redes-neuronales-a01633819

September 11, 2023

0.1 Actividad: Ajuste de redes neuronales

Andrés Alejandro Guzmán González A01633819

---

[3]:
```
# Llamado a librerías
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPRegressor, MLPClassifier
from sklearn.model_selection import KFold, cross_val_score, StratifiedKFold,␣
 ↪cross_val_predict, GridSearchCV
from sklearn.metrics import mean_squared_error, mean_absolute_error,␣
 ↪classification_report, accuracy_score
```

0.1.1 Ejercicio 1

El conjunto de datos de criminalidad de Estados Unidos publicado en el año 1993 consiste de 51 registros para los que se tienen las siguientes variables:

- **VR** = crímenes violentos por cada 100000 habitantes
- **MR** = asesinatos por cada 100000 habitantes
- **M** = porcentaje de áreas metropolitanas
- **W** = porcentaje de gente blanca
- **H** = porcentaje de personas con preparatoria terminada
- **P** = porcentaje con ingresos por debajo del nivel de pobreza
- **S** = porcentaje de familias con solo un miembro adulto como tutor.

Consideraciones:

- último nímero de matrícula = 9
  - Variable dependiente MR
  - variables independientes M, W, H, P y S

Para este conjunto de datos:

[4]:
```
df = pd.read_csv('/content/sample_data/crime_data.csv')
df.head()
```

[4]:

	State	VR	MR	M	W	H	P	S
0	AK	761	9.0	41.8	75.2	86.6	9.1	14.3
1	AL	780	11.6	67.4	73.5	66.9	17.4	11.5
2	AR	593	10.2	44.7	82.9	66.3	20.0	10.7
3	AZ	715	8.6	84.7	88.6	78.7	15.4	12.1
4	CA	1078	13.1	96.7	79.3	76.2	18.2	12.5

[5]:
```
df.isnull().sum()
```

[5]:
```
State    0
VR       0
MR       0
M        0
W        0
H        0
P        0
S        0
dtype: int64
```

[6]:
```
df = df.drop(['State', 'VR'], axis=1)
df.head()
```

[6]:

	MR	M	W	H	P	S
0	9.0	41.8	75.2	86.6	9.1	14.3
1	11.6	67.4	73.5	66.9	17.4	11.5
2	10.2	44.7	82.9	66.3	20.0	10.7
3	8.6	84.7	88.6	78.7	15.4	12.1
4	13.1	96.7	79.3	76.2	18.2	12.5

[7]:
```
# Variable de respuesta
y = np.array(df['MR'])
# Variables predictoras
x = np.array(df[['M', 'W', 'H', 'P', 'S']])

scaler = StandardScaler()
X_scaled = scaler.fit_transform(x)
```

Paso 1 Evalúa con validación cruzada un modelo pereceptrón multicapa para las variables que se te asignaron para este ejercicio.

[8]:
```
# Train classifier with all the available observations
clf = MLPRegressor(hidden_layer_sizes=(10, 10), max_iter=10000, random_state=0)
clf.fit(X_scaled, y)

# 5-fold cross-validation
kf = KFold(n_splits=5, shuffle=True)
cv_mse = []
```

```
cv_mae = []
for train_index, test_index in kf.split(X_scaled):
    x_train, x_test = X_scaled[train_index], X_scaled[test_index]
    y_train, y_test = y[train_index], y[test_index]

    regressor_cv = MLPRegressor(hidden_layer_sizes=(10, 10), max_iter=10000,␣
 ↪random_state=0)
    regressor_cv.fit(x_train, y_train)

    y_pred = regressor_cv.predict(x_test)
    mse = mean_squared_error(y_test, y_pred)
    cv_mse.append(mse)
    mae = mean_absolute_error(y_test, y_pred)
    cv_mae.append(mae)

# Calcular el error cuadrático medio promedio
mse = np.mean(cv_mse)
mae = np.mean(cv_mae)
print("MSE:", mse, "\nMAE:", mae)
```

```
MSE: 53.68201306374827
MAE: 3.526432750241672
```

Paso 2 Agrega al conjunto de datos columnas que representen los cuadrados de las variables predictoras (por ejemplo, M2, W2), así como los productos entre pares de variables (por ejemplo, PxS, MxW). Evalúa un modelo perceptrón multicapa para este nuevo conjunto de datos.

