreentrenamiento en detrimento de unos peores resultados. La otra clase de hipótesis, al tener una variabilidad mayor, ocurre que hay funciones que se ajustan mejor a los datos, traduciéndose en resultados elevados pero corremos el riesgo de que se ajusten más a lo particular que a lo general, ocasionando sobreentrenamiento.

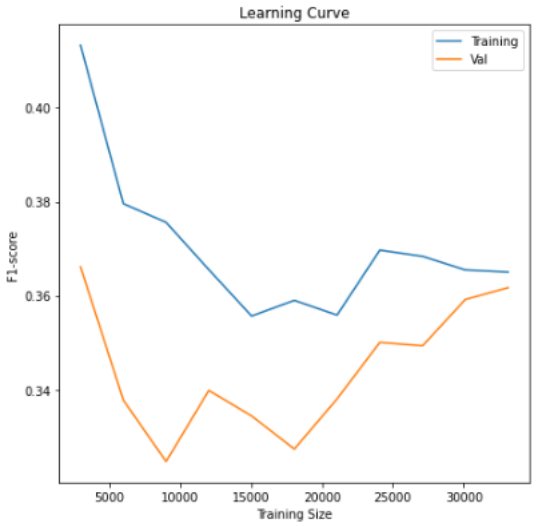

Figura 7: Curva de aprendizaje para el modelo *LinearSVG-1*

A modo de conclusión, podemos decir que hemos encontrado un buen equilibrio sesgo-varianza, en el sentido de que hemos aumentado la variabilidad de la clase de funciones haciendo que disminuya el sesgo, pero a su vez, gracias al efecto de la regularización, hemos conseguido que aplacar el efecto adverso de la variabilidad obteniendo así buenos resultados y poco sobreentrenamiento. De hecho el poco incremente de la curva de validación y la distancia existente entre ambas curvas en la gráfica del modelo *LinearSVC-2* podría dar indicios para no rechazar la hipótesis de un posible sobreentrenamiento, aunque teniendo en cuenta los resultados y los análisis realizados en apartados anteriores, no hay pruebas significativas para confirmarlo.

2.10. Entrenamiento con todos los datos

Tras haber seleccionado la mejor hipótesis procedemos a entrenar el modelo nuevamente con todos los datos obtenidos. Para ello, se van a leer de nuevo los datos almacenándolos en una única estructura que los contenga y se van a preprocesar de la misma manera que venimos haciendo.

Al no tener conjunto de test no vamos a poder estimar el error fuera de la muestra. Entonces, ¿Qué usamos para estimarlo? Se podría pensar el hecho de entrenar y validar con los mismos datos, pero esa estimación no sería representativa y estaríamos cometiendo un craso error. Nuestro objetivo es aprender de los datos y para comprobar si hemos aprendido bien no

podemos usar esos mismos datos. Para que se entienda mejor, imagina que estás estudiando para un examen de geometría y en el examen te caen los mismos ejercicios que hiciste el día anterior para ensayar, como te acuerdas de cómo lo hiciste, le pides un esfuerzo a tu memoria y lo bordas. Pero, ¿podrías decir que realmente has aprendido? La respuesta es que no lo sabes, porque no te has enfrentado a algo nuevo y no has puesto tus conocimiento en juego.

Otra posibilidad es usar un conjunto de validación para estimar E_{out}. Se sabe que el tamaño del conjunto influye en la cota que usemos para estimar el error fuera de la muestra. Un tamaño grande, hace que la diferencia en términos absolutos del E_{cv} y E_{val} sea pequeña. Sin embargo, la mejor alternativa, es usar la técnica conocida como leave one out, la cual consiste en emplear vaidación cruzada con conjunto de validación de tamaño 1 y con tantos folds como ejemplos tengamos ya que por un teorema, el error de validación cruzada es un estimador insesgado de la esperanda del error fuera de la muestra aplicada a N-1 datos. Esta opción es muy costosa en tiempo y en su lugar usaremos una sencilla validación cruzada empleando 5 folds y un 20 % de los datos para validación y emplearemos la métrica F1 en vez del error, como hemos ido haciendo. El resultado final ha sido $F1_{cv} = 0{,}5038$, un resultado prácticamente idéntico al que obtuvimos en la parte de selección de modelos. Por lo que podemos concluir que entrenar con más datos no ha resultado en una mejora significativa de esta métrica.

Tras esto, si se quisiera mandar el modelo a la empresa, se entrenaría con absolutamente todos los datos de los que dispongamos y ya la empresa sería la que nos proporcionaría los datos de test. Tras entrenar el modelo con todos los datos tenemos $F1 = 0{,}53735$. Como podemos ver es un valor más elevado que $F1_{cv}$, pero no podemos decir absolutamente de lo que ocurriría fuera de la muestra ya que no podríamos estimarla en este caso.

2.11. Propuesta del modelo a la empresa.

Ya hemos estudiado al mejor modelo de los que hemos presentado. Pero no podemos llegar a la persona que nos planteó el problema y darle el modelo así por que sí y decirle que entre todos los que hemos probado ha sido el mejor. Vamos a valorar primeramente si el modelo que hemos elegido cumple realmente con el propósito de la campaña de marketing, que es saber si el cliente se subscribirá o no a la poliza. Recordamos que las clases están desbalanceadas y el objetivo es saber qué clientes sí se subscriben, para dejar de invertir en ellos y gastar recursos para convencer a los que no.

Sabemos que las clases están desbalanceadas y que es difícil aprender bien bajo estas circunstancias. Hasta ahora hemos empleado el uso de la métrica F1score para medir cómo de bueno es un modelo, pero en estos momentos los que nos interesa es hacer algo un poco más cercano y fácil de entender para el cliente, que es quien nos va a comprar el modelo. Para ello hacemos uso de la matriz de confusión, la cual nos dice la cantidad de verdaderos y falsos positivos y negativos, la cual la presentamos en la figura 8.

