Trabajo 3 - Aprendizaje Automático

ugr | Universidad de Granada

Francisco Solano López Rodríguez

Índice

1. Problema de clasificación

1. **Comprender el problema a resolver.**

   La base de datos utilizada se corresponde con datos de dígitos manuscritos.

   El problema a realizar se basa en el reconocimiento óptico de un conjunto de datos de dígitos manuscritos. Esta consta de 5620 instancias, con un total de 64 atributos de tipo entero.

   La base de datos ha sido realizada por un total de 43 personas, 30 contribuyeron al conjunto de datos *train* y 13 para el conjunto de *test*.

   Los mapas de bits de 32×32 se dividen en bloques no solapables de 4×4 y se cuenta el número de píxeles con valor igual a 1 en cada bloque. Esto genera una matriz de entrada de 8×8 donde cada elemento es un entero en el rango $[0, 16]$; esto reduce la dimensionalidad y da invariancia a pequeñas distorsiones.

   A continuación se muestra la representación de una pequeña muestra de la base de datos:

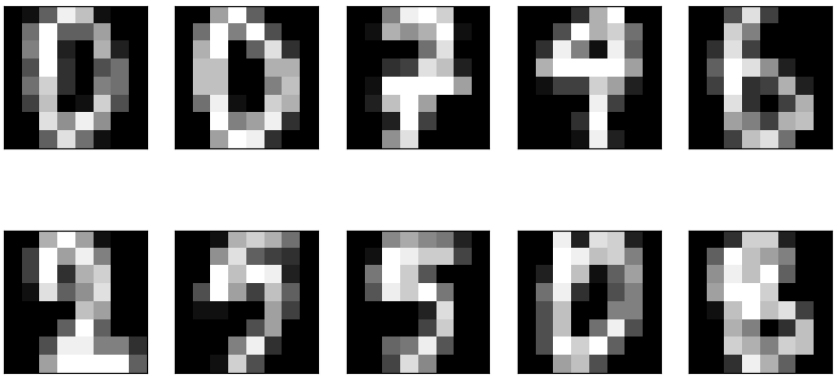

Figura 1: Representación de algunos dígitos.

2. **Preprocesado los datos: por ejemplo categorización, normalización, etc.**

   Hemos usado normalización para el preprocesado de los datos. Para ello, hemos dividido cada elemento por 16, pues es el máximo valor en el rango numérico de los atributos, quedando todos nuestros datos normalizados en el intervalo $[0, 1]$.

   Dicha normalización la hemos implementado mediante el siguiente código:

```
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
```

   Tras ello se han eliminado aquellas características con una varianza muy baja, ya que apenas van a proporcionar información:

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

print('\nLas partes negras corresponten a caracteristicas eliminadas')
plt.imshow(np.split(sel.get_support(), 8), cmap='gray')
plt.show()
```

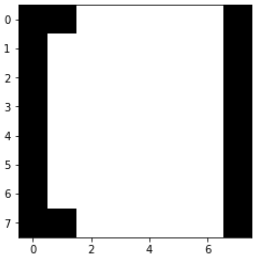

Figura 2: Características eliminadas (en negro)

Por último se han añadido características polinómicas de orden 2, para tener más información de atributos.

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

3. **Selección de clases de funciones a usar.**

   La clase de funciones a usar se corresponde con las clase de las funciones lineales. Para la realización del problema hemos decidido usar regresión logística.

4. **Definición de los conjuntos de training, validación y test usados en su caso.**

   Usaremos validación para evaluar nuestro clasificador y calcular `c` para evitar sobreajuste; en concreto, usaremos "Validación Cruzada" (*K-fold*).

   Dicha técnica consiste en separar los datos de train en K conjuntos, escogiendo uno de ellos como datos de prueba y el resto como datos de entrenamiento; estas K particiones será disjuntas con la distribución de etiquetas equilibrada (*Stratified*). Dicho proceso se repite K veces, eligiendo cada vez uno de los subconjuntos como datos test. Para terminar, realizaremos la media de los resultados obtenidos en cada iteración para obtener un único resultado.

   El código usado para implementar dicha técnica será el siguiente.