[9]:
```
df['MR'] = df['MR']
df['M2'] = df['M']**2
df['W2'] = df['W']**2
df['H2'] = df['H']**2
df['P2'] = df['P']**2
df['S2'] = df['S']**2

df['MW'] = df['M'] * df['W']
df['MH'] = df['M'] * df['H']
df['MP'] = df['M'] * df['P']
df['MS'] = df['M'] * df['S']

df['WH'] = df['W'] * df['H']
df['WP'] = df['W'] * df['P']
df['WS'] = df['W'] * df['S']

df['HP'] = df['H'] * df['P']
df['HS'] = df['H'] * df['S']

df['PS'] = df['P'] * df['S']
```

```
df.head()
```

[9]:

	MR	M	W	H	P	S	M2	W2	H2	P2	…	\
0	9.0	41.8	75.2	86.6	9.1	14.3	1747.24	5655.04	7499.56	82.81	…	
1	11.6	67.4	73.5	66.9	17.4	11.5	4542.76	5402.25	4475.61	302.76	…	
2	10.2	44.7	82.9	66.3	20.0	10.7	1998.09	6872.41	4395.69	400.00	…	
3	8.6	84.7	88.6	78.7	15.4	12.1	7174.09	7849.96	6193.69	237.16	…	
4	13.1	96.7	79.3	76.2	18.2	12.5	9350.89	6288.49	5806.44	331.24	…	

	MW	MH	MP	MS	WH	WP	WS	HP	\
0	3143.36	3619.88	380.38	597.74	6512.32	684.32	1075.36	788.06	
1	4953.90	4509.06	1172.76	775.10	4917.15	1278.90	845.25	1164.06	
2	3705.63	2963.61	894.00	478.29	5496.27	1658.00	887.03	1326.00	
3	7504.42	6665.89	1304.38	1024.87	6972.82	1364.44	1072.06	1211.98	
4	7668.31	7368.54	1759.94	1208.75	6042.66	1443.26	991.25	1386.84	

	HS	PS
0	1238.38	130.13
1	769.35	200.10
2	709.41	214.00
3	952.27	186.34
4	952.50	227.50

[5 rows x 21 columns]

[10]:
```
df.columns
```

[10]:
```
Index(['MR', 'M', 'W', 'H', 'P', 'S', 'M2', 'W2', 'H2', 'P2', 'S2', 'MW', 'MH',
       'MP', 'MS', 'WH', 'WP', 'WS', 'HP', 'HS', 'PS'],
      dtype='object')
```

[11]:
```
# Variable de respuesta
y = np.array(df['MR'])
# Variables predictoras
x = np.array(df[['M', 'W', 'H', 'P', 'S', 'M2', 'W2', 'H2', 'P2', 'S2', 'MW',
                 'MH','MP', 'MS', 'WH', 'WP', 'WS', 'HP', 'HS', 'PS']])
```

[12]:
```
# Train classifier with all the available observations
clf = MLPRegressor(hidden_layer_sizes=(10, 10), max_iter=10000, random_state=0)
clf.fit(X_scaled, y)

# 5-fold cross-validation
kf = KFold(n_splits=5, shuffle=True)
cv_mse = []
cv_mae = []
for train_index, test_index in kf.split(X_scaled):
    x_train, x_test = X_scaled[train_index], X_scaled[test_index]
```

```
    y_train, y_test = y[train_index], y[test_index]

    regressor_cv = MLPRegressor(hidden_layer_sizes=(10, 10), max_iter=10000,␣
 ↪random_state=0)
    regressor_cv.fit(x_train, y_train)

    y_pred = regressor_cv.predict(x_test)
    mse = mean_squared_error(y_test, y_pred)
    cv_mse.append(mse)
    mae = mean_absolute_error(y_test, y_pred)
    cv_mae.append(mae)

# Calcular el error cuadrático medio promedio
mse = np.mean(cv_mse)
mae = np.mean(cv_mae)
print("MSE:", mse, "\nMAE:", mae)
```

```
MSE: 74.39945882586564
MAE: 4.186758207942139
```

Paso 3 Viendo los resultados de regresión, desarrolla una conclusión sobre los siguientes puntos: * ¿Consideras que el modelo perceptrón multicapa es efectivo para modelar los datos del problema? ¿Por qué? * ¿Qué modelo es mejor para los datos de criminalidad, el lineal o el perceptrón multicapa? ¿Por qué?

Considerando el resultado que se obtuvo en ambos métodos es importante destacar que aunque los errores absolutos tienen una diferencia poco significativa. El error cuadrático medio es considerablemente mejor en el modelo que NO incluye los valores cuadráticos y los productos entre pares de variables. Sin embargo, considero que el modelo lineal es mejor para modelar estos datos por el comportimento y los valores de la variable de respuesta. Pues la naturaleza del problema es de regresión y no de clasificación.

0.1.2 Ejercicio 2

En este ejercicio trabajarás con datos que vienen de un experimento en el que se midió actividad muscular con la técnica de la Electromiografía en el brazo derecho de varios participantes cuando éstos realizaban un movimiento con la mano entre siete posible (Flexionar hacia arriba, Flexionar hacia abajo, Cerrar la mano, Estirar la mano, Abrir la mano, Coger un objeto, No moverse). Al igual que en el ejercicio anterior, los datos se cargan con la función loadtxt de numpy. A su vez * La primera columna corresponde a la clase (1, 2, 3, 4, 5, 6, y 7), * La segunda columna se ignora, y el resto de las columnas indican las variables que se calcularon de la respuesta muscular. * El archivo de datos con el que trabajarás depende de tu matrícula.