Esta matriz nos dice que de todas las personas que no se subscriben a la póliza, hemos predicho bien prácticamente la mayoría, que son 8494 habiendo fallado en muy poquitos casos 291. En cambio, un gran número de personas que sí se subscriben a la póliza, 683, lo hemos clasificado erróneamente. Y de todas las predicciones de las personas que sí se subscriben a la póliza, hemos acertado algo menos de la mitad.

Estos resultados no son muy favorables y probablemente la empresa nos eche la bronca por no conseguir una mayor tasa de aciertos en la clase positiva. Probablemente un mejor preprocesado u otras propuestas de modelos deberían de ser consideradas para ver si podemos mejorar

los resultados y cumplir las expectativas del cliente. Sin embargo, también le podemos reprochar que los datos que nos han proporcionado no tienen la calidad suficiente como para poder satisfacer sus exigencias, el motivo, el desbalanceo de clases.

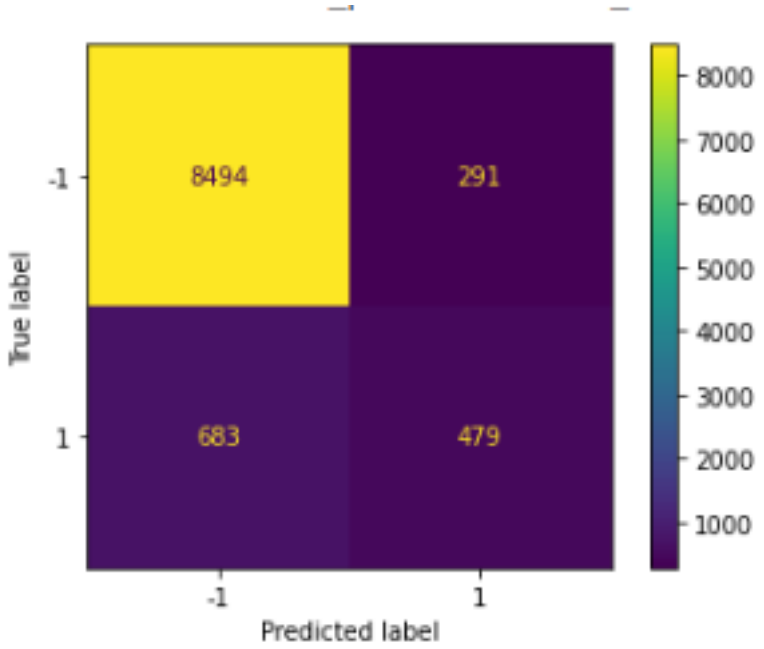

Figura 8: Matriz de confusión. El eje de ordenadas corresponde a las verdaderas etiquetas y el de abscisas a las etiquetas predichas. La clase negativa se representa por -1 y la positiva por 1.

3. Problema de regresión.

Ahora nos vamos a enfrentar a un nuevo problema de Aprendizaje Automático Supervisado pero esta vez es de regresión. Contamos con un único fichero de datas con 515345 ejemplos, 90 atributos incluyendo la salida a predecir que es la primera columna del fichero. Esta vez todos los datos son de tipo numérico y no hay de tipo categórico.

Tenemos que $\mathcal{X}$ es una matriz de valores reales de dimensión 515345×89 donde las filas representan los distintos ejemplos y las columnas las características. El conjunto de etiquetas será un vector de 515345×1 que contiene valores reales que toman las distintas instancias, por lo que $\mathcal{Y} = \mathbf{R}$. La función $f : \mathcal{X} \to \mathcal{Y}$ consiste en asignar un vector de $\mathcal{X}$ un número real.

Este problema cuenta con una gran cantidad de variables por lo que no vamos a poder hacer una visualización exhaustivas de todas ellas como en el problema anterior. En cambio, usaremos otras técnicas nuevas. Primeramente visualizamos algunos atributos del conjunto de entrenamiento usando diagramas de violín (Fig. 9). En ellas podemos ver que las distribuciones de datos están masificadas en una zona concreta y conforme te alejas de la zona se va perdiendo densidad. Esto nos dice que los datos de los atributos están concentrados en determinadas zonas y por lo que observamos, casi todos están en una zona cercana al cero. También se aprecian que están a distinta escala. En la figura 10 vemos el diagrama de violín de la etiqueta. La mediana se sitúa en el año 2000 y su rango intercuantílico corresponde a un intervalo aproximadamente entre 1990 y 2005. Observamos que hay muy poquitos datos situados en por debajo de 1960.

