APRENDIZAJE AUTOMÁTICO

DOCUMENTACIÓN PRÁCTICA 3:

Ajustes de Modelos Lineales

Universidad de Granada

GRUPO A3
(Viernes 17:30-19:30)

CURSO 2018-2019

PROFESOR: Pablo Mesejo Santiago

Realizado por:
Jesús Medina Taboada
DNI:
jemeta@correo.ugr.es

ÍNDICE

Problema de clasificación

Problema de regresión

INTRODUCCIÓN

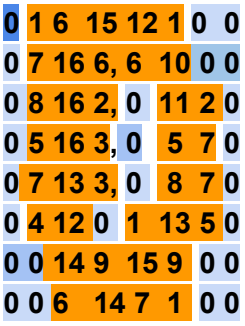
Este ejercicio se centra en el ajuste de un modelo lineal a conjuntos de datos dadas con el objetivo de obtener el mejor predictor posible. En todos los casos los pasos a desarrollar serán aquellos que nos conduzcan al ajuste y selección del mejor modelo y a la estimación del error Eout del modelo final. Seguiremos unos puntos de análisis para cada uno de los dos problemas que veremos centrándonos en el análisis y la explicación de por qué hemos tomado estas decisiones.

1.PROBLEMA DE CLASIFICACIÓN

1.1 Comprender el problema a resolver

Los datos que usaremos a continuación los hemos extraído de un problema de dígitos manuscritos, el cual se basa en el reconocimiento óptico de un conjunto de datos de números escritos a mano.
La base de datos consta de 5620 instancias, con un total de 64 atributos de tipo entero y esta base ha sido realizada por un total de 43 personas, de las cuales 30 contribuyeron al conjunto de datos train y el resto al de test.
Los mapas de bits de 32x32 se dividen en bloques no solapables de 4x4 y se cuenta el número de píxeles con valor igual a 1 en cada bloque. Esto genera una matriz de entrada de 8x8 donde cada elemento es un entero en el rango [0, 16].

Un ejemplo: se puede entrever que es el número cero



1.2 Preprocesado de los datos

Para que sea más fácil manejar los datos hemos hecho uso de normalización.
Para ello, hemos dividido cada elemento entre 16 ya que es el máximo valor que puede alcanzar un atributo, quedando todos nuestros datos normalizados en el intervalo [0, 1].

```
# Funcion para leer los datos
def readData(file_x, file_y):
    x = np.load(file_x)
    y = np.load(file_y)
    x_ = np.empty(x.shape, np.float64)

    for i in range(0,x.shape[0]):
        for j in range(0,x.shape[1]):
            x_[i][j] = np.float64(1.0*x[i][j]/16.0)

    return x_, y
```

El segundo preprocesamiento de datos ha consistido en eliminar los atributos con valores muy bajo (por debajo de 0.01) porque no proporcionarán apenas información.

```
sel = VarianceThreshold(threshold=(0.01))
X_train = sel.fit_transform(X_train)
```

Por último añadimos características polinómicas de orden 2 a nuestros datos para tener más información de los atributos y compensar.

```
poly = PolynomialFeatures(2)
X_train= poly.fit_transform(X_train)
```

1.3 Selección de clases de funciones a usar

La clase de funciones que hemos elegido es la clase de funciones lineales y en concreto para la realización del problema hemos decidido usar regresión logística.

1.4 Definición de los conjuntos de training, validación y test usados en su caso

He pensado que lo mejor que puedo utilizar en este caso es validación para comprobar el comportamiento del clasificador evitando sobreajuste.
Usaremos el tipo de validaciones “Cross Validation” (K-fold).
Primero separaremos los datos del conjunto train en K conjuntos y escogeremos uno de estos conjuntos como datos de test y el resto como train. Las divisiones en K elementos tendrán una distribución de etiquetas equilibrada o stratified.

