Aprendizaje Automático - Grupo A1 (2021-2022)
Grado en Ingeniería Informática
Universidad de Granada

Práctica 3: Ajuste de Modelos Lineales I (Clasificación)

Fernando Luque de la Torre
20953877A
fl1562001@correo.ugr.es

5 de junio de 2022

Índice

1. Análisis y descripción del problema

El primer paso en la definición de un problema de aprendizaje automático es saber cuáles son cada una de las componentes del mismo. Como ya sabemos, cualquier problema de aprendizaje automático consta con 3 elementos básicos:

- Dominio: $\mathcal{X}$
- Rango: $\mathcal{Y}$
- Función objetivo: f

Antes de continuar, toda la información está extaída del repositorio de la UCI: https://archive.ics.uci.edu/ml/datasets/Bank+Marketing [1]

1.1. ¿Qué es $\mathcal{X}$?

Como en todos los problemas de aprendizaje automático, una de las componentes es el dominio de nuestra función objetivo. En particular, para este problema una entrada de la función objetivo consta de un vector de 16 características referentes a un cliente de un banco. Estas características son de muy diversa naturaleza (números enteros, reales, categorías...), lo que nos obligará a, posteriormente, en la fase de preprocesado de los datos, a realizar ciertas transformaciones. Estas son las 16 características que podemos encontrar concretamente en el fichero bank-full.csv que es el que voy a utilizar:

- Age ($\mathbb{N}$)
- Job (categoría: administrador, amo de casa, retirado, estudiante...
- Marital (categoría: divorciado, casado, soltero o desconocido)
- Education (categoría: universitario, básica 4 años, básica 6 años...)
- Default (binaria: tiene un crédito por defecto (si, no, desconocido))
- Balance (numérica: Balance de la cuenta)
- Housing (binaria: tiene una hipoteca (si, no, desconocido)
- Loan (binaria: tiene un préstamo (si, no, desconocido))
- Contact (categoría: tipo de contacto: móvil o fijo)
- Day (numérica: último día de contacto)
- Month (categoría: último mes de contacto)
- Duration ($\mathbb{N}$: duración de la última llamada)
- Campaign ($\mathbb{N}$: número de veces contactado en la última campaña)

- Pdays ($\mathbb{Z}$: días que han pasado desde la última vez que se le contacto en una campaña previa)
- Previous ($\mathbb{N}$: veces contactado antes de esta campaña)
- Poutcome (categórica: resultado de la última campaña (fallida, inexistente, existosa, desconocida))

De esta forma, cada entrada de la función objetivo será un vector de d variables reales (que provendrán del preprocesado de estas características comentadas) y que darán una salida, la cual voy a analizar en el siguiente punto.

1.2. ¿Qué es $\mathcal{Y}$?

Dado que nos encontramos en un problema de **clasificación binaria**, el rango de la función será el siguiente:

$$\mathcal{Y} = \{-1, +1\}$$

Siendo -1 y 1 las dos etiquetas que asociamos a las dos clases de salida. Pero bien, ¿a qué hacen referencia estas clases? Pues si miramos la documentación del problema extraemos que: *The classification goal is to predict if the client will subscribe a term deposit*, es decir, dada una entrada (datos de un cliente), predecir si este se querrá suscribir a un depósito de plazo fijo. Por tanto, la salida de nuestra función objetivo debe asociar una etiqueta (por ejemplo el $+1$) con una respuesta (por ejemplo el sí), y la otra (el -1) con la otra respuesta (el no).

1.3. ¿Qué es f?

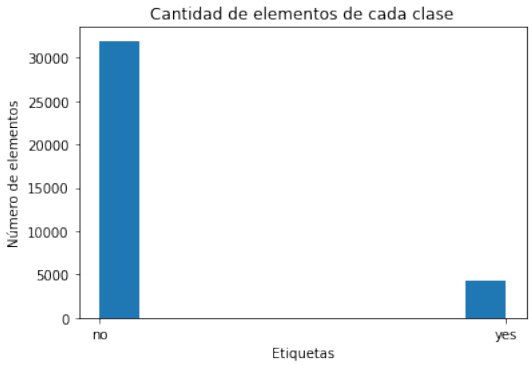
La función objetivo f es aquella función capaz de predecir, dados los datos de un cliente, si este se suscribirá o no al depósito. Como siempre, esta función es totalmente desconocida y nuestro objetivo es encontrar una función g que se asemeje lo máximo posible a esta función objetivo.

1.4. Análisis de datos

Una vez comprendidas las partes del problema, procedemos a realizar algunos pequeños análisis de las diferentes variables del problema con el objetivo de comprenderlo aún mejor.
Antes de nada, aclarar que todos los análisis están realizados sobre el conjunto de entrenamiento únicamente que supone un total del 80 % de la muestra de 45211 ejemplos que tenemos. El proceso de separación de estos conjuntos se explicará posteriormente en el punto 3.

1.4.1. Ejemplos de cada clase

Mostramos en un histograma la cantidad de ejemplos de cada clase en el conjunto de entrenamiento (como ya he dicho, la división entre elementos de train y test se especificará en siguientes apartados):



Añado también los porcentajes de cada clase respecto al total:

- Porcentaje de ejemplos negativos: $88,30\,\%$
- Porcentaje de ejemplos positivos: $11,70\,\%$

Como se puede ver claramente en el histograma, dentro del conjunto de entrenamiento, la cantidad de elementos con respuesta positiva es mucho inferior a la cantidad de elementos con respuesta negativa. Si bien es cierto que el desbalanceo de clases puede tener un fuerte impacto en el proceso de aprendizaje, con estas proporciones debería ser viable. Si bien es cierto, debemos mantener especial atención al error en la predicción principalmente en la clase menos representada, teniendo en cuenta valores como la sensitivity (proporción de verdaderos positivos reconocidos por nuestro modelo).