```
# KFold
k = 2 # Número de subconjuntos a generar
kf = StratifiedKFold(n_splits=k) # Inicializamos el K-Fold
kf.get_n_splits(X_train,y_train) # Separamos los conjuntos de entrenamiento
```

```
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]
```

   El valor de k se ha fijado solo a 2 para una ejecución mas rápida, ya que al tener muchos atributos se tarda más en ejecutar.

5. **Discutir la necesidad de regularización y en su caso la función usada para ello.**

   Se ha utilizado el modelo de regresión logística, el cual tiene un parámetro c que se corresponde con la inversa de la regularización (λ).

   Este parámetro tomara los valores $\{c = 10 * *i\}$ con $i = -5, 4, ..., 3, 4$.

   Para ver la necesidad de regularización crearemos varios modelos de regresión logística con diferentes valores del parámetro c.

   En la siguiente gráfica, podemos ver como cambia el porcentaje de acierto para cada c^i:

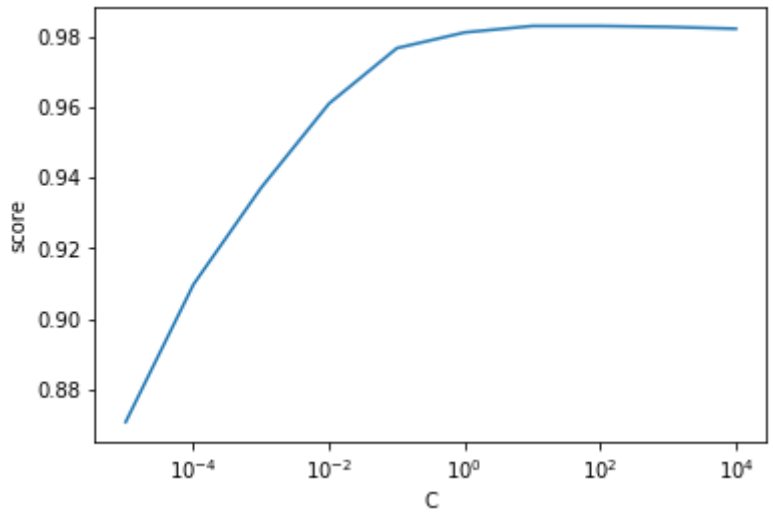

Figura 3: Porcentaje de acierto respecto del valor C. (escala log)

Aunque no se aprecia bien en la gráfica a partir del valor 10^1, el valor del score empieza a disminuir lentamente, lo cual indica que al diminuir cada vez más la regularización se obtiene valores cada vez peores debido al sobreajuste.

El código con el que hemos obtenido esta gráfica se mostrará después.

6. **Definir los modelos a usar y estimar sus parámetros e hyperparámetros.**

   Como hemos dicho antes vamos a utilizar regresión logística. En ella vamos a hacer modificaciones del parámetro c, el cual es inversamente proporcional a la regularización. Para ello vamos a probar distintos valores de c, vamos a validar cada modelo con a técnica de kFold explicada anteriormente, y nos vamos a quedar con el c que nos proporcione mejores resultados en la validación.

