UNIVERSIDAD DE GRANADA

Aprendizaje Automático

CURSO 2018/2019

Trabajo 3: Programación

Ajuste de Modelos Lineales

Laura Rabadán Ortega

Contenido

1. COMPRENDER EL PROBLEMA A RESOLVER

En esta práctica consiste en conseguir el mejor predictor posible para unos datos dados a partir de un ajuste lineal. Para este estudio, se proporcionan dos bases de datos de dos problemas distintos, uno de clasificación y otro de regresión, ambos, además, para ser estudiados por métodos de aprendizaje supervisado, ya que los dos cuentan con todas las instancias etiquetadas. A partir de este momento, se comentará por separado el estudio de cada conjunto de datos.

Clasificación

Como base de datos para el problema de clasificación se proporciona el conjunto de datos para el reconocimiento óptico de conjuntos de dígitos manuscritos, *"Optical Recognition of Handwritten Digits Data Set"* (Alpaydin & Kaynak, 1998). Este conjunto cuenta con 5620 ejemplo, cada uno con 64 atributos. Todos los valores de las características son números enteros y es un conjunto sin ruido.

Cada ejemplo se corresponde con un mapa de bits normalizado de un dígito manuscrito. Cada mapa bits es de tamaño 32x32 y se divide en bloques no solapables de 4x4 y se contabiliza el número de bits dentro del bloque. Por tanto, la entrada de un ejemplo es una matriz de 8x8 en la que cada elemento es un entero dentro del intervalo [0,16], el cual se corresponde con el número de bits relevante dentro de cada bloque de datos.

Como el problema consiste en el reconocimiento de dígitos, el número de clases es de 10, es decir, a cada entrada le corresponde un entero del intervalo [0, 9].

Regresión

La base de datos del problema de regresión consiste en datos sobre el ruido propio del perfil aerodinámico, *"Airfoil Self-Noise Data Set",* (Brooks, Pope, & Marcolini, Airfoil Self-Noise Data Set, 1989). Este conjunto de datos cuenta con 1503 instancias con 5 atributos cada uno. Los valores de las características son valores enteros y es un conjunto sin ruido.

El ruido propio del perfil aerodinámico se debe a la interacción entre una pala aerodinámica y la turbulencia producida en su propia capa límite y cerca de la estela. Es el ruido total producido cuando un perfil aerodinámico encuentra un flujo de entrada suave y no turbulento. (Brooks, Pope, & Marcolini, Airfoil Self-Noise and Prediction, 1989)

Los datos del conjunto son el resultado de varias pruebas aerodinámicas y acústicas realizadas en un túnel de viento. Las características de cada atributo se corresponden con distintas palas aerodinámicas NACA 0012 de diferentes tamaños probadas a diferentes velocidades en túneles de viento y a distintos ángulos de ataque.

Las entradas del problema son: frecuencia (hercios), ángulo de ataque (grados), longitud de la cuerda (metros), velocidad de flujo libre (metros por segundo) y el espesor del lado de succión (metros). La salida es el nivel escalado de intensidad acústica, en decibelios.

2. PREPROCESADO LOS DATOS: DATOS CATEGÓRICOS, NORMALIZACIÓN, PROYECCIÓN, ETC.

Para realizar este estudio, se utiliza, para los dos problemas, la biblioteca *ScikitLearn*, haciendo uso en cada momento de las funciones y clases que proporciona para facilitar el desarrollo del experimento.

Entes de llevar a cabo del preprocesamiento de los datos, se cargan las bases de datos y se dividen en conjunto de entrenamiento o train y conjunto de test. De esta división se hablará con más detalle en el apartado 4 de esta memoria.

Para este apartado se han utilizado las clases *VarianceThreshold* y *MinMaxScaler*. La clase *VarianceThreshold* se utiliza para seleccionar y eliminar los atributos que presenten una varianza inferior o igual a una cota dada. En este estudio, y para ambos problemas, la cota especificada es 0.0, es decir, se quieren eliminar aquellas características que no presenten variación alguna, ya que se considera que si para todos los ejemplos proporcionados el valor es el mismo, sea cual sea su salida, esa característica no resulta relevante para la resolución del problema presentado.

Por otro lado, la clase *MinMaxScaler* sirve para normalizar los datos dentro de un intervalo determinado teniendo en cuenta el valor máximo y mínimo de cada atributo del conjunto con el que se entrena. En este caso, el intervalo para ambos problemas es el [0,1]. Cabe destacar que las salidas no se normalizan.

Clasificación

Al principio, este problema constaba de 64 atributos por ejemplo. Se le aplica la selección y eliminación de características, utilizando como referencia para ello el conjunto seleccionado de entrenamiento, y se eliminan tanto los atributos del conjunto de entrenamiento como las características del conjunto de test. A la hora de reducir el conjunto de test no se tiene en cuenta si, al igual que en el conjunto de entrenamiento, los atributos seleccionados no sufren variación, ya que se va a trabajar con el conjunto de entrenamiento, pero la presentación de los datos debe ser la misma para todos los ejemplos que se vayan a predecir con el modelo final.

Por la cantidad de atributos que presentaba el problema, se esperaba que no todos fueran relevantes. Comparando las dimensiones del conjunto antes y después de aplicarle la reducción de características, se observa que el número de atributos pasa de 64 a 62, como se ve en la *Ilustración 1*. La reducción es mínima, pero puede colaborar para reducir tiempo de cómputo a la hora de encontrar una función que explique los datos.