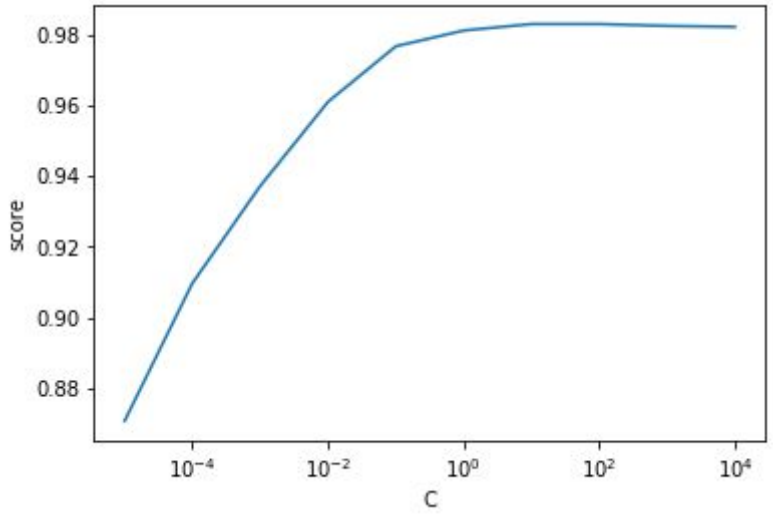
Este proceso lo realizaremos k veces eligiendo cada vez un conjunto como datos de prueba y al final realizaremos una media de los resultados obtenidos en cada iteración obteniendo un único resultado.

```
# Cross Validation
# KFold
k = 2
kf = StratifiedKFold(n_splits=k)
kf.get_n_splits(X_train,y_train)

mejor_C = 0
mejor_media = 0

x = []
y = []

for i in range(-5,5):
    suma = 0
    c = 10**i

    for train_index, test_index in kf.split(X_train,y_train):
        X_train_, X_test_ = X_train[train_index], X_train[test_index]
        y_train_, y_test_ = y_train[train_index], y_train[test_index]

        lr = LogisticRegression(C=c)
        lr.fit(X_train_, y_train_)
        suma += lr.score(X_test_, y_test_)

    media = 1.0*suma/k

    x.append(c)
    y.append(media)

    if media > mejor_media:
        mejor_media = media
        mejor_C = c
```

1.5 Discutir la necesidad de regularización y, en su caso, la función usada para ello

En mi caso usaremos un parámetro correspondiente a la inversa de la regularización, el elemento ‘c’ del modelo de regresión logística.
Este en el programa tomará los valores $c = 10^{i} \quad i = \{-5,-4,-3,-2,-1,0,1,2,3,4,5\}$.
Para justificar esta regulación veremos un ejemplo con algunos modelos de regresión logística dados diferentes valores de ‘c’ .

En la siguiente gráfica podemos observar una relación entre el porcentaje de acierto y el valor de ‘c’. Este empieza ascendiendo notablemente pero cuando llega al valor c= 10^{0} el porcentaje de acierto empieza a disminuir y esto es debido al sobreajuste de nuestro modelo.



1.6 Definir los modelos lineales a usar y estimar sus parámetros
Como llevamos argumentando en este ejemplo usaremos una regresión logística en la que modificaremos el parámetro ‘c’ que es la inversa de la regularización. Los pasos a seguir serán:

1.Probar distintos valores de ‘c’
2.Validar cada modelo con la técnica kFolf de Cross Validation
3.Nos quedaremos con el parámetro ‘c’ que nos de mejores resultados

1.7 Selección y ajuste del modelo final
Para seleccionar finalmente el modelo aplicamos la validacion kFold a cda uno de las variaciones obtenidas al modificar el parámetro c. Calculamos la media de las puntuaciones obtenidas de cada subconjunto y guardamos el mejor resultado.(código del apartado 4)