1.4.2. Resumen de variables continuas

Con el objetivo de comprender mejor los datos, incluímos algunos valores de algunas de las variables continuas:

Edad: Calculamos la media y la desviación típica en el conjunto de training para este parámetro:

- Media: 40,9 años
- Desviación típica: 10,61 años

Como podemos ver, la media de edad del conjunto de training es algo elevada. Esto es bastante lógico, dado que se tratan de datos bancarios, normalmente los jóvenes simplemente tienen alguna cuenta de ahorros simple y rara vez se interesan (o son capaces de acceder) a depósitos como el del problema. Igualmente, la desviación típica es también

bastante considerable lo que nos hace que la distribución de edades sea en un abanico bastante amplio.

Duración de la última llamada: Igualmente, mostramos la media y la desviación típica:

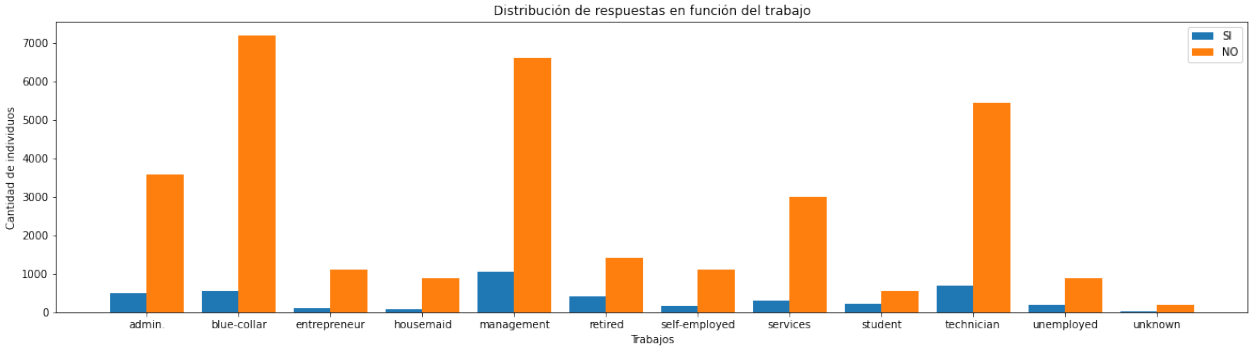
- Media: 257,63 segundos
- Desviación típica: 257,7 segundos

En cuanto a esta característica, aunque la media es relativamente elevada (en torno a los 4 minutos de duración), la grandísima desviación típica nos hace ver que no son tantos los valores que se sitúan alrededor de esta media, sino que también tenemos duraciones muy cortas (así como mucho más largas)

1.4.3. Resumen de variables categóricas

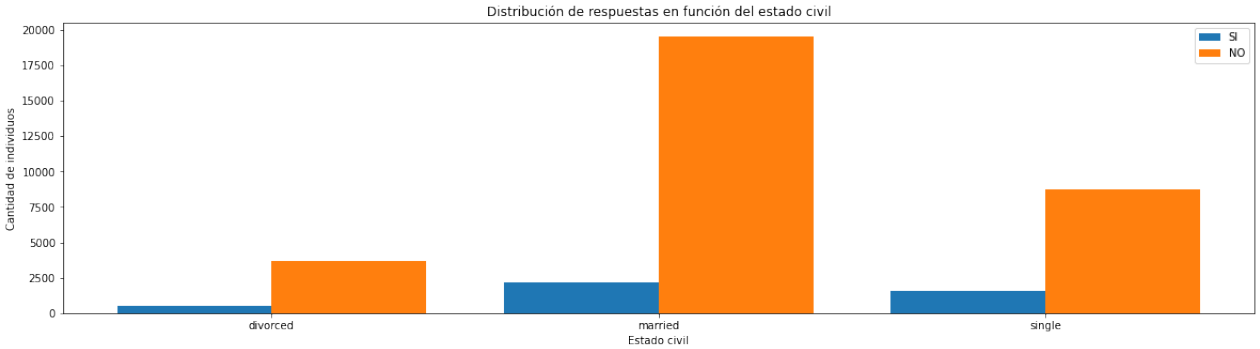
Se muestran algunas de las variables categóricas en histogramas:

Trabajo



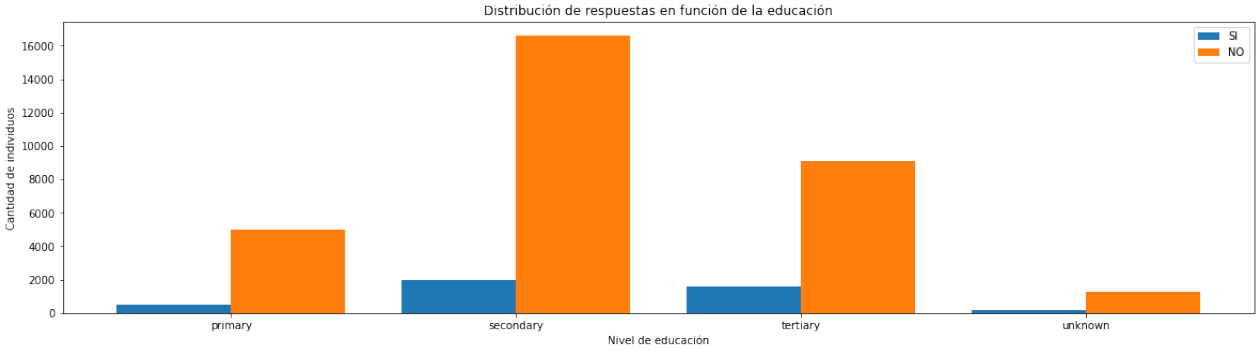
Como podemos ver, hay variabilidad en cuanto a la cantidad de individuos que representan cada uno de los diferentes trabajos. Con esto me refiero, a que hay algunos trabajos (como los blue-collar o los administrativos) que están más representados que otros como los trabajadores en servicios. Además, se puede observar que el trabajo no influye en grandísima medida en la decisión de los individuos ya que la proporción de si y no se mantiene más o menos constante para todos los trbajos. Si bien es cierto, en la categoría de jubilados, la proporción parece ser algo mayor, sin embargo, tampoco es nada determinante. Por último, vemos que la cantidad de individuos de los que no se conoce su trabajo es bastante pequeña en comparación con el resto.