Una vez se han seleccionado las características que se van a utilizar para estudiar y predecir los datos, se les aplica una normalización a los datos, dejando los valores de cada atributo dentro del intervalo [0,1]. Es decir, se pasa de tener valores enteros dentro del intervalo [0,16] a tener representaciones reales en el intervalo [0,1]. Como se ve en la *Ilustración 2*, los valores máximos y mínimos de todo el conjunto de datos pasan de ser 0 y 16 a ser 0 y 1.

```
Selección de carterísticas interesantes:
      Nuevas dimensiones de los datos:
       -> 'Digit':
              - Train:       62 características ;    3823 ejemplos
              - Test:        62 características ;    1797 ejemplos
       -> 'Airfoil':
              - Train:       5 características ;     1202 ejemplos
              - Test:        5 características ;     301 ejemplos
```

Ilustración 1 - Reducción de características

Al igual que pasaba en la reducción de características, los valores máximo y mínimo de los atributos del conjunto de test no se tienen en cuenta a la hora de normalizar sus valore, sino los sacados del conjunto de entrenamiento.

```
Normalización de los datos:
      Valores máximos y mínimos del conjunto de train:
       -> 'Digit':
              16.000000 (MAX) ;       0.000000 (min)
       -> 'Airfoil':
              20000.000000 (MAX) ;    0.000000 (min)
      Nuevos máximos y mínimos del conjunto de train:
       -> 'Digit':
              1.000000 (MAX) ;        0.000000 (min)
       -> 'Airfoil':
              1.000000 (MAX) ;        0.000000 (min)
```

Ilustración 2 - Normalización de los datos

En este problema no se considera necesario binarizar los valores de ningún atributo. Cada característica de cada instancia puede tomar valores enteros dentro del intervalo [0,16], un intervalo acotado y finito de valores, por lo que un atributo determinado solo podrá tomar 17 valores, pero los valores que toma esta característica en una instancia frente a otra están relacionados, se corresponden con el número de bits relevantes, por lo que se decide que son variables categóricas, ya que solo pueden tomas determinados valores, que no interesa binarizar.

Regresión

El desarrollo es similar al problema anterior. Primero se le aplica la reducción de atributos y a continuación, la normalización de valores solo a la entrada.

Este problema contaba con 5 características por entrada, por lo que se puede intuir que todos los atributos de las instancias de este problema van a ser necesarias. Esto se corrobora una vez se aplica la reducción, como se puede ver en la *Ilustración 1*. Todas las características se mantienen, es decir, se pasa de tener 5 atributos a tener 5 características, es decir, no sufre ninguna reducción de atributos.

En este caso, a diferencia del problema anterior, la normalización es necesaria. En el problema de clasificación, como se comentó en el subapartado anterior, todos los valores estaban acotados entre 0 y 16, por lo que, al normalizar, simplemente se está reduciendo el valor del

intervalo. En este caso, cada característica está representada con una medición propia, como hercios, grados o metros, por lo que cada atributo puede tomar diversos valores totalmente distintos a los valores del resto. Además, como se ve en la *Ilustración 2,* el valor máximo de todas las instancias del conjunto de datos es 20 000, un valor muy alto que puede proporcionar resultados elevados con los que será más difícil trabajar. Por esa razón, se decide normalizar los valores de los conjuntos entre [0,1].

Al igual que pasaba en el problema de clasificación, para reducir y normalizar los atributos solo se tiene en cuenta los valores del conjunto de train, pero se aplican en ambos conjuntos a partir de esa referencia.

3. SELECCIÓN DE CLASES DE FUNCIONES A USAR

Para realizar el estudio se han utilizado, para los dos problemas, dos clases distintas de funciones. La clase de funciones lineales y la clase de funciones cuadráticas. Para llevar a cabo la transformación de clases se utiliza la clase *PolynomialFeatures*, la cual, a partir de un conjunto de referencia, transforma el conjunto dado en un conjunto de la clase de funciones polinómicas del grado especificado. Si se quiere que se transforme en un conjunto de la clase de funciones lineales, se llama a la clase con grado 1. El conjunto resultante es el mismo pero con una columna añadida al principio de unos que servirá como constante para la función final.

Si se quiere una función de la clase de funciones cuadráticas, se transforma el conjunto de la manera especificada a continuación llamando a la clase con grado 2.

$$[x_1\ x_2] \Longrightarrow [1\ x_1\ x_2\ x_1^2\ x_1x_2\ x_2^2]$$

Clasificación

Para este conjunto se va a realizar un estudio seleccionando dos clases de funciones para comprobar cual es la que proporciona mejores resultados. En principio, se piensa que, tanto para este problema como para el de regresión, la clase de funciones proporcionará mejores resultados ya que el número de características aumentan y, además, aparecen nuevos valores que son el resultado del producto de dos atributos, los cuales pueden aportar nueva información relevante a la hora de resolver el problema.

Para la clase de funciones lineal, como se dijo al principio de esta sección, la transformación que sufre el conjunto de datos es simplemente una nueva columna de valores constantes, unos, necesaria para conseguir la función final.

En cambio, para la clase de funciones cuadráticas, el conjunto no solo toma la forma especificada al principio de la sección, sino que requiere de un procesamiento de los datos. El número de características pasa de ser 63, clase de funciones lineal, a ser 2016, 32 veces más atributos en la clase de funciones cuadráticas frente a la clase de funciones lineal. Por ese motivo, se estudia si, tras la generación de nuevos atributos, hay características que no interesan ya que no presentan ningún tipo de variación (reducción de atributos). Tras este estudio y transformación se pasa de tener 2016 atributos a tener 1864, consiguiendo prescindir de 152 características. Al hacer esta limpia de atributos, se elimina la columna constante de unos, necesaria para poder generar un

modelo correcto. Por esa razón, se le vuelve a añadir después de llevar a cabo la limpia. El valor de 1864 características incluye esa columna constante.

Regresión

En este problema, el número de atributos es muy pequeño, por lo que se puede pensar que aumentar el número de características ayudaría a conseguir un mejor ajuste, ya que hay más valores con los que trabajar.