1.8 Discutir la idoneidad de la métrica usada en el ajuste
Para empezar tenemos claro nuestro objetivo en este problema que es clasificar los dígitos de la forma más precisa posible minimizando el número de errores.
Para ver de una forma clara esto usaremos una matriz de confusión que nos premitirá ver que variables “se confunden”.

```
print ("\nMatriz de confusión:")
cm = metrics.confusion_matrix(y_test, predictions)
plt.figure(figsize=(9,9))
sns.heatmap(cm, annot=True, fmt=".3f", linewidths=.5, square = True);
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
all_sample_title = 'Accuracy Score: {0}'.format(score)
plt.title(all_sample_title, size = 15);
```

Por otra parte necesitaremos saber el porcentaje de acierto que hemos obtenido fuera de la muestra con los datos del conjunto de test.

```
lr = LogisticRegression(C=mejor_C)

lr.fit(X_train, y_train)
predictions = lr.predict(X_test)
score = lr.score(X_test, y_test)
```

1.9 Estimación del error Eout del modelo lo más ajustada posible

Para calcular el error del conjunto test simplemente he restado 1 al mejor valor obtenido de la función score cuando hicimos la validación de modelos.
Por otra parte para obtener el Eout usamos la fórmula

$$Eout \leq Etest + \sqrt{\frac{1}{2N} ln(2M/\delta)}$$

Para M = 1 y δ=0.05

```
Etest = 1-mejor_media
print('Etest: 1 - score_validacion = ', 1-mejor_media)
print('Cota de Eout:', Etest+np.sqrt(1/(2*X_train.shape[1]) * np.log(2/0.05) ) )
```

Obtenemos los siguientes resultados

```
Etest: 1 - score_validacion =  0.01699725164799859
Cota de Eout: 0.05743413582570024

Score obtenido en el test: 0.9760712298274903
Eout: 1 - score_test =  0.023928770172509717

Mejor Score obtenido en la validación:
0.9830027483520014
```

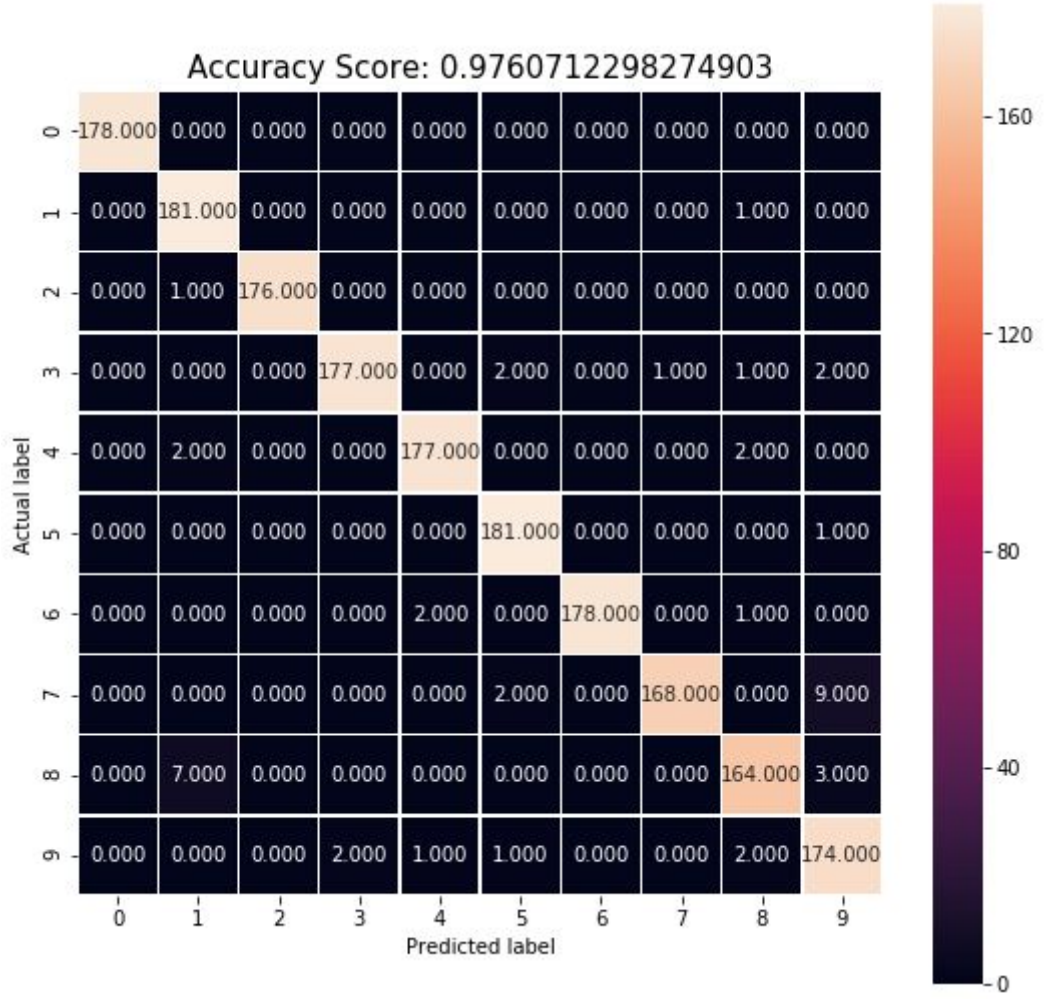
Podemos destacar que el Eout tiene un valor muy bueno lo que nos indica que el error fuera de la muestra será muy bajo y además la cota que hemos cogido tiene una confianza del 95%.