Estados civiles



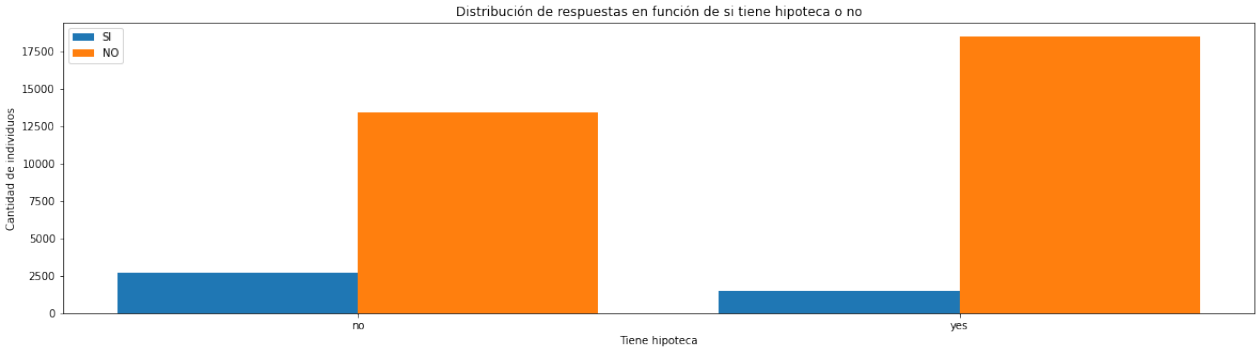
Se puede ver claramente que el estado civil más fuertemente representado es casado/a. De este resultado no se pueden sacar muchas más conclusiones, más que la mayor parte de la población en el rango de edad del que nos habla el análisis de la variable edad, está casada, y no tanta soltera o divorciada (o mucho menos viuda). Sin embargo, sí que se puede observar que la proporción de respuestas positivas es bastante mayor en la clase de solteros, lo cual tiene lógica, ya que las personas solteras tienen mayor independencia económica y no dependen de su pareja para tomar este tipo de decisiones.

Estudios



Podemos observar aquí lo mismo que ocurría con las personas solteras. La proporción de personas con mayores niveles de educación cuya respuesta es positiva es mayor que para niveles de educación inferiores. Esto también es lógico dado que las personas con un mayor nivel de estudios pueden acceder a puestos de trabajo mejor remunerados y por tanto poseer la suficiente liquidez como para contratar estos productos.

Hipoteca



Por último, quizás esta sea la conclusión más clara. Las personas que ya tienen contratada una hipoteca tienen unos gastos fijos todos los meses, lo cual no les permite acceder a estos productos. Sin embargo, las que no tienen hipoteca sí que se lo pueden permitir. Es por eso que la proporción de respuestas positivas es mucho mayor en las personas que no tienen hipoteca que en las que sí.

2. Conjuntos de hipótesis

Dado que la práctica se basa en la aplicación de modelos lineales para un problema de clasificación, los modelos que utilizaremos serán los siguientes:

- **Regresión logística**: que aplica la función sigmoide a la salida de la función con pesos lineales. Este es un modelo útil para problemas de clasificación, que además, nos da una un resultado que se puede interpretar de forma probabilística.
- **Perceptrón**: el algoritmo perceptrón para clasificación. Concretamente, por facilidad, he escogido la implementación que aporta sklearn, ya que, pese a dar resultados no muy buenos en algunas ocasiones, en este caso particular, he obtenido resultados comparables con los de Regresión Logística

En principio, no introduciré ninguna transformación no lineal a los datos. Sin embargo, esta decisión no es definitiva y se analizará la posibilidad de introducirlas posteriormente, si los resultados obtenidos no son aceptables.

El preprocesado de los datos y las técnicas de regularización se comentarán posteriormente en sus respectivos apartados.

Adicionalmente, comentar que he decidido excluir del análisis tanto la Regresión Lineal como otros modelos no lineales como SVM dada la limitación de tiempo para esta práctica y he decidido dejar estos análisis para la práctica final.

3. Definición de conjuntos de training y test

La base de datos del repositorio nos daba diferentes opciones. En mi caso, he escogido el archivo que trae todos los ejemplos juntos y he decidido separarlos en dos conjuntos de training y test. Para ello, he decidido hacer las siguientes particiones:

- Un conjunto de test que no se utilizará en absoluto, más que para estimar el E_{out}, que tendrá un tamaño del 20 % de la muestra total. Reitero que este conjunto no se tendrá en cuenta para ninguno de los cálculos (con el objetivo de evitar el data snooping).
- Un conjunto de entrenamiento (80 %) con el que: se escogerá el modelo, se ajustarán los hiperparámetros y se seleccionará la mejor hipótesis. Además, el proceso de escoger el modelo y de ajuste de los hiperparámetros conllevará una subdivisión de este conjunto de train en 5 particiones, que irán rotando entre 4 de entrenamiento y 1 de validación y que serán utilizadas para la técnica de 5-fold cross-validation.

Esta división será realizada con la función del paquete sklearn.model_selection train_test_split que nos dará la división del porcentaje de ejemplos en test y train que queramos manteniendo la proporción de cada una de las clases en los dos conjuntos.