```
lr = LogisticRegression(C=c)
lr.fit(X_train_, y_train_)
```

   La modificación del parámetro c se va a hacer de forma exponencial, $c = \{10^{-5}, 10^{-4}, \cdots, 10^{3}, 10^{4}\}$.

7. **Selección y ajuste modelo final.**

   Para la selección del modelo, aplicamos la técnica de validación kFold sobre cada uno de los modelos obtenidos al variar el parámetro c. Se calcula la media de las puntuaciones obtenidas de cada partición y nos quedamos con aquel que nos de mejores resultados.

   El código completo es el siguiente:

```
# 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
```

8. **Discutir la idoneidad de la métrica usada en el ajuste.**

   En este problemas tenemos que clasificar dígitos y el objetivo es maximizar el acierto, o lo que es lo mismo minimizar el número de errores. Una forma de visualizar esto es hacer uso de la matriz de confusión, la cual nos permite ver facilmente que variables se están confundiendo.

   En el siguiente código se muestra una forma de visualizar la matriz 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);
```

   También necesitamos saber el porcentaje de acierto que hemos obtenido fuera de la muestra, con los datos del test.

```
lr = LogisticRegression(C=mejor_C)

lr.fit(X_train, y_train)
predictions = lr.predict(X_test)
score = lr.score(X_test, y_test)
print('\nScore obtenido en el test:', score)
```

9. **Estimación del error E_{out} del modelo lo más ajustada posible.**

   El Etest lo he cálculado como el resultado de restar a 1 el mejor valor de la tasa de acierto obtenida con la función .score en la validación de los modelos.

```
Etest = 1-mejor_media
```

   Obteniendo un valor de 0.01699725164799859.

   Para obtener una cota del Eout he utilizado la fórmula:

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

   Donde M = 1 y el valor de delta se ha fijado a 0.05.

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

   La cota del valor de Eout obtenida es: 0.05743413582570024. Es un valor muy bueno, el cual nos indica que el error fuera de la muestra va a ser muy bajo, teniendo en cuenta también que es una cota con una confianza del 95 %.

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.**

    Como podemos apreciar en la matriz de confusión, los mayores valores han salido en la diagonal, lo cual no indica que se ha conseguido un acierto en la mayoría de las veces. También podemos ver que ha habido errores en algunas predicciones como por ejemplo el 7 con un 9, pero los errores han sido muy bajos en proporción con los aciertos.

    EL porcentaje de acierto obtenido ha sido de 97,6 %, un valor muy ercano al 100 %, por lo que concluimos que el ajuste ha sido muy bueno.

    La matriz de confusión obtenida es la siguiente:

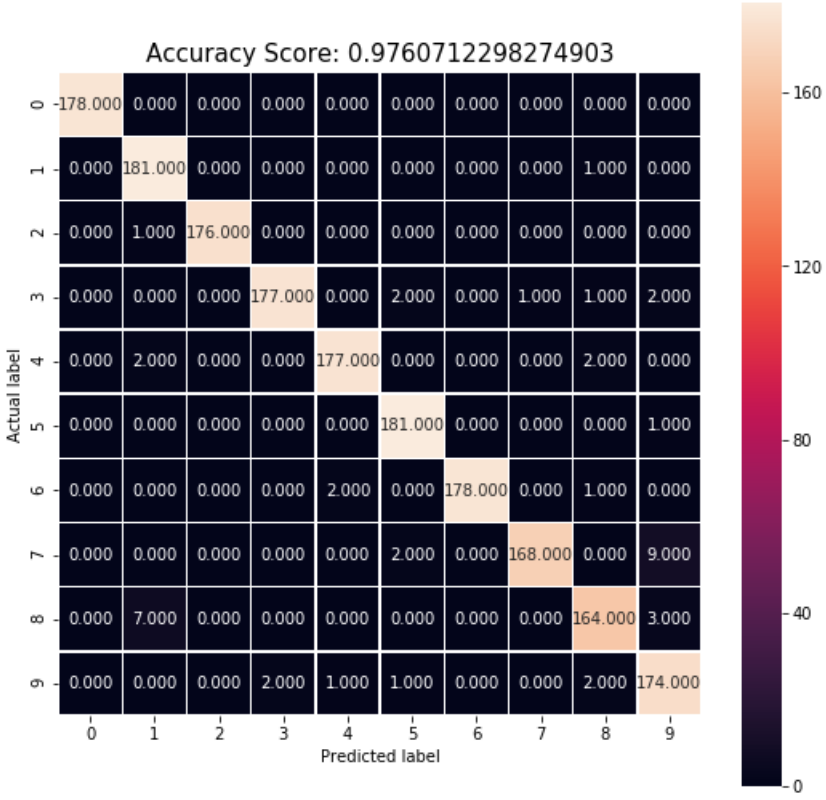

Figura 4: Matriz de confusión

2. Problema de regresión.

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.