1.10 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.

A continuación mostramos la matriz de confusión y podemos ver que los mayores valores salen en la diagonal de la misma. Con esto no podemos afirmar que se haya conseguido un acierto en la mayoría de las veces.

Un dato a destacar sería que se ha cometido bastante error en clasificar el número 7 y el 9 y tiene sentido ya que se parecen mucho al escribirse pero en comparación al acierto sigue siendo un error muy bajo.

Al final la accuracy es del 97,6% un porcentaje que roza el 100% por lo que podemos concluir que nuestro ajuste ha sido muy acertado y bastante bueno en términos generales.



2.PROBLEMA DE REGRESIÓN

1.1 Comprender el problema a resolver

La base de datos utilizada se llama Airfoil Self-Noise, es un conjunto de datos de la Nasa, obtenidos a partir de una serie de pruebas aerodinámicas y acústicas de secciones de palas de aerodinámica bidimensionales y tridimensionales realizadas en un túnel aerodinámico anecoico.
La base de datos consta de 1503 instancias. Las columnas de este conjunto de datos son los siguientes:

1. Frecuencia, en hercios.
2. Ángulo de ataque, en grados.
3. Longitud del acorde, en metros.
4. Velocidad de flujo libre, en metros por segundo.
5. Espesor de desplazamiento del lado de succión, en metros.
6. Nivel de presión sonora escalonado, en decibelios.

De las cuales las 5 primeras corresponden a los atributos y la última a la salida, nuestro objetivo es intentar predecir esta última variable.

1.2 Preprocesado de los datos

Los datos que obtuvimos no estaban separadaos en train y test por lo que hemos tenido que hacerlo nosotros, en mi caso he escogido la partición 30% test y 70% training.
Como preproceso he permutado los datos antes de su separación en train y test ya que se puede observar que tienen un orden debido a los atributos lo que pienso que perjudica el trabajo.

```
# Leemos el conjunto de entrenamiento
X, y = readData('datos/airfoil_self_noise_X.npy',
                'datos/airfoil_self_noise_y.npy')
```

```
# Permutamos los datos antes de separar en train y test
random_state = check_random_state(0)
permutation = random_state.permutation(X.shape[0])
X = X[permutation]
y = y[permutation]
```