Para este conjunto de datos:

[13]:
```
df2 = np.loadtxt('/content/sample_data/M_1 (1).txt')
df2
```

[13]:
```
array([[ 1.        ,  1.        ,  0.6819565 , …,  1.69262835,
         1.34553809,  1.81638713],
       [ 1.        ,  1.        ,  0.56855303, …,  0.64268369,
         0.38791499,  1.59719973],
       [ 1.        ,  1.        ,  1.43149784, …,  0.53153428,
         1.10834576,  2.14520145],
       …,
       [ 7.        ,  1.        , -4.63831072, …, -1.9786276 ,
        -4.04741071, -5.17131175],
       [ 7.        ,  1.        , -5.2325368 , …, -1.31486405,
        -4.31667728, -4.56499901],
       [ 7.        ,  1.        , -4.95990009, …, -1.47060583,
        -4.8555384 , -5.13386256]])
```

[14]:
```
# Variables predictoras
x = df2[:,2:]
# Variable de respuesta
y = df2[:,0]
```

Paso 1 Evalúa un modelo perceptrón multicapa con validación cruzada utilizando al menos 5 capas de 20 neuronas.

[15]:
```
n_features = len(x)
n_features

clf = MLPClassifier(hidden_layer_sizes=(20,20,20,20,20), max_iter=10000)
clf.fit(x,y)

y_pred = cross_val_predict(MLPClassifier(hidden_layer_sizes=(20,20,20,20,20),␣
  ↪max_iter=10000), x, y)
print(classification_report(y, y_pred))
```

```
              precision    recall  f1-score   support

         1.0       0.99      0.97      0.98        90
         2.0       0.90      0.93      0.92        90
         3.0       0.98      0.93      0.95        90
         4.0       0.98      0.99      0.98        90
         5.0       0.92      0.94      0.93        90
         6.0       0.89      0.92      0.91        90
         7.0       0.99      0.96      0.97        90

    accuracy                           0.95       630
   macro avg       0.95      0.95      0.95       630
weighted avg       0.95      0.95      0.95       630
```

Paso 2 Evalúa un modelo perceptrón multicapa con validación cruzada, pero encontrando el número óptimo de capas y neuronas de la red.

[16]:
```
# Optimal number of layers and neurons
num_layers = np.arange(1, 20, 5)
num_neurons = np.arange(10, 110, 20)

layers = []

for l in num_layers:
    for n in num_neurons:
        layers.append(l*[n])

clf = GridSearchCV(MLPClassifier(max_iter=10000), {'hidden_layer_sizes':␣
 ↪layers}, cv = 5)
clf.fit(x, y)
print(clf.best_estimator_)
```

```
MLPClassifier(hidden_layer_sizes=[70], max_iter=10000)
```

Paso 3 Prepara el modelo perceptrón multicapa: * Opten los hiperparámetros óptimos de capas y neuronas de la red. * Con los hiperparámetros óptimos, ajusta el modelo con todos los datos.

[18]:
```
clf = GridSearchCV(MLPClassifier(max_iter=10000), {'hidden_layer_sizes':␣
 ↪layers}, cv = 5)
y_pred = cross_val_predict(clf, x, y, cv = 5)
print(classification_report(y, y_pred))
```

```
              precision    recall  f1-score   support

         1.0       0.99      0.99      0.99        90
         2.0       0.92      0.98      0.95        90
         3.0       0.97      0.94      0.96        90
         4.0       1.00      0.99      0.99        90
         5.0       0.98      0.98      0.98        90
         6.0       0.93      0.90      0.92        90
         7.0       0.99      0.99      0.99        90

    accuracy                           0.97       630
   macro avg       0.97      0.97      0.97       630
weighted avg       0.97      0.97      0.97       630
```

Paso 4 Contesta lo siguientes: * ¿Observas alguna mejora importante al optimizar el tamaño de la red? ¿Es el resultado que esperabas? Argumenta tu respuesta. * ¿Qué inconvenientes hay al encontrar el tamaño óptimo de la red? ¿Por qué?

La optimización del tamaño con la búsqueda exhaustiva de hiperparámetros mejoró el rendimiento

del modelo de clasificación. Los resultados aumentaron en la precisión, recall y F1-score en la mayoría de las clases. Y la precisión general aumentó del 95% al 97%, lo que nuestra usar hiperparámetros para encontrar el número de capas ocultas y neuronas fue muy buena. Sin embargo, el optimizar el tamaño de la red puede aumentar la complejidad del problema y esto depende también de la cantidad de datos. Además, este proceso puede ser computacionalmente costoso y aumenta el riesgo de sobreajuste si se selecciona un modelo muy grande.