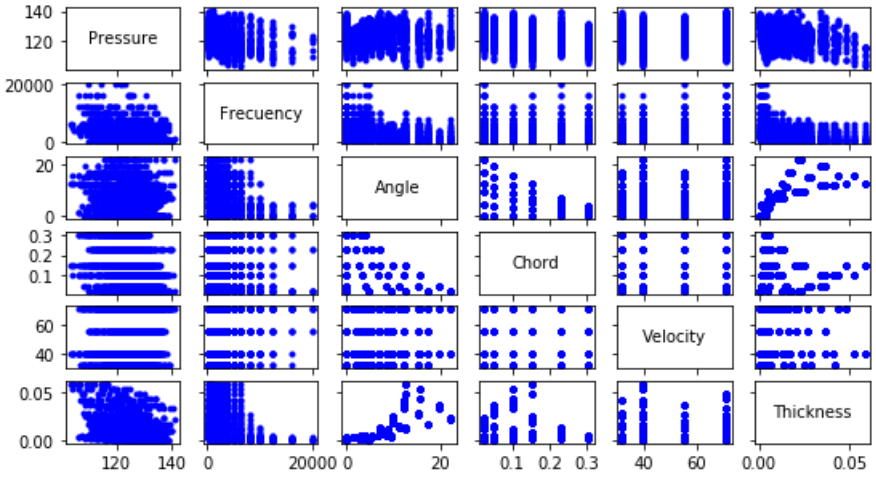

Figura 5: Matriz de confusión

En la gráfica anterior se muestran las gráficas de los atributos por pares. Viendo esto cuesta ver relaciones que puedan determinar el valor de la presión.

2. **Preprocesado los datos: por ejemplo categorización, normalización, etc.**

   Como nuestros datos no estaban separados en train y test hemos tomado un tercio de ellos para test y los restantes para training. Antes de la separación de los datos estos han sid permutados, ya que observe que la muestra de datos tenía un cierto orden debido a uno de los atributos.

```
# Leemos el conjunto de entrenamiento
X, y = readData('airfoil_self_noise_npy/airfoil_self_noise_X.npy',
'airfoil_self_noise_npy/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]

# Seramos 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]]
```

Como los datos se mueven en distintos rangos, aquellos que se mueven en valores más altos podrían dominar frente a los otros al realizar la regresión. Debido a ello los datos se han escalado con el siguiente código:

```
min_max_scaler = preprocessing.MinMaxScaler()
X = min_max_scaler.fit_transform(X)
```

Los datos escalados con el código anterior toman media cero y varianza unitaria.

Además se han añadido características polinómicas al problema, cosa que puede venir muy bien debido al bajo número de atributos de los que disponemos.

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

3. **Selección de clases de funciones a usar.**

   La clase de funciones a usar son las funciones lineales. El modelo que utilizaremos para la regresión es el llamado Ridge.

4. **Definición de los conjuntos de training, validación y test usados en su caso.**

   Al igual que en clasificación hemos realizado la técnica de validación cruzada Kfold. En el de clasificación teníamos un valor de K=2 debido al tiempo de ejecución. Como en el problema de ahora el número de características es bastante menor se ha puesto un valor de k=5.

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

```
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]
```

5. **Discutir la necesidad de regularización y en su caso la función usada para ello.**

   Se ha usado el modelo de regresión Ridge, el cual tiene un parámetro alfa que controla la regularización. En este caso al contrario que en el de clasificación (en el cual el parámetro c se correspondía con la inversa de la regularización $c = 1/\lambda$), el parámetro α se corresponde con la regularización (c^{-1}).