En este caso, el conjunto para la clase de funciones lineal cuenta con 6 atributos, y el conjunto para la clase de funciones cuadráticas, después de la reducción de características, tiene 21 características. Como el número de características sigue pareciendo pequeño, frente al número de características del conjunto de datos inicial del problema de clasificación, se prueba a añadirle otra dimensión más a los datos, clase de funciones cúbicas, pasando a tener 126 características. Se compararán las 3 clases para ver cual presenta mejores resultados.

El número de características finales de cada conjunto para cada clase de funciones para los dos problemas se puede comprobar en la *Ilustración 3*.

```
Dimensiones del conjunto de train según la clase de funciones seleccionado:
-> Clase de Funciones Lineales:
        -> 'Digit':
                - Train:       63 características ;    3823 ejemplos
        -> 'Airfoil':
                - Train:       6 características ;     1202 ejemplos
-> Clase de Funciones Cuadráticas:
        -> 'Digit':
                - Train:       1864 características ;  3823 ejemplos
        -> 'Airfoil':
                - Train:       21 características ;    1202 ejemplos
-> Clase de Funciones Cúbicas:
        -> 'Airfoil':
                - Train:       126 características ;   1202 ejemplos
```

Ilustración 3 - Dimensiones de los conjuntos de datos para cada clase de funciones

4. DEFINICIÓN DE LOS CONJUNTOS DE TRAINING, VALIDACIÓN Y TEST USADOS EN SU CASO

Para los dos problemas se utiliza el método de validación cruzada, es decir, se divide el conjunto de entrenamiento en x fracciones determinadas, el número de particiones varía en función del problema y la clase de funciones que se esté aplicando, y se estudia cada conjunto resultado de quitar una de las particiones. Este método se utiliza tanto para conseguir la función resultado como para probarla.

Para ello, se utiliza la clase de funciones *kFold,* indicándole le número de particiones que se quiere realizar y la semilla que se debe utilizar para mezclar los datos de manera aleatoria.

Clasificación

En este problema no fue necesario la partición del conjunto de datos en dos conjuntos distintos, train y test, ya que se proporcionaban en distintos ficheros. Como se puede comprobar en la *Ilustración 4*, el conjunto de entrenamiento cuenta con 3823 ejemplos y el conjunto de test con 1797 instancias, es decir, aproximadamente, el 68% del total de los datos de ejemplos forman parte del conjunto de entrenamiento, y el 32% de las instancias al conjunto de test.

```
Se leyó el fichero de datos 'optdigits'
        -> Train:      64 características ;    3823 ejemplos
        -> Test:       64 características ;    1797 ejemplos
```

Ilustración 4 - Número de instancias de cada conjunto de datos

A la hora de buscar la función objetivo, se utiliza la técnica de validación cruzada partiendo el conjunto de entrenamiento en 5 subconjuntos distintos, independientemente de la clase de funciones utilizada y el modelo empleado. En cambio, para validar la función objetivo, se realizan 10 particiones al conjunto train, y se comprueba el error producido al estimar cada uno de los subconjuntos.

Regresión

A diferencia del problema de clasificación, en este caso se cuenta con un único fichero que recoge todos los ejemplos de los que se dispone, por tanto, hay que dividir el conjunto en train y test. Esta partición se lleva a cabo con ayuda de una función, *train_test_split*. Esta función recibe como parámetros de entrada el conjunto de datos con sus características, el porcentaje de los datos que se quiere que contenga el conjunto de entrenamiento y el porcentaje de los datos que se quiere que formen parte del conjunto de test y la semilla que se va a utilizar para mezclar los datos antes de dividirlos. Tras este proceso, se consigue un conjunto de datos de entrenamiento con 1202 ejemplos y un conjunto de datos de test con 301 instancias, es decir, aproximadamente el 80% de los datos forman parte del conjunto de train y el 20% del conjunto de test.

```
Se leyó el fichero de datos 'airfoil_self_noise'
        -> Train:      5 características ;     1202 ejemplos
        -> Test:       5 características ;     301 ejemplos
```

Ilustración 5 - Número de instancias de cada conjunto de datos

Para la validación del ajuste tanto a la hora de conseguirlo como una vez se tiene seleccionada una función objetivo, se utiliza el mismo método que en el problema de clasificación. En este caso, para conseguir la función objetivo y para evaluarla, se divide el conjunto en 10 particiones distintas.

5. DISCUTIR LA NECESIDAD DE REGULARIZACIÓN Y EN SU CASO LA FUNCIÓN USADA PARA ELLO

La regulación es un método para producir modelos más simples con mejor generalización y solo se ha utilizado en el problema de clasificación.

Clasificación

Cómo se dijo en la primera parte de esta memoria, los atributos de entrada de este problema es el número de bits relevante en cada subcuadrante de 4x4, por tanto, el número de bits del cuadrante x guardará correlación con los cuadrantes adyacentes a él. Por este hecho, se decide aplicarle una Regularización Ridge para los conjuntos de datos de ambas clases de funciones.

Regresión

Para este problema no se aplicó ningún tipo de regulación, ya que, como el número de atributos de entrada era bajo, se consideró que no era necesario reducir la complejidad del modelo.

6. DEFINIR LOS MODELOS A USAR Y ESTIMAR SUS PARÁMETROS E HYPERPARÁMETROS

Los modelos disponibles eran: Regresión Lineal, Regresión Logística y Perceptrón. Por tanto, para el problema de clasificación se probaron dos modelos distintos, Regresión Logística y Perceptrón, en cambio, para el problema de regresión, solo se utilizó el modelo de Regresión Lineal.

Para poder probar distintas combinaciones de opciones de entrada, se utiliza una clase de la biblioteca, *SearchGridCV*. Además de probar las distintas combinaciones de parámetros al modelo especificado, proporciona la funcionalidad de validación cruzada, parándole por parámetro el número de particiones con las que se quiere trabajar.

Clasificación

Para este problema se probaron los modelos de Regresión Logística y de Perceptrón, cada modelo funcionando tanto para la clase de funciones lineal como para la clase de funciones cuadrática.