4. Preprocesado

4.1. Variables categóricas

Como ya hemos analizado, en nuestro problema, contamos con una gran cantidad de variables categóricas, el trabajo, el estado civil, la educación... Por lo tanto, será necesario aplicar transformaciones a estos datos con el objetivo de convertirlas en variables numéricas procesables por nuestros modelos. Para la mayor parte de variables categóricas, una buena transformación es la transformación one-hot. Esta transformación sustituye la variable categórica por tantas columnas como diferentes valores tome esta variable. Es decir, si tenemos una variable que toma 4 valores, se sustituirá la respectiva columna por 4 columnas binarias, en las que, para cada individuo, tomará el valor 1 en aquella que le corresponda por la columna original. De esta forma, aplicaremos esta transformación a las columnas: job, marital, education, contact y poutcome. Así, la sustitución de estas 5 características cada una por una codificación one-hot nos dará el resultado siguiente:

- La columna job toma 12 valores diferentes, por lo que añade 12 columnas
- La columna marital 3 valores
- La columna education 4 valores
- La columna contact 3 valores
- La columna poutcome 4 valores

De esta forma, la codificación de todas estas columnas nos debe dar lugar a 26 columnas binarias de 1s y 0s, que sustituirán a las 5 que previamente contenían las categorías.

4.2. Variables binarias

En este problema, encontramos 3 variables binarias (aunque yo las denomine binarias por su naturaleza, según nos informa la documentación, en este dataset son variables categóricas ya que toman 3 valores (si, no y desconocido)):

- La columna default, que nos informa de si el cliente tiene un crédito por defecto
- La columna housing, que nos dice si tiene una hipoteca
- La columna loan, que nos dice si tiene un préstamo personal

Sin embargo, simplemente mostrando los valores que toman estas columnas:

```
[94] print(np.unique(x_train[:,features["default"]]))
     print(np.unique(x_train[:,features["housing"]]))
     print(np.unique(x_train[:,features["loan"]]))

     ['no' 'yes']
     ['no' 'yes']
     ['no' 'yes']
```

aunque la documentación muestre que puede tomar el valor desconocido, al menos para nuestro conjunto de entrenamiento, no lo toma, por tanto, estas columnas no serán codificadas con one-hot, sino que simplemente serán columnas binarias que tomarán el valor 1 si la categoría es yes y el valor 0 si es no.

Por tanto, estas transformaciones no añadirán nuevas columnas

4.3. Variables cíclicas

Echando un vistazo a las características de nuestro problema, son 2 las variables que pueden hacernos pensar en utilizar con ellas unas transformaciones cíclicas.

La primera característica sería la característica month. En esta característica vamos a utilizar una transformación de tipo seno/coseno, con el objetivo de relacionar los meses de forma cíclica. De esta forma, el mes de enero estará más relacionado con el de diciembre que con el de junio por ejemplo. La transformación a seguir es la siguiente:
Si $i \in \{0, 1, ..., 11\}$ es el índice del mes en el año (lo cual es una transformación previa de categoría a ordinal), se sustituiría este valor por los valores de seno y coseno en el ciclo: $month\_transform(i) = (sin(\frac{2\pi * i}{12}), cos(\frac{2\pi * i}{12}))$. De esta forma, la característica month se sustituirá por dos características, una para el seno y otra para el coseno.

Lo mismo ocurrirá para la característica day, que nos indica el día del mes en que se contactó al cliente. En este caso, la transformación es similar: $Sii \in \{1, 2, ..., 31\}$, se susituirá este valor por los valores de seno y coseno en el ciclo: $day\_transform(i) = (sin(\frac{2\pi * i}{31}), cos(\frac{2\pi * i}{31}))$. Por tanto, la característica day se sustituirá también por otras dos columnas.

4.4. Resultado

Concluyendo con las codificaciones:

- 5 variables categóricas se sustituyen por 26 variables one-hot
- 3 variables binarias se sustituyen por 3 variables binarias
- 2 variables cíclicas se sustituyen por 4 variables numéricas

Por tanto, la nueva matriz X quedaría de tamaño $N \times (16 - 5 + 26 - 2 + 4) = N \times 39$

4.5. Escalado

El último paso del preprocesado de los datos sería el escalado de las variables. El escalado es un proceso que se aplica a las variables con el objetivo de hacer que todas estén en un rango similar de valores, lo cual, se ha demostrado empíricamente que acelera mucho el proceso de aprendizaje.

En este caso, introduciremos el escalado MinMax, que nos situará todas las variables entre los valores 0 y 1 realizando la siguiente transformación:

```
X_std = (X-X.min(axis=0)) / (X.max(axis=0)-X-min(axis=0))
X_scaled = X_std * (max-min) + min
```

Siendo max y min los valores a los que queramos escalar, por defecto y en mi caso: 0, 1.

5. Métrica de error

Dado que estamos realizando un problema de clasificación, la métrica de error utilizada será el porcentaje de error en la predicción (o Accuracy). Esta medida nos dirá cuántas veces (en proporción) fallará nuestro modelo en la predicción tanto dentro de la muestra, como en el conjunto de test como fuera de este (con la cota que establezcamos para E_{out}.

Cabe destacar la diferencia entre la métrica de error y la función de pérdida. La primera nos dice cómo de bueno es el modelo que hemos escogido para el problema que lo hemos aplicado y la segunda la utilizamos para optimizar el modelo (y dependerá del modelo que escojamos).

Como he comentado en un apartado anterior, dado que tenemos una menor proporción de clases positivas en el conjunto de entrenamiento, tendremos también en cuenta la medida de *sensitivity* (o *recall*) o sensibilidad, que nos mide:

$$\frac{Positivos\ que\ acierta\ nuestro\ modelo}{Positivos\ totales}$$

entendiendo por *Positivos totales* aquellos que nuestro modelo acierta (dice que sí, siendo sí la verdadera respuesta) pero también los que dice que no siendo la respuesta en la muestra sí.