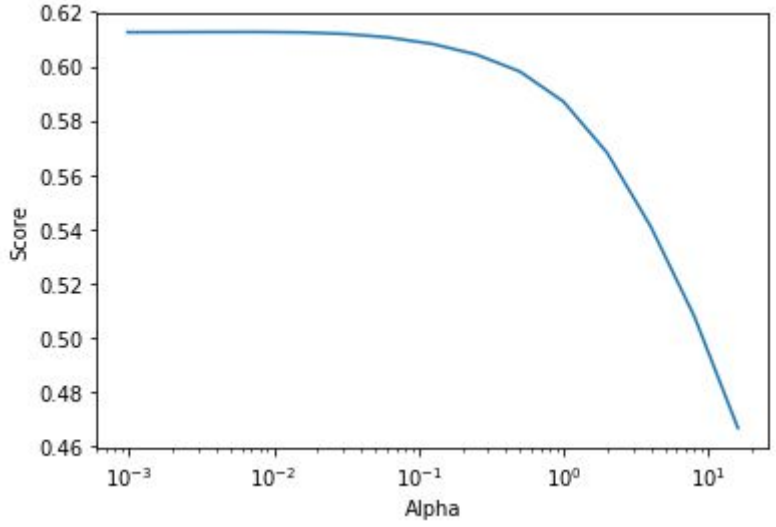
```
# Separamos los datos en train y test
X_train = X[0:2*X.shape[0]//3]
y_train = y[0:2*y.shape[0]//3]
X_test = X[2*X.shape[0]//3:X.shape[0]]
y_test = y[2*y.shape[0]//3:y.shape[0]]
```

Por último como los rangos de los datos eran muy dispares los valores más altos podrían predominar sobre los baos por lo que he realizado un escalado.
Además añadimos características polinómicas de grado 2 al problema cosa que será muy beneficiosa porque tenemos pocos atributos en el problema.

```
# Escalamos los datos
min_max_scaler = preprocessing.MinMaxScaler()
X = min_max_scaler.fit_transform(X)

poly = PolynomialFeatures(2)
X = poly.fit_transform(X)
```

1.3 Selección de clases de funciones a usar
La clase de funciones a usar son las funciones lineales y el modelo que utilizaremos para la regresión es será el modelo Ridge
.
1.4 Definición de los conjuntos de training, validación y test usados en su caso
Al igual que en el ejercicio 1 de clasificación usaremos la técnica de validación cruzada KFold, la diferencia más notable es que en el problema anterior pusimos un valor k=2 por el tiempo de ejecución pero en este caso como el número de atributos es muy inferior pondremos n k=5.

1.5 Discutir la necesidad de regularización y, en su caso, la función usada para ello
Como he dicho usamos un modelo de regresión Ridge que gira entorno al parámetro alfa que controla la regularización misma.(a diferencia de en el ejercicio de clasificación el cual se correspondía con la inversa.
A continuación se muestra una gráfica entre el porcentaje de acierto y el valor de alfa.
Este empieza decreciendo muy lentamente hasta que la regularización incremente lo que conlleva el descenso del score. Por lo que concluimos que a menor regularización, mejores resultados.



1.6 Definir los modelos lineales a usar y estimar sus parámetros

Usamos el modelo de regresión Ridge y obtendremos varios modelos correspondientes a los diferentes valores que le demos a la variable alfa que usaremos para variar la regularización.

```
        lr = linear_model.Ridge(alpha = a)
        lr.fit(X_train_, y_train_)
```

1.7 Selección y ajuste del modelo final

Seguiremos los siguientes pasos para elegir nuestro modelo definitivo:

1.Usamos cross validation para cada modelo obtenido al modificar alfa

2.Hacemos media de resultados obtenido con cada alfa por cada subconjunto usado como test

3.Elegimos el modelo que mejores resultados obtenga.

```
# KFold
k = 5
kf = KFold(n_splits=k)

mejor_alpha = 0
mejor_media = 0

alphas = []
scores = []

for i in range(-10,5):
    suma = 0
    a = 2**i
    for train_index, test_index in kf.split(X_train):
        X_train_, X_test_ = X_train[train_index], X_train[test_index]
        y_train_, y_test_ = y_train[train_index], y_train[test_index]

        lr = linear_model.Ridge(alpha = a)
        lr.fit(X_train_, y_train_)
        suma += lr.score(X_test_, y_test_)

    media = suma/k

    alphas.append(a)
    scores.append(media)

    if media > mejor_media:
        mejor_media = media
        mejor_alpha = a
```