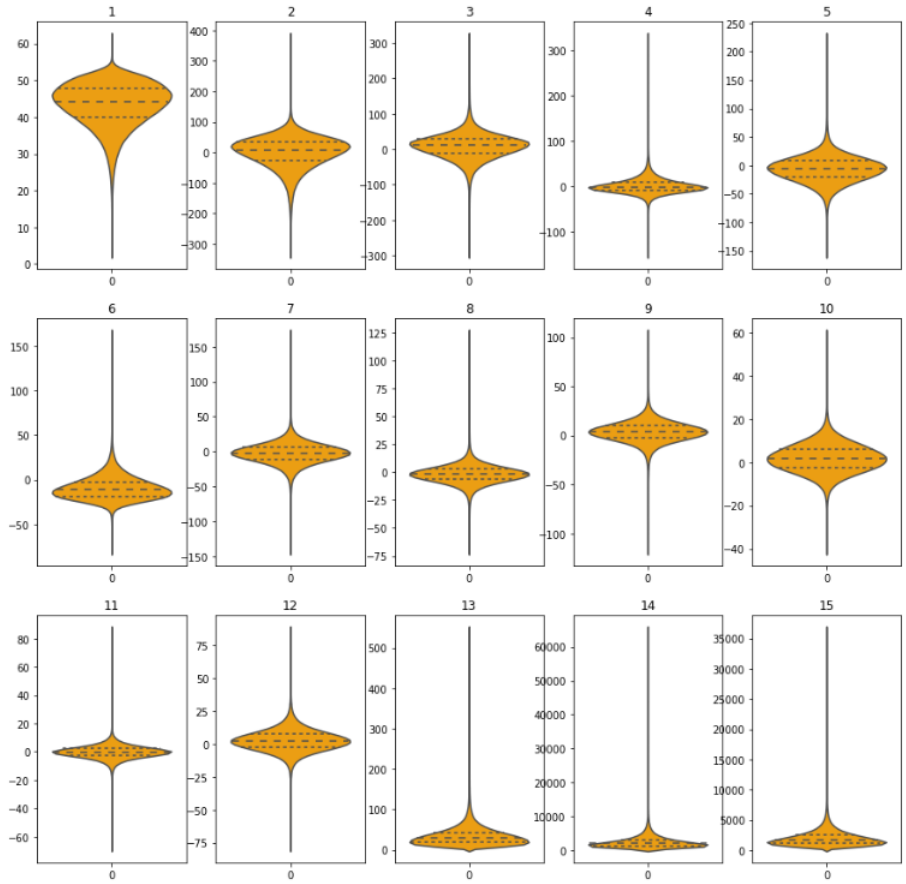

Figura 9: Diagramas de violín para las primeras 15 variables (sin contar la etiqueta). Un diagrama de violín es una mezcla de un boxplot y un diagrama de densidad que muestra la forma de distribución de los datos. Las tres líneas discontinuas representan el rango intercuantílico y la mediana

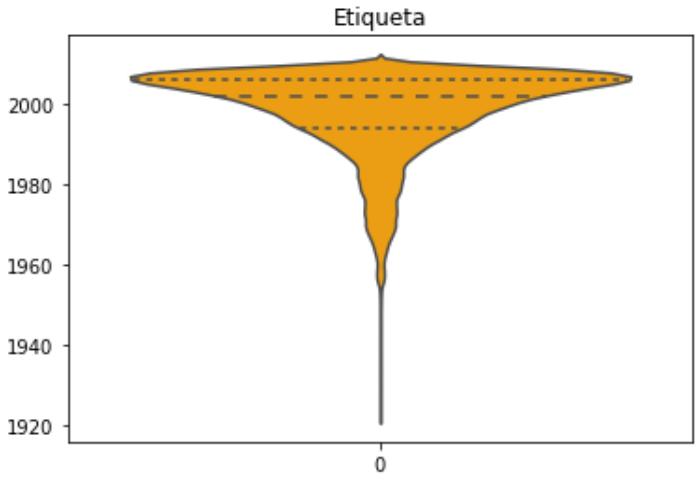

Figura 10: Diagrama de violín para la etiqueta

Como hay muchos atributos sospechamos la posibilidad de que haya correlación entre éstos. Y efectivamente, tras mostrar el mapa de correlaciones (Fig. 11) notamos que la existencia de atributos altamente correlados por la zona superior izquierda del mapa. Imponemos un umbral de $umb = 0{,}8$ y buscamos aquellos atributos que tengan una correlación igual o superior a ese umbral y obtenemos que son los atributos número $16, 18, 23, 20, 23$.

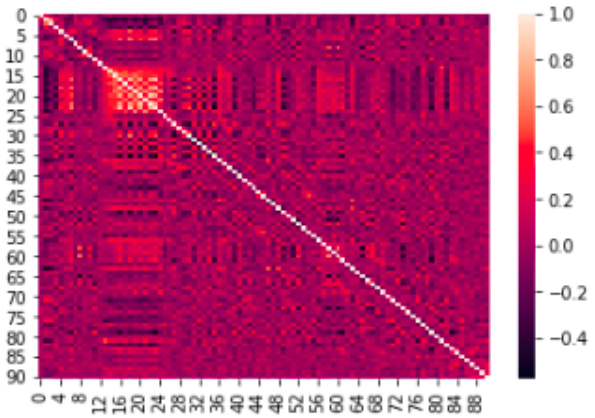

Figura 11: Mapa de calor mostrando las correlaciones de los atributos

Ahora vamos a hacer uso de técnicas un poco más avanzadas para visualización, emplearemos en particular las técnicas PCA y T-SNE. La primera es una técnica de reducción de la dimensionalidad. Hay dos grandes vertientes, las técnicas basadas en proyección y las técnicas basadas en variedades.

El PCA pertenece al primer grupo. EL PCA toma un nuevo sistema de coordenadas ortogonales para el conjunto de datos en el cual la varianza de mayor tamaño del conjunto de datos es capturada en el primer eje, llamada primera componente, la segunda varianza más grande es la segunda componente principal y así sucesivamente. Para el caso que nos trae, vamos a calcular las tres primeras componentes principales. Esta técnica hace uso de los subespacios propios más grandes asociados a la matriz de correlaciones ya que de ahí podemos extraer los valores y vectores propios que apuntan en la dirección de mayor variación del conjunto de datos. Tras aplicar la técnica usando dos componentes principales hemos obtenido una varianza

explicada total del $17,47\,\%$ que corresponde a un $10\,\%$ de la primera componente y a un $7,47\,\%$ de la segunda. Al usar tres componentes principales obtenemos que la tercera explica un $6\,\%$ de la varianza llegando a una varianza total explicada de $23{,}51\,\%$. En ambos casos la varianza obtenida es muy pobre e indica que si nos quedamos con solo esas componentes hemos perdido realmente mucha información. En las figuras de abajo mostramos un diagramas de puntos 2D y 3D que nos muestra la distribución en las componentes principales. Podemos observar que los datos están muy juntos y es difícil distinguirlos. En cambio en la representación 3D hay una zona en la que se distinguen puntos de tonalidades suaves y fuertes.