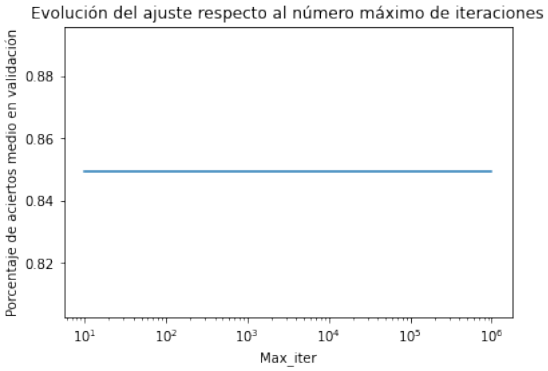
6. Parámetros y regularización

6.1. Perceptrón

En esta sección, escogeremos los valores para los hiperparámetros del algoritmo Perceptrón. Para ello, utilizaremos 5-fold cross_validation sobre los diferentes modelos que surgen de combinar diferentes valores de hiperparámetros. Esto nos dará tantos Errores de Validación medios como modelos propongamos y nos quedaremos con el modelo que mejor resultado de (menor error). Este proceso será realizado automáticamente por la clase GridSearchCV de sklearn la cual se encargará de hacer todas las combinaciones de los parámetros que le digamos y de quedarse con la mejor opción.

Número máximo de épocas

Sin embargo, antes de empezar a realizar los experimentos con cross validation, voy a tratar de reducir al máximo el número máximo de épocas con el objetivo de optimizar los tiempos e ejecución y que los diferentes entrenamientos (que serán bastantes dadas las diferentes combinaciones) tarden lo menos posible. Para ello, ejecuto 5-fold cross validation (con la función de sklearn cross_validation_score, que ya realiza las particiones y nos dice los aciertos en cada iteración) cambiando únicamente el valor del hiperparámetro max_iter y con los valores recomendados por sklearn del resto de hiperparámetros, obteniendo los siguientes resultados:



Aclarar que lo que se muestra en la gráfica se trata del porcentaje de aciertos y no del porcentaje de fallos. De igual modo, podemos observar que el número máximo de épocas no es influyente en el ajuste, por lo que podríamos establecer el más bajo posible. Dado que he obtenido problemas de convergencia en entrenamientos posteriores dando un valor de 10, he decidido establecerlo a 100.

Método de regularización, learning rate y λ

Ahora, una vez establecido un número de épocas máximo razonable, procedemos a hacer el GridSearch con las siguientes alternativas a los parámetros:

- Para el método de regularización escogeremos entre L1 (Lasso) y L2 (Ridge)
- Para el learning rate probaremos 3 valores: {0.1,1,10}. Aunque puedan parecer valores muy altos, la experiencia ha mostrado que son adecuados para este problema
- Para la constante λ (en sklearn alpha), probaremos con $10^{-6}, 10^{-5}$ y 10^{-4}
- Para el criterio de parada tol (para cuando $error > error\_antiguo - tol$), daremos {0.001, 0.01, 0.1}

Por tanto, aplicando 5-fold cross validation sobre los modelos resultantes de combinar los diferentes valores de estos 4 hiperparámetros estaríamos obteniendo $2333 = 54$ errores de validación medios, quedándonos con el modelo que lo minimice. Cabe destacar que dado que estamos haciendo 5-fold cross validation, la obtención de cada error de validación medio supone el entrenamiento del mismo modelo 5 veces con conjuntos de train diferentes, por lo que, en total, la llamada de la clase GridSearchCV a fit estaría entrenando en este caso un total de 270 modelos, lo cual hace que tarde bastante.

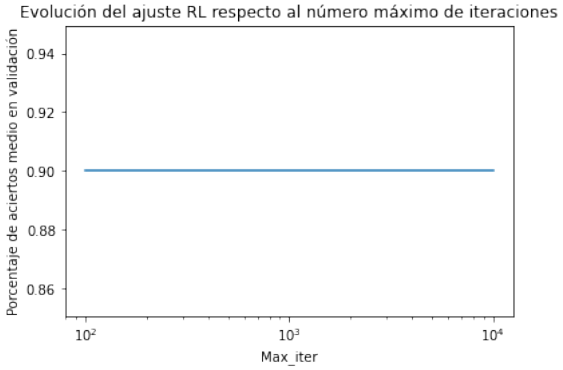
Por tanto, una vez ejecutado este ajuste podemos obtener los valores de los hiperparámetros más óptimos de entre los analizados que serían: {$\lambda = 10^{-6}$, learning rate = 1, regularización = L2, tol = 0.001}

6.2. Regresión logística

En esta sección, haremos para el modelo de regresión logística lo mismo que hemos hecho para el perceptrón.

Número máximo de épocas

Como antes, fijamos un número máximo de épocas razonable:



De nuevo, el número máximo de épocas no influye en el ajuste por lo que lo volvemos a establecer a 100 (se podría establecer a menos pero da avisos de convergencia para algunos valores de hiperparámetros).

