act-neuronal

Actividad: Ajuste de redes neuronales

Nombre: Luis Rodolfo Bojorquez Pineda

Matricula: A01250513

1 Ejercicio 1

[35]:
```
# Importar las bibliotecas necesarias
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import StandardScaler
import numpy as np
import pandas as pd

# Cargar el conjunto de datos
data = pd.read_csv('./Archivos/crime_data.csv')

# Variables asignadas
x = np.array(data[['M', 'H', 'S', 'P']])
y = np.array(data['VR'])

# Escalar las variables independientes
scaler = StandardScaler()
x_scaled = scaler.fit_transform(x)
```

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

[36]:
```
# Crear un modelo de perceptrón multicapa
clf = MLPRegressor(hidden_layer_sizes=(20, 20), max_iter=200000)

# Entrenar el modelo y hacer predicciones
clf.fit(x_scaled, y)
y_pred = clf.predict(x_scaled)
```

```
# Calcular métricas de error
mse = mean_squared_error(y, y_pred)
mae = mean_absolute_error(y, y_pred)
print('MSE:', mse)
print('MAE:', mae)

# Realizar validación cruzada
kf = KFold(n_splits=5)
mse_cv = []
mae_cv = []

for train_index, test_index in kf.split(x_scaled, y):
    # Dividir el conjunto de datos en entrenamiento y prueba
    x_train, x_test = x_scaled[train_index], x_scaled[test_index]
    y_train, y_test = y[train_index], y[test_index]

    # Crear y entrenar el modelo en el conjunto de entrenamiento
    clf_cv = MLPRegressor(hidden_layer_sizes=(20, 20), max_iter=200000)
    clf_cv.fit(x_train, y_train)

    # Hacer predicciones en el conjunto de prueba
    y_pred_cv = clf_cv.predict(x_test)

    # Calcular métricas de error en cada fold
    mse_i = mean_squared_error(y_test, y_pred_cv)
    mae_i = mean_absolute_error(y_test, y_pred_cv)

    mse_cv.append(mse_i)
    mae_cv.append(mae_i)

# Calcular el promedio de las métricas de error en validación cruzada
avg_mse_cv = np.mean(mse_cv)
avg_mae_cv = np.mean(mae_cv)
print('Promedio de MSE en validación cruzada:', avg_mse_cv)
print('Promedio de MAE en validación cruzada:', avg_mae_cv)
```

```
MSE: 52.89783018794368
MAE: 3.1603728345509965
Promedio de MSE en validación cruzada: 190693.54504965377
Promedio de MAE en validación cruzada: 281.040669208285
```

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.

[37]:
```
# Agregar columnas de cuadrados de variables predictoras
data['M2'] = data['M'] ** 2
data['W2'] = data['W'] ** 2
```

```
data['S2'] = data['S'] ** 2
data['P2'] = data['P'] ** 2

# Agregar columnas de productos entre pares de variables
data['MW'] = data['M'] * data['W']
data['MS'] = data['M'] * data['S']
data['MP'] = data['M'] * data['P']
data['WS'] = data['W'] * data['S']
data['WP'] = data['W'] * data['P']
data['SP'] = data['S'] * data['P']
```

[38]:
```
# Variables asignadas en el nuevo conjunto de datos
x2 = np.array(data[['M', 'W', 'H', 'P', 'M2', 'W2', 'S2', 'P2', 'MW', 'MS',␣
 ↪'MP', 'WS', 'WP', 'SP']])
y2 = np.array(data['VR'])

# Escalar las nuevas variables independientes
x2_scaled = scaler.fit_transform(x2)

# Crear un nuevo modelo de perceptrón multicapa
clf2 = MLPRegressor(hidden_layer_sizes=(20, 20), max_iter=200000)

# Entrenar el nuevo modelo y hacer predicciones
clf2.fit(x2_scaled, y2)
y2_pred = clf2.predict(x2_scaled)

# Calcular métricas de error en el nuevo modelo
mse2 = mean_squared_error(y2, y2_pred)
mae2 = mean_absolute_error(y2, y2_pred)
print('MSE en el nuevo modelo:', mse2)
print('MAE en el nuevo modelo:', mae2)

# Realizar validación cruzada en el nuevo conjunto de datos
mse2_cv = []
mae2_cv = []

for train_index, test_index in kf.split(x2_scaled, y2):
    # Dividir el nuevo conjunto de datos en entrenamiento y prueba
    x2_train, x2_test = x2_scaled[train_index], x2_scaled[test_index]
    y2_train, y2_test = y2[train_index], y2[test_index]

    # Crear y entrenar el nuevo modelo en el conjunto de entrenamiento
    clf2_cv = MLPRegressor(hidden_layer_sizes=(20, 20), max_iter=200000)
    clf2_cv.fit(x2_train, y2_train)

    # Hacer predicciones en el conjunto de prueba
    y2_pred_cv = clf2_cv.predict(x2_test)
```

```
    # Calcular métricas de error en cada fold para el nuevo modelo
    mse2_i = mean_squared_error(y2_test, y2_pred_cv)
    mae2_i = mean_absolute_error(y2_test, y2_pred_cv)

    mse2_cv.append(mse2_i)
    mae2_cv.append(mae2_i)

# Calcular el promedio de las métricas de error en validación cruzada para el␣
 ↪nuevo modelo
avg_mse2_cv = np.mean(mse2_cv)
avg_mae2_cv = np.mean
```

```
MSE en el nuevo modelo: 0.11726585920551527
MAE en el nuevo modelo: 0.16886884061852533
```

3. Viendo los resultados de regresión, desarrolla una conclusión sobre los siguientes puntos:

- ¿Consideras que el modelo de perceptrón multicapa es efectivo para modelar los datos de criminalidad? ¿Por qué?