Para el modelo de Regresión Logística se utiliza la clase *LogisticRegression*, la cual recibe como parámetros de entrada que se quiere usar una regularización de tipo L2 (Ridge, *penalty)*, una tolerancia de 0.05 (*tol*), un máximo de 500 iteraciones (*max_iter)*, *saga* como algoritmo de aprendizaje (gradiente de rápida convergencia, *solver*), la fuerza de regulación que se quiere (*C*) y la semilla para la generación de números aleatorios (*random_state*). Se le especifica que todas clases tienen la misma importancia (*class_weight*), es decir, que ninguna es más o menos importante que otra y que no es un problema binario, sino que hay varias clases, opciones de salida (*multi_class*).

Por otro lado, para el modelo de Perceptrón se utiliza la clase *Perceptron*, se le pasa como argumento que se quiere que se estudie el conjunto con regulación de tipo L2 y sin regularizar (*penalty*) y la fuerza que se quiere que se aplique a la regularización (*eta0*). Además, se le especifica que el número máximo de iteraciones permitidos son 1000 (*max_iter*), la tolerancia es de 0.05 (*tol*) y que se quiere que se mezclen los datos después de cada época (*shuffle*), usando el valor de semilla especificado (*random_state*). El algoritmo parará antes si, tras 1 o 5 iteraciones, se prueban ambos parámetros, el conjunto de validación no cambia (*n_iter_no_change*). Como conjunto de validación, se guarda un 10% de datos del conjunto de entrenamiento (*validation_fraction*). Ninguna clase tiene más o menos importancia que otra (*class_weight*).

Regresión

Para el problema de regresión se tenía como opción un único modelo, dentro de los modelos lineales estudiados. Por esta razón y por el poco número de atributos de las instancias, se decide estudiar una clase de funciones más que en el problema de clasificación.

Para la implementación del modelo para el problema de regresión, Regresión Lineal, se utiliza la clase *LinearRegression.* Como parámetros, se le dice que no se quiere que se normalicen los datos (*normalize*), que no se quiere que se sobreescriban las transformaciones sobre los datos que se tiene (*copy_X*) y el número de núcleos a usar en caso de que se utilice programación paralela (*n_jobs*).

7. SELECCIÓN Y AJUSTE MODELO FINAL

Clasificación

Para el estudio de clasificación, se utilizaron 2 clases distintas, lineal y cuadrática, y dos modelos diferentes, Regresión Logística y Perceptrón. La predicción se evalúa con el porcentaje de aciertos a la hora de estimar la clase, *Accuracy*. El porcentaje del error de cada estimación se puede calcular haciendo $100 - \%aciertos$, lo que daría que el error, en este caso, sería muy pequeño. Los resultados conseguidos en el estudio se muestran en la *Tabla 1*. Se puede observar que la clase de funciones lineales, en ambos algoritmos, tarda mucho menos que la clase de funciones cuadráticas, ya que el número de características con el que trabaja es mucho menor. Ambos modelos, independientemente de la clase de funciones que utilizen, consiguen una cota de aciertos muy alta. Para seleccionar un único modelo, se calcula la cota del error a partir del error del train (más en profundidad en el punto 9). A partir de este resultado, se selecciona el modelo de Perceptrón con clase de funciones cuadráticas. Todos los modelos dan errores aproximados, pero el error más pequeño es el que se consigue con este modelo y con esta clase.

Como se puede comprobar en la *Tabla 1*, se llevó a cabo un estudio para comprobar el funcionamiento de la función seleccionada, partiendo el conjunto de entrenamiento en 10 subconjuntos distintos y calculando el número de aciertos conseguidos. En todos los casos, se consigue un porcentaje de aciertos superior al 95%, por lo que, para el conjunto de train, la función seleccionada consigue un buen ajuste, pero puede que se haya conseguido un sobreajuste de los datos. Por esa razón, se calcula la cota de error fuera de la población y se selecciona el conjunto que presente una cota menor.

	Regresión Logística		Perceptrón	
	CF lineales	**CF cuadráticas**	**CF lineales**	**CF cuadráticas**
Tiempo	1.1386s	49.70s	2.5930s	31.57s
Aciertos Medio	96.39%	97.62%	92.91%	96.62%
Aciertos Train	97.41%	98.93%	96.57%	99.03%
Aciertos conj. 1	98.43%	98.96%	97.91%	99.22%
Aciertos conj. 2	97.12%	99.48%	96.34%	98.96%
Aciertos conj. 3	97.65%	98.96%	96.61%	98.69%
Aciertos conj. 4	97.64%	98.69%	96.07%	98.95%
Aciertos conj. 5	96.86%	98.95%	95.29%	99.48%
Aciertos conj. 6	96.60%	98.17%	96.60%	98.95%
Aciertos conj. 7	97.12%	98.95%	96.07%	98.95%
Aciertos conj. 8	98.17%	98.95%	96.60%	99.74%
Aciertos conj. 9	97.12%	98.69%	96.86%	98.17%
Aciertos conj. 10	97.38%	99.48%	97.38%	99.21%

Tabla 1 - Resultados del problema de clasificación

Una vez seleccionada la clase y el modelo, se comprueba el porcentaje de aciertos que consigue la función seleccionada sobre el conjunto de test. El resultado conseguido es el que se muestra en la *Ilustración 6.* Se puede comprobar que se consigue un error de un 4.17%, un error relativamente bajo.

```
Resultados al usar el modelo en el conjunto de test de los problemas:
-> Regresión:  7.964718% de error en el conjunto de test
-> Clasificación:      95.826377% de acierto en el conjunto de test
```