Solver, regularización, C, tol, warm_start

De nuevo, vamos a analizar el desempeño en 5-fold cross validation que tienen cada una de las combinaciones de los siguientes hiperparámetros:

- El parámetro solver, que indica qué algoritmo utilizar en el problema de optimización. Las opciones serán liblinear y saga que son los que nos permiten utilizar los dos tipos de regularización L1 y L2
- Tipo de regularización: entre L1 y L2
- Constante C: constante de regularización, al revés que el parámetro λ, a mayor C, menor fuerza de regularización. Tomará los valores {0.001,0.1,1}
- Criterio de parada tol: los valores $\{10^{-5}, 10^{-4}, 10^{-3}\}$
- El parámtero warm_start para que utilice (o no) la solución de la última llamada como inicialización

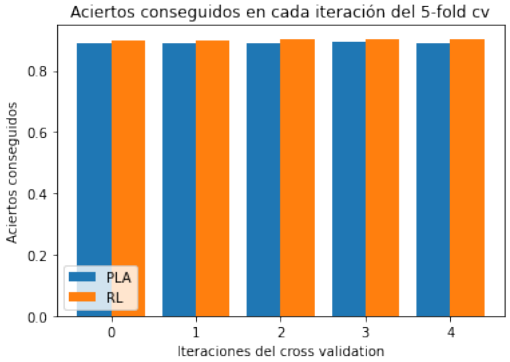
Por tanto, en esta ocasión, estaremos obteniendo $22332 = 72$ errores de validación medios obtenidos cada uno del entrenamiento 5 veces de un mismo modelo lo que supondrá $725 = 360$ entrenamientos.

Una vez realizada la llamada a fit obtenemos que los mejores valores para estos hiperparámetros son: {Solver=saga, Regularización=L1, C=0.1, Tol=0.001, warm_start=True}

7. Selección de hipótesis

7.1. Elección RL vs PLA

Una vez escogidos los hiperparámetros de cada uno de los modelos, vamos a llevar a cabo de nuevo un cross-validation para cada modelo con los hiperparámetros que escojamos con el objetivo de comparar sus mejores resultados entre sí:



Como podemos observar, pese a que las diferencias no son extremadamente notables, el modelo de Regresión Logística nos da un mejor ajuste que el perceptrón por lo que será el modelo que escojamos.

8. Entrenamiento RL

Una vez escogido el modelo, ahora sí, nos encargaremos de aplicarlo sobre el conjunto de training en su totalidad. Es decir, ahora no escogeremos 4 de las 5 particiones para entrenar y la 5ª para validar, sino que utilizaremos todos los datos de los que disponemos para entrenar:

```
print(RL.score(X,y))
0.9009068790090687
```

Como vemos, el error dentro de la muestra es de apenas el 10 %, lo que nos da un ajuste bueno para la muestra escogida.

8.1. Comparación estimador naive

Con el objetivo de comprobar cómo de bueno es nuestro estimador, vamos a compararlo con un estimador naive. En este caso, he decidido escoger uno que, para este caso, parece una buena opción dado el desbalanceo de clases: el estimador que nos dice siempre la respuesta del elemento más frecuente.

Con el objetivo de comparar este estimador con el nuestro, utilizaremos la clase de sklearn.dummy DummyClassifier, la cual puede incluir el parametro strategy='most_frequent' para que clasifique con el elemento más común en la muestra.

Una vez definidos los dos clasificadores, hacemos un 5-fold cross validation sobre cada uno de ellos y comparamos su resultados:

```
Evs = {'RL':cross_val_score(LogisticRegression(max_iter=1e3,penalty='l1',solver='liblinear',C=1e-1),X,y,scoring='accuracy'),
       'Common':cross_val_score(DummyClassifier(strategy='most_frequent'),X,y,scoring='accuracy')}

print("Relación accuracy RL vs Most Common: ", Evs['RL'].mean(), " ," ,Evs['Common'].mean())
```

```
Relación desempeño RL_unbalanced vs Most Common:  0.9006858139251406  , 0.8831563823245692
```

Como podemos observar, pese a que el desempeño en accurarcy de nuestro estimador es algo mejor, tampoco obtenemos una mejora increíblemente notable. Esto se puede deber a que nuestro modelo es demasiado simple para la muestra y quizás sería necesario abordar el problema con otro tipo de modelos no lineales.

Sin embargo, si pasamos a comparar otra medida de error como puede ser el recall (Verdaderos positivos predichos/Verdaderos positivos):

```
Recallsvs = Evs = {'RL':cross_val_score(LogisticRegression(max_iter=1e3,penalty='l1',solver='liblinear',C=1e-1),X,y,scoring='recall'),
       'Common':cross_val_score(DummyClassifier(strategy='most_frequent'),X,y,scoring='recall')}

print("Relación recall RL vs Most Common: ", Recallsvs['RL'].mean(), " ," ,Recallsvs['Common'].mean())
```

```
Relación recall RL vs Most Common:  0.31022451634562925  , 0.0
```

Vemos que la proporción de positivos que consigue predecir nuestro modelo es mucho mayor que la que consigue predecir el estimador naive (que es 0). Sin embargo, podemos observar también que esta proporción de verdaderos positivos es igualmente muy baja para nuestro modelo y este será el aspecto que estaremos tratando en el siguiente apartado.

8.2. Desbalanceo de clases: Problemática

Tal y como vimos en los primeros apartados, la cantidad de ejemplos que encontrábamos clasificados de forma positiva (con respuesta zesz etiqueta $+1$) eran mucho menos que los que encontrábamos clasificados con "no". Es por eso que, a pesar de nuestro modelo consigue un ajuste bastante bueno (aproximadamente un 90 % en el conjunto de test), si echamos un vistazo a otros parámetros de clasificación como el *recall* que ya comentamos que nos decía cuantos verdaderos positivos nuestro modelo acertaba respecto al total de positivos, encontramos lo siguiente:

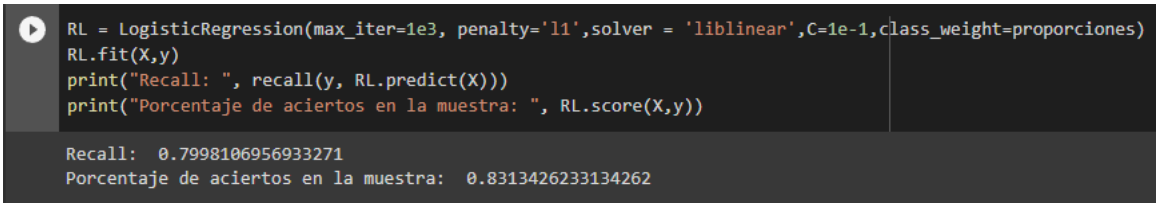
```
RL = LogisticRegression(max_iter=1e3, penalty='l1',solver = 'liblinear',C=1e-1)
RL.fit(X,y)
print("Recall en la muestra: ", recall(y,RL.predict(X)))
```

```
Recall en la muestra:  0.31282536677709416
```