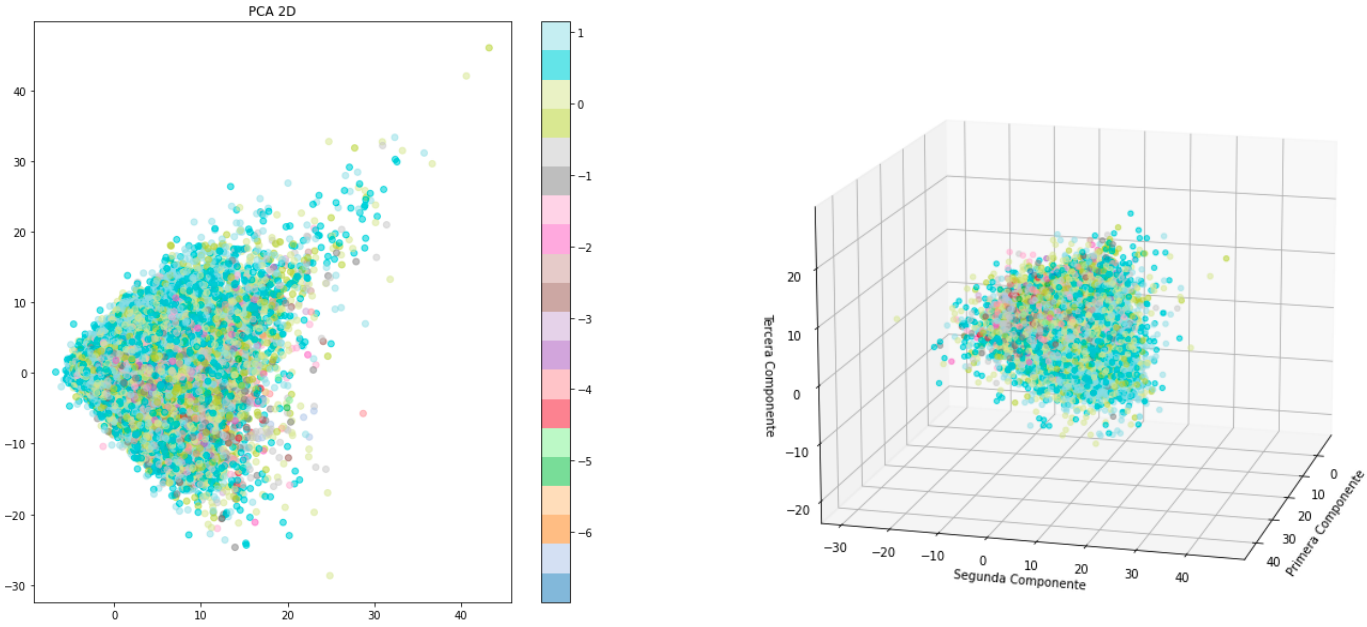

Figura 12: (Izquierda) PCA para dos componentes principales (Derecha) PCA para tres componentes principales

La técnica T-SNE es un método principalmente usado para la visualización de datos. Esta técnica crea una distribución de probabilidad que representa las similitudes entre vecinos en un espacio de gran dimensión y en uno de menor dimensión. Las probabilidades se obtienen a partir de las distancias. Primeramente se calculan las similitudes de puntos en el espacio inicial de grandes dimensión y se calculan para cada punto una lista de probabilidades condicionales observadas con respecto a otros puntos basados en una distribución Gaussiana. Después se crea un espacio de menor dimensión en el que se representen los datos. Se distribuyen los puntos de manera aleatoria por el espacio y se calculan las similitudes de los puntos en el nuevo espacio pero usando una distribución t-Student. Finalmente, para representar fielmente los datos se intenta que las medidas de similitud entre los dos espacios coincidan, para ello se usa la medida de *Kullback_Leibler* y luego se usa el gradiente descendente para minimizar la diferencia entre las distribuciones de probabilidad entre el espacio original y el espacio de menor dimensión. Comentamos que este método es muy costoso en tiempo por lo que solo lo hemos empleado para una visualización en 2 dimensión. Además, se han usado como datos de partida las dos primeras componentes principales obtenidas en el PCA. En la figura 13 observamos que loS datos se han agrupado en una especie de elipse y las etiquetas de un tono más claro (corresponden a las etiquetas que rondan los años 2000) están distribuidas de manera más uniforme. Las de un tono más oscuro están distribuidas mayoritariamente por la zona de la derecha, aunque están repartidas en términos generales también por toda la elipse.

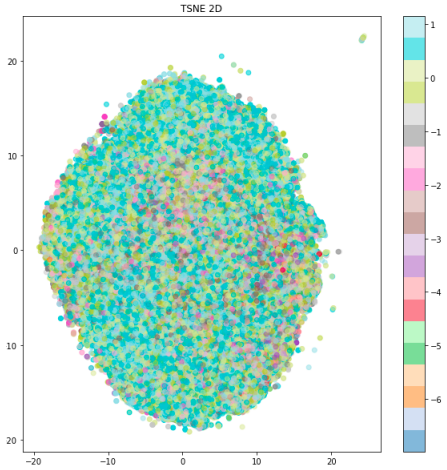

Figura 13: TSNE en 2D usando los datos obtenidos en las dos primeras componentes principales

Más adelante comentaremos que para tratar el conjunto de datos hemos reducido la dimensionalidad a 55 componentes principales y reducir así 35 dimensiones pero previamente haciendo el preprocesdo básico que requiere PCA. Vamos a ahora a representar los diagramas de violín para algunas de las componentes principales. En el preprocesdo se ha normalizado a media cero y desviación típica uno y vemos como esto queda reflejado en los diagrama.as Ahora las distribuciones son todas muy similares entre sí.