Ilustración 6 - Predicción del conjunto de test

Regresión

En este caso, se contaba con un solo modelo, Regresión Lineal, por lo que es este el modelo seleccionado. En cambio, se tienen tres clases de funciones distintas, línea, cuadrática y cúbica. En el problema anterior, se calculaba la precisión del ajuste en función al número de aciertos que conseguía la función. En este caso, se calcula por medio del error medio cuadrático, que se calcula como:

$$E_{in} = (h(x) - y)^2 \text{ siendo } h(x) = xw^T$$

Con este calculo del error de la estimación del error y el mismo método que el que se utiliza en el problema de clasificación, se consigue la *Tabla 2* de resultados. En este caso, la clase de funciones que más tarda es la clase de funciones lineales, aproximadamente 3 segundos más que el resto de las clases. Se puede observar que, cuanto mayor es el número de clases, menor error consigue. Al aumentar la dimensión de la clase de funciones, la función objetivo es más complicada, por lo que, en vista de los resultados, podría estarse sobreajustando la función objetivo a mayor dimensionalidad de la clase. Por eso mismo, al igual que se hizo en el problema de clasificación, se calcula la cota del error a partir de los resultados obtenidos.

	Regresión Lineal		
	CF Lineales	**CF Cuadráticas**	**CF Cúbicas**
Tiempo	3.5830s	0.1110s	0.3220s
Error Medio	24.07%	18.11%	10.51%
Media Error Train	23.86%	17.33%	6.79%
Media Error conj. 1	23.87%	16.03%	6.24%
Media Error conj. 2	26.04%	17.31%	7.11%
Media Error conj. 3	20.24%	17.13%	6.97%
Media Error conj. 4	21.01%	14.11%	5.65%
Media Error conj. 5	30.06%	22.05%	6.79%
Media Error conj. 6	30.18%	21.83%	7.96%
Media Error conj. 7	22.46%	15.82%	5.82%
Media Error conj. 8	22.12%	14.41%	6.72%
Media Error conj. 9	21.11%	16.70%	6.69%
Media Error conj. 10	21.38%	17.97%	7.99%

Tabla 2 - Resultados del problema de regresión

Según esta estimación (punto 9 para más detalle) se determina que la mejor cota de error es la que determina el modelo con la clase de funciones cúbicas. Por tanto, se comprueba el error en el conjunto test conseguido con este modelo y esta clase de funciones, y se obtiene una tasa de acierto de 92.04% y un error de 7.96% (*Ilustración 6*). El error conseguido está acorde al error conseguido en los estudios realizados en el conjunto de entrenamiento, por lo que parece ser aceptable frente a los errores que se podrían haber conseguido con las otras clases de funciones.

8. DISCUTIR LA IDONEIDAD DE LA MÉTRICA USADA EN EL AJUSTE

Para estudiar calidad de la función objetivo conseguida en cada experimento de cada problema, se divide el conjunto de entrenamiento en diez subconjuntos y se estudia el porcentaje de acierto o de error, en función del problema que se esté estudiando.

Clasificación

Para este problema se utiliza como métrica el porcentaje de aciertos, *Accuracy*, y una matriz de confusión con la predicción del conjunto de train. Se consiguen cuatro matrices distintas, una para cada conjunto del estudio.

```
Matriz de confusión:
[[372   0   0   0   0   2   1   0   0   1]
 [  0 376   1   2   0   0   0   1   6   3]
 [  0   1 373   2   0   1   0   0   1   2]
 [  0   1   0 382   0   2   0   0   0   4]
 [  0  13   0   0 368   0   0   0   0   6]
 [  0   0   2   2   0 364   0   0   0   8]
 [  0   9   0   0   2   2 362   0   2   0]
 [  0   0   0   2   0   1   0 382   0   2]
 [  1  14   0   4   3   2   0   0 353   3]
 [  1   4   0   6   3   1   0   2   5 360]]
```

Ilustración 7 - Perceptrón, Lineal

```
Matriz de confusión:
[[369   0   0   1   1   3   0   0   2   0]
 [  0 387   0   0   0   0   0   1   1   0]
 [  0   4 373   1   0   0   1   0   1   0]
 [  0   0   0 384   0   1   0   1   1   2]
 [  0   1   0   0 386   0   0   0   0   0]
 [  0   1   0   0   0 373   0   0   0   2]
 [  0   1   0   0   0   0 376   0   0   0]
 [  0   0   0   0   0   0   0 387   0   0]
 [  0   7   0   0   0   1   0   0 372   0]
 [  0   1   0   0   0   1   0   1   0 379]]
```

Ilustración 8 - Perceptrón, Cuadrática

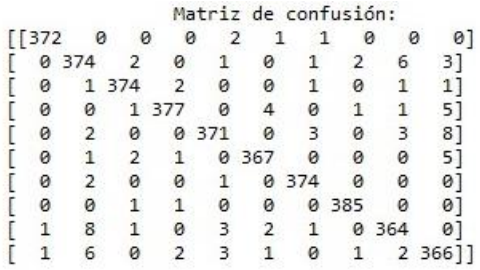

Ilustración 9 - Regresión Logística, Lineal

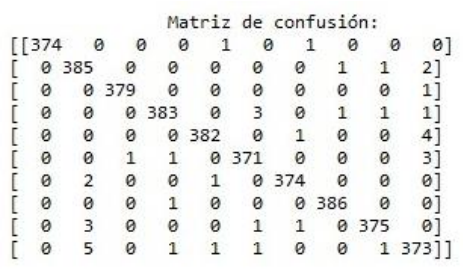

Ilustración 10 - Regresión Logística, Cuadrática

En las ilustraciones de la 7 a la 10 se puede observar la matriz de confusión de cada conjunto de modelo-clase de funciones. En estas matrices se representa a que número de instancias con salida i (filas) se le asocia un valor de salida determinado j (columnas). Un muy buen ajuste tendrá mayor número de asignaciones en la diagonal, lo que representa que le está asignando la salida que le corresponde.