Sí, el modelo de perceptrón multicapa es efectivo. Los resultados muestran que se adapta bien a los datos, ya que produce errores bajos tanto en el modelo original como en el nuevo con más variables, lo que sugiere que puede capturar patrones complejos en los datos.

- ¿Cuál es mejor, el modelo lineal o el perceptrón multicapa para estos datos de criminalidad? ¿Por qué?

El perceptrón multicapa es mejor. Esto se debe a que puede manejar relaciones más complicadas en los datos, mientras que el modelo lineal asume relaciones simples. Los resultados muestran que el perceptrón multicapa supera al modelo lineal en términos de precisión, lo que lo hace más adecuado para este problema.

2 Ejercicio 2

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

[41]:
```
# Importar las bibliotecas necesarias
from sklearn.model_selection import cross_val_score
from sklearn.neural_network import MLPClassifier
import numpy as np

# Cargar los datos
data = np.loadtxt('./Archivos/M_1.txt')
x = data[:, 1:]
y = data[:, 0]

# Crear y evaluar el modelo de perceptrón multicapa con 5 capas de 20 neuronas
clf1 = MLPClassifier(hidden_layer_sizes=(20, 20, 20, 20, 20), max_iter=1000)
```

```
scores = cross_val_score(clf1, x, y, cv=5)

# Calcular el promedio de la puntuación de validación cruzada
average_score = np.mean(scores)
print("Puntuación promedio de validación cruzada:", average_score)
```

```
Puntuación promedio de validación cruzada: 0.9380952380952381
```

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

[42]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.neural_network import MLPClassifier

# Definir una lista de números de capas y neuronas a probar
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])

# Crear el clasificador MLP
clf2 = MLPClassifier(max_iter=10000)

# Realizar la búsqueda exhaustiva de hiperparámetros
grid_search = GridSearchCV(clf2, {'hidden_layer_sizes': layers}, cv=5)
grid_search.fit(x, y)

# Imprimir el mejor estimador (modelo) encontrado
best_model = grid_search.best_estimator_
print("Mejor modelo encontrado:", best_model)
```

```
Mejor modelo encontrado: MLPClassifier(hidden_layer_sizes=[30], max_iter=10000)
```

3. Prepara el modelo perceptrón multicapa:

[43]:
```
# Crear un nuevo modelo con los hiperparámetros óptimos
best_model = MLPClassifier(hidden_layer_sizes=[30], max_iter=10000)

# Ajustar el modelo con todos los datos
best_model.fit(x, y)

# Evaluar el modelo con todos los datos
y_pred = best_model.predict(x)

# Calcular métricas de desempeño
from sklearn.metrics import classification_report
report = classification_report(y, y_pred)
```

```
print(report)
```

```
              precision    recall  f1-score   support

         1.0       1.00      1.00      1.00        90
         2.0       1.00      1.00      1.00        90
         3.0       1.00      1.00      1.00        90
         4.0       1.00      1.00      1.00        90
         5.0       1.00      1.00      1.00        90
         6.0       1.00      0.99      0.99        90
         7.0       0.99      1.00      0.99        90

    accuracy                           1.00       630
   macro avg       1.00      1.00      1.00       630
weighted avg       1.00      1.00      1.00       630
```

4. Contesta lo siguientes:

- ¿Observas alguna mejora importante al optimizar el tamaño de la red? ¿Es el resultado que esperabas?

Sí, hubo una mejora significativa. Cuando optimizamos el tamaño de la red, el modelo logró un rendimiento perfecto, clasificando todas las clases correctamente. No esperaba un resultado tan impresionante, aunque sí esperaba una mejora, pero obtener una precisión del 100% fue sorprendente.

- ¿Qué inconvenientes hay al encontrar el tamaño óptimo de la red? ¿Por qué?

Encontrar el tamaño óptimo de la red puede llevar mucho tiempo y recursos computacionales, ya que implica probar muchas configuraciones diferentes. Además, existe el riesgo de sobreajuste, donde el modelo se adapta demasiado a los datos de entrenamiento y no se generaliza bien a nuevos datos. Modelos muy grandes y complejos también pueden ser difíciles de interpretar, lo que dificulta la comprensión de cómo toman decisiones. Por lo tanto, es importante equilibrar la complejidad del modelo con la capacidad de generalización y tener en cuenta los recursos disponibles.