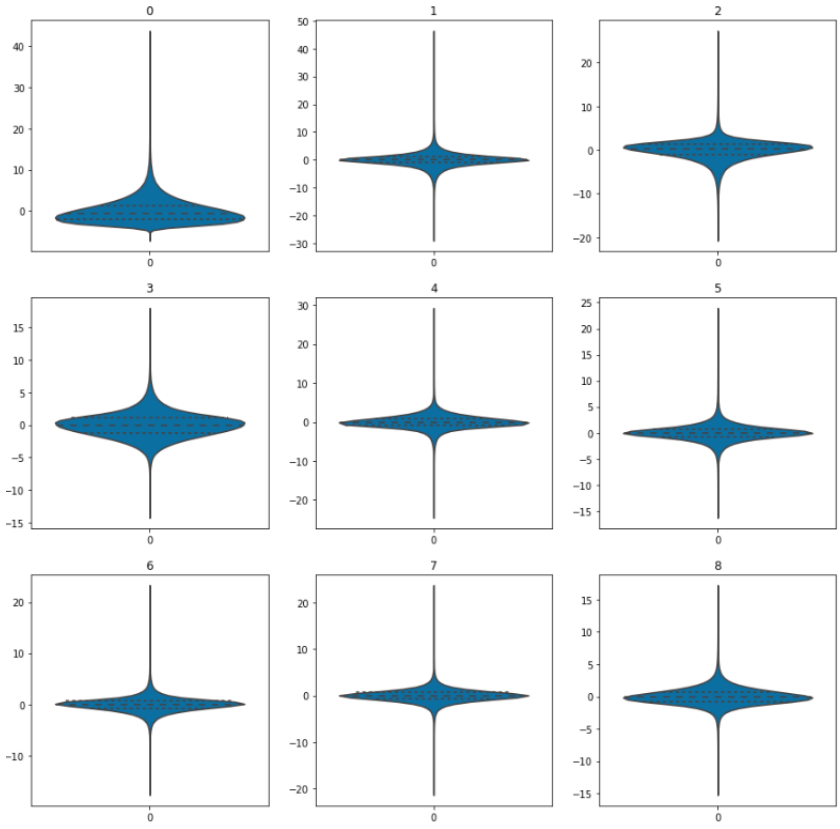

Figura 14: Diagramas de violín para las componentes principales

3.1. Preprocesado

En este apartado no entraremos tanto en detalle como en el apartado anterior ya que los mecanismos para preprocesar ya han sido explicados. Primeramente eliminados los **outliers**

usando *LocalOutliersFactors* el cual hemos usado anteriormente. Los parámetros usados también han sido los mismos que el problema anterior por los motivos que también comentamos. Notamos que antes de aplicar el método normalizamos y tras aplicarlo desnorlizamos para luego volver a normalizar con la media y desviación típica del conjunto de datos sin outliers.

Gracias al mapa de calor estudiado con anterioridad, procedemos a eliminar atributos correlados, teníamos 5 atributos correlados de los cuales el número 16 estaba correlado con el 18 y el 23 y el atributo 20 está correlado con el 22. Por lo que nos quedamos con el 16 y el 20 y eliminamos el resto.

También se ha usado la técnica de *Mutual Information, (MI)* para regresión y hemos visto que tampoco hay característica que sea especialmente importante. De hecho todas tienen poca importancia.

Tras lo anterior, tenemos los datos listos para aplicarle PCA y así reducir la dimensionalidad. Hemos elegido un número de componentes igual a 55 ya que así podemos explicar poco más del 90 % de la varianza y evitar perder información.

3.2. Clase de funciones

En este problema tenemos dos clases de hipótesis. La primera la denotaremos como $\mathcal{H}$ que son funciones lineales con las características originales, es decir, los 87 atributos del problema y $\mathcal{H}_{pca}$ que son también funciones lineales pero con las características obtenidas tras hacer PCA, de este modo, la primera clase de funciones tendrá 87 grados de libertad mientras que el segundo tendrá 55 grados.

En este problema no vamos aumentar la complejidad de la clase de funciones ya que si usáramos las características originales, tendríamos $90^2 = 8100$ dimensiones y el aprendizaje llevaría un considerable gasto de tiempo. Tampoco hemos elegido las dimensiones obtenidas tras aplicar PCA para aplicar transformaciones polinómicas ya que no son las originales del problema y podríamos empeorar los resultados.

3.3. Conjuntos de Training, Test y Validación

Nuestro problema presentaba un único fichero con todos los datos. Del total hemos hecho una partición para quedarnos con un 80 % para entrenamiento y un 20 % para test. Esta regla de $80-20$ se usa bastante cuando la cantidad de datos en lo suficientemente grande, como es el caso.

Posteriormente aplicamos validación cruzada empleando 5 folds, que es otro estándar, por lo que dividimos ahora el conjunto de entrenamiento en otro $80 - 20$ para entrenar y validar respectivamente. Tenemos entonces 103mil instancias para test, 80 mil para validación y 326 mil para entrenamiento. Aunque estos dos conjuntos irán cambiando durante la validación cruzada.