Como podemos observar, este valor de recall es tan solo del 31 %. Pero bien, ¿qué quiere decir esto? Pues el tener un valor tan bajo de recall significa que nuestro modelo, es muy conservador, es decir, que muchos de los ejemplos que se nos presenten (ya fuera de la muestra) y que presenten buenas condiciones para ser respuestas afirmativas de nuestro problema, el modelo escogido va a clasificarlas como respuestas negativas.

Como ya he analizado previamente, la causa principal de este problema es el desbalanceo de clases, pero, ¿cuál sería una posible solución para este problema? Pues bien, los métodos de ajuste de sklearn nos permiten introducir un parámetro class_weight el cual nos permite penalizar especialmente los errores en algunas clases. En nuestro caso, la solución sería penalizar aquellos errores en la clase positiva que sería aquella que tiene menos ejemplos con el objetivo de que nuestro modelo consiga darles más importancia.

Dado que esta es una modificación sencilla, voy a probar a introducir este parámetro en el ajuste para ver cuáles son los resultados obtenidos. Para ello, asignamos a cada clase el peso correspondiente a la proporción de los elementos de la otra clase, es decir, si tenemos 7,56 veces más elementos negativos que positivos, los elementos negativos tendrán un peso de 1 y los positivos de 7,56. De esta forma



Como vemos, aunque la accuracy en general disminuye, ya que clasificamos incorrectamente más elementos en general, se ha reducido enormemente el recall, pasando de un 31 % a un 80 %, es decir, mientras antes predecíamos únicamente un 31 % de los verdaderos positivos, ahora predecimos correctamtente un 80 % lo cual es una mejora enorme a cambio de un pequeño sacrificio en el error en general.

8.3. Acotación E_{out}

Como vimos en la práctica anterior, hay dos modos de acotar el valor de E_{out}. La primera de ellas es con el valor de E_{in}:

$$E_{out} \leq E_{in}(h) + \sqrt{\frac{8}{N} \log \frac{4((2N)^{d_{vc}}+1)}{\delta}}$$

Sin embargo, en esta cota, el término $(2N)^{d_{vc}}$ nos penalizaba mucho, haciendo que esta fuera demasiado laxa y no nos aportara información.

Por otro lado, la otra cota surgía de la idea de utilizar un subconjunto de test que se utilizara únicamente para evaluar la calidad del ajuste fuera de la muestra.

Este subconjunto de test es el subconjunto extraído al inicio de la práctica. Es importante que este subconjunto sea utilizado únicamente para esta estimación de la cota ya que si no, estaríamos haciendo *data snooping*. Además, dado que el conjunto de test se guardó tal y como se extrajo (únicamente convertimos las clases de categóricas a {-1,1}), debemos aplicar a las características **el mismo preprocesado** que aplicamos al conjunto

de entrenamiento, esto es, convertir las variables categóricas, las binarias, las cíclicas y escalar con los valores de máximo y mínimo del entrenamiento.

Una vez aplicados estos mismos preprocesados, procedemos a calcular la cota de la siguiente manera:
$E_{out} \leq E_{test}(h) + \sqrt{\frac{1}{2N} \log \frac{2|H|}{\delta}}$

En esta fórmula, ya que el conjunto de test no se ha usado en ningún momento del entrenamiento, el término $|H| = 1$, ya que solo tenemos una función en la clase, la función obtenida del entrenamiento del modelo. Por tanto con un grado de confianza del 95 %, el error E_{out} será:

```
[21] print("Porcentaje de acierto en test: ", RL.score(X_test,y_test))
     print("Cota E_out: ", (1-RL.score(X_test,y_test))+np.sqrt(1/(2*y_test.size)*np.log(2/0.05)))

     Porcentaje de acierto en test:  0.8239522282428398
     Cota E_out:  0.19032934227083848
```

Como vemos, el haber conseguido extraer un conjunto de test relativamente grande, nos ha permitido reducir en gran cantidad el segundo término de la cota y hacer que esta sea bastante restringida.

9. Curvas de aprendizaje

Las curvas de aprendizaje son una buena herramienta visual que nos permite valorar el comportamiento de nuestro modelo comparando su calidad en el ajuste dentro y fuera de la muestra de entrenamiento.

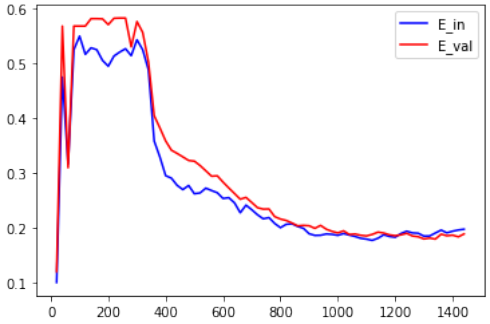
En esta práctica, concretamente la curva de aprendizaje será obtenida de la siguiente manera:

1. Del conjunto de entrenamiento, separaremos un subconjunto de validación que nos servirá para evaluar nuestro modelo entrenado en un subconjunto de los datos

2. Del conjunto de entrenamiento habiendo extraído el conjunto de validación ya comentado, iremos entrenando varias veces nuestro modelo con diferentes tamaños crecientes de subconjunto

3. Para cada modelo entrenado, evaluaremos su error tanto en el subconjunto de entrenamiento para el que se ha entrenado como para el conjunto de validación extraído.