Si se juntan los resultados obtenidos con el *Accuracy*, la tasa de acierto, con los resultados de la matriz de confusión, se observa que, como todos los algoritmos consiguen un porcentaje de aciertos superior en todos los casos al 95%, el mayor número de asignaciones se encuentran en la diagonal principal.

Además, se puede comprobar que en el caso del Perceptrón, tiende a asignar con mayor error la clase correspondiente a la segunda columna, es decir, el mayor porcentaje de error ha sido asignándole esa salida a instancias que tienen otra salida distinta. En cambio, en la Regresión Logística, cuando etiqueta de manera incorrecta, tiende a asignar la última clase.

Independientemente de esos errores cometidos en la asignación, ambas métricas representan de manera ilustrativa la calidad de la función objetivo. El *Accuracy* muestra el porcentaje de aciertos que ha conseguido, útil para cálculos rápidos, y a matriz de confusión muestra de forma visual las asignaciones que realiza la función, por lo que permite conocer las salidas con las que mayor error comete la función, por ejemplo.

Regresión

Para el problema de regresión se utiliza la Media del Error Cuadrático, calculando la diferencia entre la salida asignada por la función objetivo y la salida real que debería proporcionarse. Al hacer la media entre todas las instancias que se tiene, devuelve el porcentaje de error cometido. Este porcentaje puede interpretarse como cuanto se ha desviado la función objetivo de la etiqueta real de los datos que intenta aproximar, de media.

Como es un problema de regresión, este error es bastante informativo, ya que está indicando cuanto se está desviando la función objetivo, de media, de la función objetivo óptima.

9. ESTIMACIÓN DEL ERROR EOUT DEL MODELO LO MÁS AJUSTADA POSIBLE

La cota del error se utiliza para poder seleccionar el modelo y la clase de funciones que se va a seleccionar.

Clasificación

Para calcular la cota del error se quería utilizar la función presentada en la *Ilustración 11*, pero con la clase de funciones cuadrática conseguía valores muy altos y Python no los admitía. Por consecuente, tuvo que cambiarse a la fórmula de la *Ilustración 12*, con la cual se han conseguido las cotas mostradas en la *Ilustración 13*.

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

Ilustración 11 - Cota para el error global con el conjunto de train

$$E_{out}(g) \leq E_{in}(g) + \sqrt{\frac{1}{2N}\ln\frac{2|\mathcal{H}|}{\delta}}$$

Ilustración 12 - Cota para el error global con el conjunto de test

```
Errores globales estimados para ambos problemas con cada modelo y clase utilizado:
-> Regresión:
        - Regresión Lineal
                - Lineal:       Eout =  23.859394%
                - Cuadrática:   Eout =  17.334803%
                - Cúbica:       Eout =  6.793262%
-> Calsificación (Con probabilidad de almenos 0.95 de que...):
        - Regresión Logística
                - Lineal :      Eout <=  0.7324017729334076
                - Cuadrática:   Eout <=  0.3348082599854609
        - Perceptron
                - Lineal :      Eout <=  0.8809761909297454
                - Cuadrática:   Eout <=  0.2848474961873966
```

Ilustración 13 - Cotas de error para ambos problemas

Si se compara los cuatro resultados obtenidos para el problema de clasificación, se puede deducir que la mejor clase de funciones para este problema parece ser la clase de funciones cuadráticas, ya que en ambos modelos son los que consiguen una menor cota de error. Si se tuviera que utilizar la clase de funciones lineal se seleccionaría el modelo de Regresión Logística, el cual tiene una cota 1.2029 veces mejor que la conseguida con el modelo de Perceptrón. En cambio, al seleccionar la clase de funciones cuadráticas, el mejor modelo el del Perceptrón, siendo una cota 1.1754 veces mejor que la conseguida con el modelo de Regresión Logística. De cara a estos resultados, se escogió el conjunto formado por la clase de funciones cuadráticas con el modelo de Perceptrón, y se utilizó para estimar el conjunto de test (Punto 7).

Regresión

Para este problema, se utiliza como función de predicción la presentada en la *Ilustración 14*. Se probó a utilizar la misma función que en el problema de clasificación y se conseguían resultados similares, por lo que se mantuvo esta manera de calcular el error.

$$E_{out}(h) = \mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{P}}[(h(\boldsymbol{x}) - y)^2]$$

Ilustración 14 - Cálculo del error en el problema de regresión

Como se observa en la *Ilustración 13*, se consigue una menor cota de error con la clase de funciones cúbica, la cual es 2.5518 veces mejor que la cota conseguida con la clase cuadrática y 3.5122 veces mejor que la cota conseguida con la clase de funciones lineal. De cara a estos resultados, se seleccionó como clase definitiva la clase funciones cúbicas, como se explicó en el punto 7 de esta memoria.

10. DISCUTIR Y JUSTIFICAR LA CALIDAD DEL MODELO ENCONTRADO Y LAS RAZONES POR LAS QUE CONSIDERA QUE DICHO MODELO ES UN BUEN AJUSTE QUE REPRESENTA ADECUADAMENTE LOS DATOS MUESTRALES

Ambos problemas han demostrado una mejora del ajuste a mayor dimensionalidad, lo que hace pensar que, a mayor número de atributos de entrada, menor cota de error. Pero este aumento de dimensionalidad lleva consigo una complejidad de la función objetivo. Al ser una función objetivo más compleja, podría ocurrir que se sobreajustara la función objetivo a la muestra, de manera que, una clase de función junto con un modelo que de primeras parece ser muy bueno, consigue resultados malísimos en el conjunto de test.

En el caso del problema de clasificación parece que se ha conseguido un ajuste muy bueno. En cambio, en el problema de regresión puede haberse sobreajustado la función objetivo al conjunto de entrenamiento.