3.4. Métrica de Error

La métrica de error empleada es la del error cuadrático medio (MSE) cuya expresión viene dada por,

$$MSE(w) = \frac{1}{N}\sum_{i=1}^{N}(y_i - w^T x_i)^2$$

donde N es el número de ejemplos del conjunto de datos., w es el vector con los pesos, y_i es la i-ésima etiqueta y x_i el i-ésimo ejemplo.

Hemos usado esta métrica por ser la que generalmente se usa para problemas de regresión, además es continua y diferenciable, por lo que nos viene genial para poder aplicar algoritmos de optimización que se basan en funciones diferenciables como el gradiente descendente o sus variantes.

Además, MSE tiene la ventaja de que viene muy bien para que nuestro modelo no haga predicciones extrañas que originen errores grandes gracias el cuadrado. De este modo, MSE tiene en cuenta los datos de manera individualizada.

3.5. Regularización

No hemos visto la regularización algo necesaria para aplicarla a este problema ya que no nos enfrentamos a una clase de funciones complejas y los modelos empleados tampoco altamente sofisticados. De todas formas hemos empleado regularización porque tampoco nos resulta perjudicial. Las componentes principales del PCA tienen correlación nula entre ellas y los atributos más correlados se han eliminado del conjunto de variables originales. Además, sabemos que las características de entrada aportan poca información, por lo que emplearemos la regularización de tipo L1 (Lasso) siempre que sea posible.

3.6. Modelos empleados e hiperparámetros a tunear

Hemos usado como modelos Regresión Lineal empleando la función *SGDRegression* de sklearn y una máquina lineal de soporte vectorial para regresión empleando el método *LinearSVC* de *sklearn*.

3.6.1. SGDRegression

Los parámetros que hemos fijado en este modelo ha sido la regularización, *penalty='l1'*, el número máximo de iteraciones a mil, la tolerancia a 0,0001. Hemos seleccionado también que baraje aleatoriamente los ejemplos usando como semilla 42. También hemos indicado que el learning rate sea adaptativo, esto eso, que dependiendo de la iteración valga más o menos con el objetivo de acelerar la convergencia. Como función de pérdida hemos usado *squared_epsilon_intensitve* que es el cuadrado de la función *epsilon_intensive* que es diferenciable y penaliza los errores grandes. No hemos usado *early_stopping* porque habría que dejar un trozo de atos para hacer validación y no queremos eliminar más datos de entrenamiento. Además el criterio de parada es por la tolerancia y/o número de iteraciones y hemos considerado que no hace falta.

Hay algunos más parámetros como *average* que calcula la media de los pesos del SGD cuando se actualizan el cual hemos puesto a *False* porque no nos interesa. El parámetro *warm_start* lo que hace es reusar la solución de la anterior llamada a *fit*, lo hemos puesto a como estaba por defecto, ya que solo llamamos a fit una vez y nos da igual. Hay otros hiperparámetros más importantes que como *eta0, power_t* que juegan un papel en la regularización y que por motivos de tiempo de cómputo no hemos podido tunearlos, así que dejamos sus valores por defecto ya

que según los experimentos hechos por los autores, los valores que tienen por defecto son los que mejor suelen ir.

Los hiperparámetros que hemos tuneado han sido dos, uno es *alpha*, el cual juega un importante papel en la regularización y *epsilon*, que es un parámetro relativo a la función de pérdida.

3.6.2. SVRLinear

Las máquinas de soporte vectorial se pueden extender también para el caso de regresión manteniendo sus propiedades. La solución se consigue gracias a los puntos de soporte y minimiza la dimensión VC. En este caso hemos usado como kernel el lineal.

Este método nos permite definir un margen de error que permite admitir errores dentro del pasillo que genera este método. Cuanto mayor sea este margen, más errores admitirá la solución, en cambio, si es muy pequeño, se van a penalizar todos los errores.

Como se comentó en el anterior problema, este método tiene dos formulaciones equivalentes pero distintas de cierta manera, el enfoque primal y el dual. El dual se usa cuando el número de variables es mayor al número de ejemplos, por lo que obviaremos este enfoque y nos centraremos en el primal.

Como función de error usamos *squared_epsilon_insensitive* que viene por la expresión $e_n(w, b) = max(0, |y_n - (w^T x_n + b)| - \epsilon)^2$, siendo (x_n, y_n) un punto. Con esta función de pérdida, aquellos errores que sean menores al margen de error son ingorados. entoncces, la formulación primal queda como sigue,

$$\min_{b,w} \frac{1}{2} w^T w + C \sum_{n=1}^{N} max(0, |y_n - (w^T x_n + b)| - \epsilon)^2.$$

Este método tiene como parámetros muchos de los que llevamos usando hasta ahora: $tol = 0,001$, $fit\_intercept = True$, $dual = False$, $max\_iter = 10000$, y $seeed = 42$ y la tolerancia, que es el criterio de parada, lo hemos fijado a 0,001. Hemos usado los parámetros *epsilon* y C para el tuneamiento, ya que el primero es bastante importante por ser el margen de error que permitimos y el segundo por jugar un papel importante en la regularización.

3.7. Análisis y Selección de las mejores hipótesis

Vamos a presentar el proceso de selección de los modelos. Tenemos cuatro propuestas que se dividen en dos grupos, dos prupuestas que se corresponden a SGDRegressor y otras dos a LinearSVR (SVR). Dentro de cada propuesta, la diferencia está en los datos. Por un lado se han mantenido las características originales de los datos y por el otro se han usado unos datos a los que se le han aplicado PCA. Vamos a denotar a estas cuatro propuestas de la forma *SGDR-PCA*, *SGDR-ORIG*,*SVR-PCA* y *SVR-ORIG*. Comentar que por motivos que desconocemos, *sklearn* tiene implementado *neg_mean_squared_error* en vez del *MSE* normal, por lo que ahora el mejor valor es el error máximo, no el mínimo, ya que tiene signo negativo.