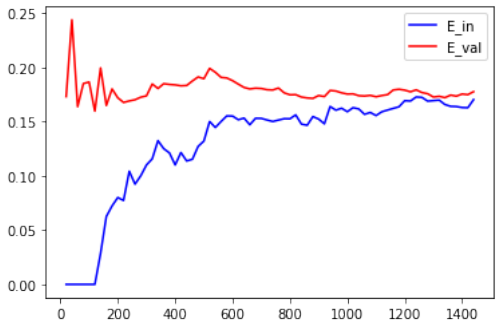
Pues bien, una vez realizado este experimento, el resultado de mostrar las curvas obtenidas es el siguiente:



Como podemos observar, el comportamiento no es quizás el más esperado. Por un lado, como vemos, es lógico que para un número muy pequeño de elementos (primer tamaño que en mi caso era 20), el valor obtenido de error dentro de la muestra era bastante pequeño. Conforme el tamaño crece un poco, nuestro modelo aún no es capaz de obtener buenos resultados ya que necesita de más ejemplos para adaptar bien el hiperplano y vemos que lo consigue para unos tamaños de en torno a 600 muestras.

A pesar de esto, lo que no se ve lógico es que la curva del error de validación (conjunto que no se usa para entrenar) y del error de entrenamiento sean tan parecidas. Esto se debe a que en el experimento, he utilizado los hiperparámetros de regularización obtenidos en la fase de 5-fold cross-validation, los cuales hacen que el modelo (por pocas muestras que tenga) no **sobreajuste** la muestra y generalice (distancia entre el error en la muestra y el error fuera) mucho mejor.

Sin embargo, si ahora repetimos el experimento sin los hiperparámetros de regularización (indicándole que no penalice ni con l1 ni con l2) la cosa es muy diferente:



Como podemos observar, ahora, el comportamiento es más parecido al comportamiento usual de una curva de aprendizaje.

Como podemos ver, el error dentro de la muestra se mantiene en 0 hasta que el número de muestras supera la dimensión de nuestro separador. Mientras tanto, el error fuera de la muestra se mantiene muy alto y con valores un tanto aleatorios.

Sin embargo, conforme va creciendo el tamaño de la muestra, conseguimos que nuestro modelo deje de sobreajustar y consiga generalizar mejor, es decir, aunque crezca el valor de error dentro de la muestra, se reduce la distancia con el error fuera de esta. A pesar de esto, vemos que el modelo que utilizaba los hiperparámetros adecuados de regularización consigue generalizar mejor (siempre mantiene cercana la curva del error fuera a la del error dentro) y por eso sería el modelo con el que nos quedáramos.

10. Caso práctico

Suponga ahora que Ud. debe realizar este ajuste para una empresa que le ha proporcionado los datos, sin distinción entre training y test. ¿Cuál sería el mejor modelo que les propondría, y qué error E_{out} les diría que tiene? Justifique todas las decisiones Pues bien, en el caso de que la empresa nos aportara un único conjunto de datos, se podrían adoptar varios enfoques distintos, en función del tamaño de este:

- En el caso de que hubiera elementos de sobra, un posible enfoque sería el adoptado en esta práctica: en primer lugar separar una parte como conjunto de test y olvidarnos de este hasta el momento en que nuestro modelo esté entrenado y utilizarlo **únicamente** para estimar el valor de E_{out} y no para nada más. El proceso de selección de hiperparámetros lo realizaría con algún tipo de cross-validation y el entrenamiento del modelo con los parámetros escogidos con el conjunto de train en su totalidad.

- En el caso de que tengamos bastantes, pero no suficientes elementos, una buena idea sería omitir la separación del conjunto de train. Es decir, con todo el conjunto, realizaríamos la técnica de cross-validation para elegir hiperparámetros, pero ahora también para elegir los parámetros (pesos) quedándonos con el modelo que menor E_{cv} nos diera. Además, con este E_{cv} concreto, podríamos establecer también una cota para E_{out} que quizás es más laxa que la vista en el punto anterior, pero al fin y al cabo, nos permite trabajar con una menor cantidad de datos.

- Por último, en problemas en los que dispongamos de muy poca cantidad de datos, la técnica que utilizaría sería la de *leave-one-out*, es decir, si N es el tamaño de la muestra que nos aportan, entrenar N modelos dejando como validación un único elemento fuera del conjunto de entrenamiento. De nuevo, nos quedaríamos con el mejor de los modelos tanto en hiperparámetros como en parámetros y podríamos establecer una cota del error fuera de la muestra con el error obtenido en el elemento que hemos dejado fuera.

Como ya he dicho, en función del tamaño del conjunto de los datos, propondría del primer punto al último en orden de preferencia, teniendo en cuenta cuál es el que nos permite utilizar ese tamaño en concreto.

Por último, aclarar que sería una alternativa muy mala validar y entrenar con el mismo conjunto de datos. ¿Por qué? Pues bien, como nos decía la desigualdad de Hoeffding:

$P[|E_{in} - E_{out}| > \epsilon] \leq 2Me^{-2\epsilon^2 N}$

Donde M es el tamaño de la clase de funciones de la que elegimos nuestra función. Entonces, al no hacer distinción entre conjunto de training y test, el tamaño de la clase de funciones en esta expresión sería infinito (realmente sería finito dado que los ordenadores no pueden representar todos los reales, pero igualmente, muy grande) y por tanto, no se podría establecer una cota lo suficientemente pequeña como para asegurar un buen rendimiento de nuestro modelo fuera de la muestra.

11. Bibliografía

Referencias

[1] Sérgio Moro, Paulo Cortez y Paulo Rita. "A data-driven approach to predict the success of bank telemarketing". En: *Decision Support Systems* 62 (2014), págs. 22-31.