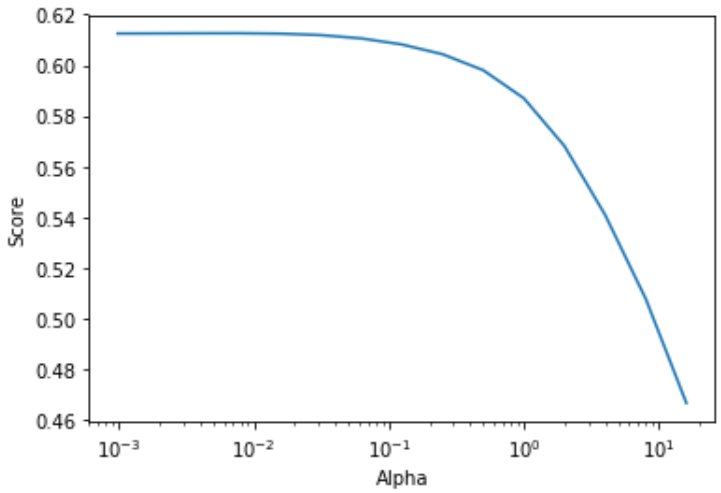

Figura 6

En esta gráfica observamos que se obtienen mejores resultados cuando hay poca regularización.

6. **Definir los modelos a usar y estimar sus parámetros e hyperparámetros.**

   Utilizando el modelo de regresión Ridge, vamos a obtener varios modelos mediante la variación del parámetro alpha, con el que vamos a variar regularización.

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

7. **Selección y ajuste modelo final.**

   Usamos la técnica de validación cruzada, para cada modelo obtenido al variar el parámetro alfa, hacemos la media de los resultados obtenidos con cada alfa por cada partición usada como test. Nos quedamos con el modelo que nos dé mejores resultados.

```
# 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

plt.plot(alphas,scores)
plt.xscale('log')
plt.xlabel('Alpha')
plt.ylabel('Score')
#plt.title('C')
plt.show()
```

8. **Discutir la idoneidad de la métrica usada en el ajuste.**

   Para ver la idoneidad vamos a hacer 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.

9. **Estimacion del error E out del modelo lo más ajustada posible.**

   El Etest lo he cálculado como el resultado de restar a 1 el mejor valor del coeficiente de determinación obtenido en la validación.

```
   Etest = 1-mejor_media
```

   Obteniendo un valor de 0.38735252694135625.

   Para obtener una cota del Eout he utilizado la fórmula:

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

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

La cota del valor de Eout obtenida es: 0.6837145744325777.

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)

# Check the score test
score = lr.score(X_test, y_test)
print('\nTest Score: ', score)
```

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.

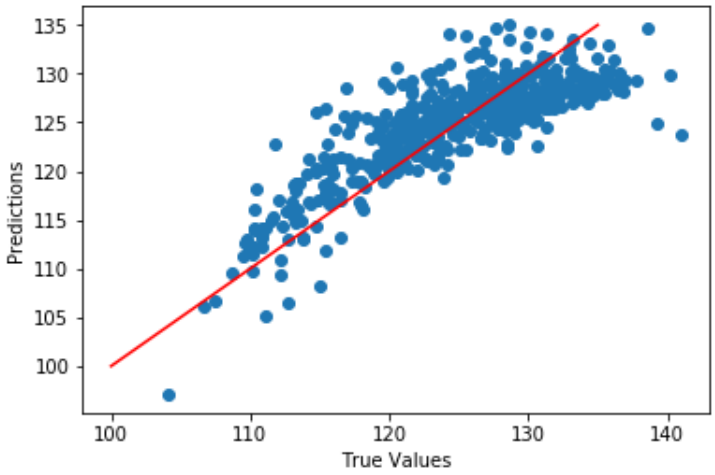

Figura 7: Predicciones

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.