Dentro de los dos modelos de SGDRegressor tenemos que para *SGDR-PCA* hay varias casuísticas que empatan y lo hacen con distintos valores de alfa, de hecho, en algunos casos la regularización juega un fuerte papel y en otros prácticamente es inapreciable. Parece que la

combinación de α junto con el valor de ϵ es la causante. En el caso de *SGDR-ORIG* ocurre algo similar. Podemos observar también que los resultados usando PCA son suavemente inferiores, pero tampoco nada significativo. Los resultados se pueden ver en las tablas 13 y 12.

Dentro de los modelos *SVR* tenemos que los que mejores resultados presentan son aquellos a los que no se han aplicado PCA a los datos. A diferencia del anterior caso, ya no hay un empate entre modelos, ahora tenemos para cada caso un claro ganador y en ambos casos tenemos los mismos valores de los parámetros, que son $C = 0,00316$ y $\epsilon = 0$. Ocurre que ha hecho falta muy poca regularización, quizás pueda deberse a que al tener menos variables los datos, no hay tanta complejidad como para incluirla.

En la tabla 16 presentamos los resultados de los mejores modelos y como podemos ver ha ganado *SVR-ORIG* aunque tampoco tiene mucha diferencia con el segundo puesto. Destacamos que los resultados ganadores en los dos tipos de modelos han sido aquellos a los que no le hemos aplicado PCA. No es de extrañar ya que PCA reduce la dimensión en detrimento de una pequeña pérdida de información traducida en menos varianza para los datos.

Tabla 12: GridSearch SGDRegressor (PCA)

param_alpha	**param_epsilon**	**mean_test_score**	**rank_test_score**
0	0,10	-0,86099	2
0	0,01	-0,86031	1
0,0	0,10	-0,86187	4
0,0	0,01	-0,86121	3
1	0,10	-1,00084	8
1	0,01	-1,00001	5
316,2278	0,10	-1,00084	8
316,2278	0,01	-1,00001	5
100000	0,10	-1,00084	8
100000	0,01	-1,00001	5

Tabla 13: GridSearch SGDRegressor (no PCA)

param_alpha	**param_epsilon**	**mean_test_score**	**rank_test_score**
0,00001	0,1	-0,777	2
0,00001	0,01	-0,771	1
0,00316	0,1	-0,787	3
0,00316	0,01	-0,788	4
1,00000	0,1	-1,001	8
1,00000	0,01	-1	5
316,22777	0,1	-1,001	8
316,22777	0,01	-1	5
100000,00000	0,1	-1,001	8
100000,00000	0,01	-1	5

Tabla 14: GridSearch SVR (PCA)

param_C	param_epsilon	mean_test_score	rank_test_score
0,00001	0,0	-0,8617	13
0,00001	0,01	-0,8617	14
0,00001	0,10	-0,8625	15
0,00316	0,0	-0,8603	1
0,00316	0,01	-0,8603	5
0,00316	0,10	-0,8610	9
1,00000	0,0	-0,8603	2
1,00000	0,01	-0,8603	6
1,00000	0,10	-0,8610	10
316,22777	0,0	-0,8603	3
316,22777	0,01	-0,8603	7
316,22777	0,10	-0,8610	11
100000	0,0	-0,8603	4
100000	0,01	-0,8603	8
100000	0,10	-0,8610	12

Tabla 15: GridSearch SVR (no PCA)

param_C	param_epsilon	mean_test_score	rank_test_score
0,00001	0	-0,7793	13
0,00001	0,01	-0,7795	14
0,00001	0,1	-0,7828	15
0,00316	0	-0,7632	1
0,00316	0,01	-0,7632	8
0,00316	0,1	-0,7639	12
1,00000	0	-0,7632	2
1,00000	0,01	-0,7632	7
1,00000	0,1	-0,7639	11
316,22777	0	-0,7632	3
316,22777	0,01	-0,7632	6
316,22777	0,1	-0,7639	10
100000	0	-0,7632	4
100000	0,01	-0,7632	5
100000	0,1	-0,7639	9

Tabla 16: Resultados de los mejores modelos

Modelos	MSE	Ranking
SVR-PCA	-0,860	3
SVR-ORIG	-0,763	4
SGDR-PCA	-0,860	1
SGDR-ORIG	-0,771	2

3.8. Evaluación con el conjunto de test

Ya hemos seleccionado nuestro modelo, *SVR-PCA* y lo que vamos a hacer ahora es entrenar con todos los datos que tenemos para entrenamiento y evaluar nuestro modelo usando el conjunto de test. Tenemos pues que para entrenamiento obtenemos $MSE_{training} \approx 765$ y $MSE_{test} \approx 0{,}763$. Notamos que ahora sí usamos MSE en vez de *neg_MSE*.

Como podemos ver ocurre que la diferencia entre entre el MSE en training y test es muy pequeña, de tan solo del orden de la milésimas. De hecho con el test podemos calcular la cota a E_{out}

$$E_{out}(g) \leq E_{test}(g) + \sqrt{\frac{1}{2N}\log\frac{2}{\delta}} = E_{test}(g) + \sqrt{\frac{1}{2 \times 407806}\log\frac{2}{0{,}05}} \approx 0{,}763 + 0{,}00212 \approx 0{,}76512$$

.
dicha cota es bastante buena ya que es menor cero. Recordemos que el error cuadrático medio es la suma de cuadrados de la diferencia del estimador y lo que se estime. Si en algún momento la diferencia fuera muy grande tendríamos que el MSE se dispararía. En este caso es menor que uno, y lo que indica es que dicha diferencia cuadrada es muy pequeña.