Clasificación

Como cota de error se estimó un error de 0.2848 (*Ilustración 13*), en cambio se consiguió un error de 0.0417 (*Ilustración 6*), mucho más pequeño de lo esperado, pero dentro del intervalo calculado. Con este resultado, se puede considerar que el ajuste conseguido en este problema es bastante bueno, ya que se estima que tendrá un acierto superior a un 71.52%. Es decir, se espera que sea capaz de clasificar de manera correcta bastante más de la mitad de los datos. Parece ser una cota bastante baja, en cambio se consigue un acierto superior al 95% al probar la función objetivo en el conjunto de test.

Que funcione bien con el conjunto de test no implica que vaya a funcionar igual en el resto de la población, pero si parece que la función objetivo va a ser bastante buena prediciendo futuras instancias.

Regresión

En este problema, se estimó una cota de error de 0.0679 (*Ilustración 13*), en cambio, se obtuvo un error en el test de 0.0796 (*Ilustración 13)*. El error conseguido es 1.173 veces superior a la cota esperada. El error conseguido sigue siendo muy bajo, pero al tener una cota de error tan

baja y haberla superado al predecir el conjunto de test, puede pensarse que la función objetivo esté sobreajustada y, por tanto, no vaya a garantizar resultados satisfactorios en la población.

Como el error conseguido en el test es muy bajo, aunque sea superior al esperado, también puede que se haya conseguido una función que cuenta con un leve sobreajuste pero que proporciona resultados satisfactorios. Por esta razón se comprueba el error que se consigue al utilizar la clase de funciones cuadrática como clase candidata y se obtiene un error de 0.1540 (*Ilustración 15)*, bastante superior al conseguido con el ajuste anterior, pero, a diferencia de ese error, este me mantiene por debajo de la cota estimada, siendo 0.1733 (*Ilustración 13*) dicha cota.

```
-> Regresión (cuadrática):     15.397597% de error en el conjunto de test
```

Ilustración 15 - Error conseguido con la clase de funciones cuadrática

Por tanto, si se quisiera asegurar conseguir un error pequeño a riesgo de tener un comportamiento malo en la población, se escogería la clase de funciones cúbica, ya que es el que menor error consigue en todos los casos. En cambio, si se considera una cota de error 2.4864 veces superior con mayor fiabilidad de conseguir un error en la población menor a dicha cota, se seleccionaría la clase de funciones cuadrática.

11. REFERENCIAS

scikit-learn developers. (2019). *sklearn.linear_model.LinearRegression*. Obtenido de scikitlearn: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

Abu-Mostafa, Y. S., Magdon-Ismail, M., & Lin, H.-T. (2012). *Learning From Data A Short Course.* AMLbook.com.

Aler Mur, R. (s.f.). *Aprendizaje Automático para el Análisis de Datos.* Obtenido de http://ocw.uc3m.es/ingenieria-informatica/aprendizaje-automatico-para-el-analisis-de-datos-2013/ocw7-knn.pdf

Alpaydin, E., & Kaynak, C. (1 de Julio de 1998). *Optical Recognition of Handwritten Digits Data Set*. Obtenido de UCI Machine Learning Repository: https://archive.ics.uci.edu/ml/datasets/optical+recognition+of+handwritten+digits

Amber. (1 de Junio de 2010). *Python: Indexing a file that is tab delimited*. Obtenido de stackoverflow: https://stackoverflow.com/questions/3160152/python-indexing-a-file-that-is-tab-delimited

Brooks, T. F., Pope, D. S., & Marcolini, M. A. (1989). *Airfoil Self-Noise and Prediction.* Hampton, Virginia: NASA Reference Publication 1218.

Brooks, T. F., Pope, D. S., & Marcolini, M. A. (Julio de 1989). *Airfoil Self-Noise Data Set*. Obtenido de UCI Machine Learning Repository: https://archive.ics.uci.edu/ml/datasets/airfoil+self-noise

Cobo Rodríguez, B., & Lara Porras, A. M. (2016). *Práctica 3 Regresión y Correlación*. Obtenido de Estadística Universidad de Granada: http://wpd.ugr.es/~bioestad/guia-r-commander/practica-3/

DataCamp. (s.f.). *DataCamp*. Obtenido de Pythos For Data Science Cheat Sheet Matplotlib: https://s3.amazonaws.com/assets.datacamp.com/blog_assets/Python_Matplotlib_Cheat_Sheet.pdf

DataCamp. (s.f.). *Python For Data Science Cheat Sheet*. Obtenido de DataCamp: https://s3.amazonaws.com/assets.datacamp.com/blog_assets/Numpy_Python_Cheat_Sheet.pdf

DataCamp. (s.f.). *Python For data Science Cheat Sheet Scikit-Learn.* Obtenido de DataCamp: https://s3.amazonaws.com/assets.datacamp.com/blog_assets/Scikit_Learn_Cheat_Sheet_Python.pdf

Hendrik Metzen, J. (s.f.). *Comparison of kernel ridge regression and SVR*. Obtenido de scikitLearn: https://scikit-

learn.org/stable/auto_examples/plot_kernel_ridge_regression.html#sphx-glr-auto-examples-plot-kernel-ridge-regression-py

Lara Porras, A. M. (2017). *Práctica 3 - Redresión y Correlación*. Obtenido de Estadística Universidad de Granada: http://wpd.ugr.es/~bioestad/guia-r-studio/practica-3/

Martinez Heras, J. (14 de Marzo de 2019). *Regularización Lasso L1, Ridge L2 y ElasticNet*. Obtenido de IArtificial.net Inteligencia Artificial y Machine Learning: https://iartificial.net/regularizacion-lasso-l1-ridge-l2-y-elasticnet/

Minitab. (2019). *Relaciones lineales, no lineales y monótonas*. Obtenido de Soporte de Minitab 18: https://support.minitab.com/es-mx/minitab/18/help-and-how-to/statistics/basic-statistics/supporting-topics/basics/linear-nonlinear-and-monotonic-relationships/