1.8 Discutir la idoneidad de la métrica usada en el ajuste

Para comprobar la idoneidad de la métrica que hemos usado en el ajuste haremos uso de la función score del modelo Ridge que nos devuelve el coeficiente de determinación R^2, el cual toma valores entre 0 y 1, aunque debido a la implementación computacional score puede llegar a devolver valores negativos. Cuanto más cercano sea el valor a 1 mejor explicará el modelo y por lo tanto menor sería el error de predicción.

1.9 Estimación del error Eout del modelo lo más ajustada posible
El Etest lo calculamos como el resultado de restar a 1 el meor valor obtenido del coeficionete den la validación.

Y para la cota del Eout usamos la fórmula

$$Eout \leq Etest + \sqrt{\frac{1}{2N} ln(2M/\delta)}$$

```
Etest: 1 - score_validacion =  0.38735252694135613
Cota de Eout: 0.6837145744325774
```

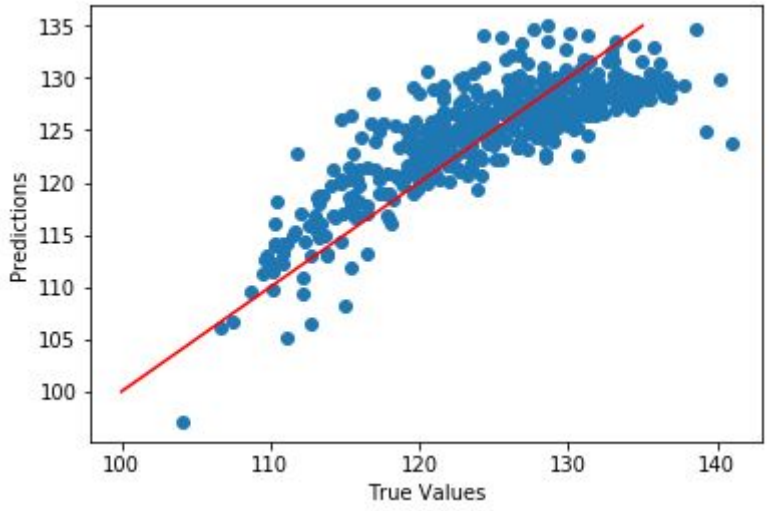
1.10 Discutir y justificar la calidad del modelo encontrado y las razones por las que considera que dicho modelo es un buen ajuste que representa adecuadamente los datos muestrales.

Tras obtener el valor de alpha que mejores resultados a dado en la validación, procedemos a ver la calidad de nuestro modelo, obteniendo la puntuación de este modelo sobre los datos de test.

```
# Ridge
lr = linear_model.Ridge(alpha = mejor_alpha)
# Entrenamos el modelo
model = lr.fit(X_train, y_train)
```

El valor obtenido en el score es 0.661774377836111, que se corresponde con el valor del coeficiente de determinación. A continuación se muestra una gráfica de los valores reales junto con sus predicciones. Se muestra también la función f(x) = x para ver bien la desviación de las predicciones.

```
Mejor Score obtenido en la validación:
0.6126474730586439
```



El valor del coeficiente de determinación obtenido es de 0.66, con lo que nuestro modelo explica bastante bien las relaciones entre los datos. Además en la gráfica anterior vemos como los puntos están entorno a la diagonal, lo cual es un buen indicativo de que nuestro modelo es bueno ya que las predicciones se desvían poco de su valor real. Por último se ha realizado también el cálculo de el error cuadrático medio obteniendo un valor de 15.80, un aceptable teniendo en cuenta que los valor de las etiquetas se mueven entre 100 y 140.