En el anterior problema entrenamos todas las mejores hipótesis con el entrenamiento y la evaluamos con el test y las comparamos entre sí. Sin embargo, el interés de eso es meramente informativo ya que el mejor modelo se elige previo a este proceso, y cualquier modelo que tenga mejor E_{test} que el modelo que hayamos escogido **no** podrá ser considerado como nuevo mejor modelo. Por lo que en este caso no lo haremos.

3.9. Curvas de aprendizaje

Vamos a estudiar la curva de aprendizaje del modelo ganador. La metodología para la creación de esta curva es la misma que la expuesta en el anterior problema, así que no la volveremos a comentar.

Podemos observar en la imagen de abajo la curva y podemos ver que la curva de entrenamiento es menor que la de validación, como es normal, ya que el MSE es mejor cuanto menor. Observamos que las la distancia entre validación y entrenamiento al principio es muy pequeña y poco a poco se agranda aunque en términos de distancia no se separan mucho, por lo que no podemos observar sobreaprendizaje. Ya que de ser así el curva de entrenamiento debería de estar muy distante de la de validación. Además si observamos el rango de valores en el eje de ordenadas, vemos que la diferencia es de 5 centésimas entre el valor más pequeño que se puede tomar y el más grande y la diferencia que hay al final entre las dos curvas en menor que una centésima.

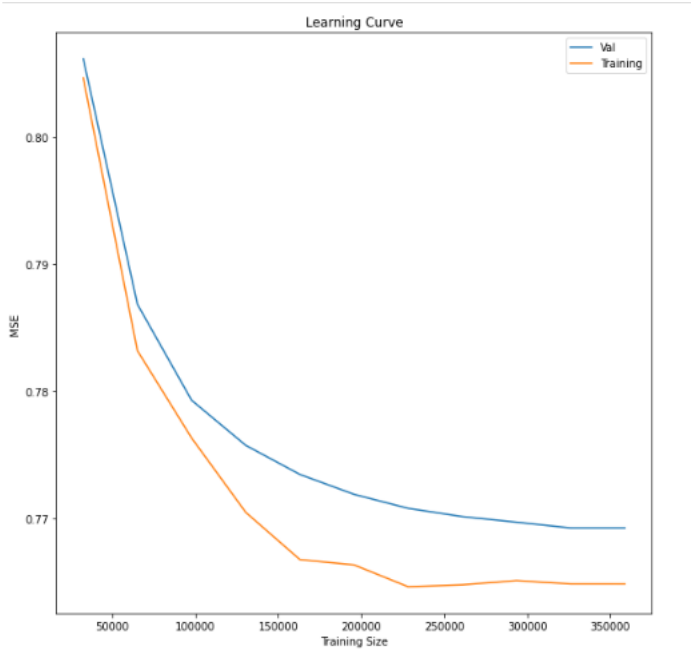

Figura 15: Curva de aprendizaje

3.10. Entrenamiento con todos los datos

Hacemos como en el anterior problema, no usamos conjunto de test, es decir, los datos no se van a particionar ahora. Entrenamos usando cross-validation y tenemos que $E_{cv} \approx -0,7652$. Un resultado similar al que tuvimos a la hora de escoger la mejor hipótesis, por lo que este pequeño aumento de datos en el entrenamiento no ha resultado en un menor error. Notamos que hemos usado *neg_squared_error*

Tras lo anterior, procedemos a entrenar el modelo con absolutamente todos los datos de entrenamiento para presentarlo a la empresa. Tenemos que $E_{in} \approx -0,7645$, un error muy parecido al anterior. El hecho de haber entrenado con más datos no ha supuesto una gran mejora.

3.11. Propuesta del modelo a la empresa

En este caso la empresa no nos ha dado información del problema ni de lo que representan cada atributo. Por lo que respecta a la comprensión del problema, hemos ido un poco a ciegas. En cambio, los resultados de la regresión son bastante buenos.

Hemos usado un error que represnta el cuadrado de la diferencia de lo que estimamos y el valor de la etiqueta y hemos obtenido un valor bastante bajo, por lo que nuestro modelo explica bastante bien los datos. Así que los resultados son favorables y hay mucha probabilidad de que le gusten a la empresa.

4. Anexo

En el guión de la prácticas hay una pregunta ambigua y que vamos a intentar de aclarar. En ella se nos dice que supongamos que tenemos que realizar el ajuste para una emprea (lo hemos supuesto desde el principio) y no nos dan distinción de training y test. Tenemos que decir cuál

sería el mejor modelo que les propondríamos y qué error E_{out} les diría que tiene.

Lo primero de todo, el problema de regresión no tenía distinción de training y test pero el primero sí. sin embargo, en un apartado de la práctica se nos exige que lo juntemos y hagamos nosotros nuestra propia separación. Entonces la respuesta a la pregunta ya está dada. ¿Si no nos dan distinción entre training y test que hacemos? Pues lo que hemos hecho, separar y ya está. Pero ojo, hay que separar con vista, no podemos perjudicar el entrenamiento disminuyendo mucho la cantidad de ejemplos para ese fin.

La otra pregunta es que digamos cúal es el mejor modelo y qué E_{out} le daríamos a la empresa. La respuesta es justamente lo que hemos hecho en esta práctica. Así que no hay nada nuevo que explicar. Primeramente se escoge un criterio de selección, ERM, SRM, etc. Y tras ello se tunean los hiperparámetros del modelo y seleccionamos aquel según el criterio que hayamos tomado. Para dar una cota de E_{out} usamos el conjunto de test