RV, R., Lemaitre, G., & Unterthiner, T. (s.f.). *Compare the effect of different scalers on data with outliers*. Obtenido de scikitLearn: https://scikit-learn.org/stable/auto_examples/preprocessing/plot_all_scaling.html#sphx-glr-auto-examples-preprocessing-plot-all-scaling-py

scikit-learn . (2019). *sklearn.cross_decomposition.CCA*. Obtenido de scikitLearn: https://scikit-learn.org/stable/modules/generated/sklearn.cross_decomposition.CCA.html

scikit-learn. (2019). *sklearn.preprocessing.StandardScaler*. Obtenido de scikitLearn: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

scikit-learn developers . (2019). *sklearn.model_selection.StratifiedKFold*. Obtenido de scikitlearn: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html

scikit-learn developers. (2019). *1.1. Generalized Linear Models*. Obtenido de scikitLearn: https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

scikit-learn developers. (2019). *3.2. Tuning the hyper-parameters of an estimator*. Obtenido de scikitLearn: https://scikit-learn.org/stable/modules/grid_search.html#multimetric-grid-search

scikit-learn developers. (2019). *3.3. Model evaluation: quantifying the quality of predictions*. Obtenido de scikitLearn: https://scikit-learn.org/stable/modules/model_evaluation.html

scikit-learn developers. (2019). *3.3. Model evaluation: quantifying the quality of predictions*. Obtenido de scikitLearn: https://scikit-learn.org/stable/modules/model_evaluation.html#scoring-parameter

scikit-learn developers. (2019). *5. Dataset transformation*. Obtenido de scikitLearn: https://scikit-learn.org/stable/data_transforms.html

scikit-learn developers. (2019). *5.3. Preprocessing data*. Obtenido de scikitLearn : https://scikit-learn.org/stable/modules/preprocessing.html#preprocessing

scikit-learn developers. (2019). *Glossary of Common Terms and API Elements*. Obtenido de scikitLearn: https://scikit-learn.org/stable/glossary.html#term-cv-splitter

scikit-learn developers. (2019). *L1 Penalty and Sparsity in Logistic Regression*. Obtenido de scikitLearn: https://scikit-learn.org/stable/auto_examples/linear_model/plot_logistic_l1_l2_sparsity.html

scikit-learn developers. (2019). *Logistic Regression 3-class Classifier*. Obtenido de scikitLearn: https://scikit-learn.org/stable/auto_examples/linear_model/plot_iris_logistic.html#sphx-glr-auto-examples-linear-model-plot-iris-logistic-py

scikit-learn developers. (2019). *sklearn.linear_model.LogisticRegression*. Obtenido de scikitLearn: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

scikit-learn developers. (2019). *sklearn.linear_model.LogisticRegression*. Obtenido de scikitLearn: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

scikit-learn developers. (2019). *sklearn.linear_model.LogisticRegressionCV*. Obtenido de scikitLearn: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegressionCV.html

scikit-learn developers. (2019). *sklearn.linear_model.Perceptron*. Obtenido de scikitLearn: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Perceptron.html

scikit-learn developers. (2019). *sklearn.metrics.accuracy_score*. Obtenido de scikitLearn: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html#sklearn.metrics.accuracy_score

scikit-learn developers. (2019). *sklearn.metrics.confusion_matrix*. Obtenido de scikitLearn: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html#sklearn.metrics.confusion_matrix

scikit-learn developers. (2019). *sklearn.metrics.matthews_corrcoef*. Obtenido de scikitLearn: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.matthews_corrcoef.html

scikit-learn developers. (2019). *sklearn.metrics.mean_squared_error*. Obtenido de scikitLearn: https://scikit-

learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html#sklearn.metrics.mean_squared_error

scikit-learn developers. (2019). *sklearn.model_selection.cross_val_score*. Obtenido de scikitLearn: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html#sklearn.model_selection.cross_val_score

scikit-learn developers. (2019). *sklearn.model_selection.GridSearchCV*. Obtenido de scikitLearn: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html#sklearn.model_selection.GridSearchCV.score

scikit-learn developers. (2019). *sklearn.model_selection.KFold*. Obtenido de scikitLearn: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html#sklearn.model_selection.KFold

scikit-learn developers. (2019). *sklearn.model_selection.train_test_split*. Obtenido de scikitLearn: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

scikit-learn developers. (2019). *sklearn.preprocessing.OneHotEncoder*. Obtenido de scikitLearn: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html#sklearn.preprocessing.OneHotEncoder

scikit-learn developers. (2019). *sklearn.preprocessing.PolynomialFeatures*. Obtenido de scikitLearn: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html#sklearn.preprocessing.PolynomialFeatures

scikit-learn developers. (2019). *sklearn.preprocessing.PowerTransformer*. Obtenido de scikitLearn: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PowerTransformer.html#sklearn.preprocessing.PowerTransformer

scikit-learn developers. (s.f.). *sklearn.feature_selection.VarianceThreshold*. Obtenido de scikitLearn: https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.VarianceThreshold.html

scikit-learn developers. (s.f.). *sklearn.preprocessing.MinMaxScaler*. Obtenido de scikitLearn: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html

The SciPy community. (23 de Agosto de 2018). *numpy.delete*. Obtenido de SciPy.org: https://docs.scipy.org/doc/numpy-1.15.1/reference/generated/numpy.delete.html

The SciPy community. (31 de Enero de 2019). *numpy.loadtxt*. Obtenido de SciPy.org: https://docs.scipy.org/doc/numpy/reference/generated/numpy.loadtxt.html

The SciPy community. (31 de Enero de 2019). *numpy.unique*. Obtenido de SciPy.org: https://docs.scipy.org/doc/numpy/reference/generated/numpy.unique.html

Variable Categórica. (22 de Diciembre de 2017). Obtenido de Wikipedia: https://es.wikipedia.org/wiki/Variable_categórica