Actividad: Ajuste de redes neuronales

Tania Sayuri Guizado Hernández

Ejercicio 1 (50 puntos)

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

Para este conjunto de datos:

In [1]:
```
#Cargamos las librerías que se usaran en la actividad
from sklearn.model_selection import StratifiedKFold, cross_val_predic
t, GridSearchCV, KFold
from sklearn.metrics import accuracy_score, classification_report,mean
_squared_error, mean_absolute_error
from sklearn.neural_network import MLPRegressor, MLPClassifier
from sklearn.preprocessing import StandardScaler
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv('crime_data.csv')
```

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

In [2]:
```
x = np.array(df[['M', 'W', 'S', 'P']])
y = np.array(df['VR'])
n_features = 4
```

In [3]:
```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(x)
```

In [4]:
```
clf = MLPRegressor(hidden_layer_sizes=(20, 20), max_iter=200000)
clf.fit(X_scaled, y)
y_pred = clf.predict(X_scaled)
print('MSE: ', mean_squared_error(y, y_pred))
print("MAE: ", mean_absolute_error(y, y_pred))
kf = KFold(n_splits=5)
mse_cv = []
mae_cv = []
for train_index, test_index in kf.split(x, y):
    # Training phase
    x_train = X_scaled[train_index, :]
    y_train = y[train_index]
    clf_cv = MLPRegressor(hidden_layer_sizes=(20, 20), max_iter=20000)
    clf_cv.fit(x_train, y_train)
    # Test phase
    x_test = X_scaled[test_index, :]
    y_test = y[test_index]
    y_pred = clf_cv.predict(x_test)
    mse_i = mean_squared_error(y_test, y_pred)
    mse_cv.append(mse_i)
    mae_i = mean_absolute_error(y_test, y_pred)
    mae_cv.append(mae_i)

print('MSE:', np.average(mse_cv), '  MAE:', np.average(mae_cv))
```

```
MSE:  0.7265824647595412
MAE:  0.5077574500355346
MSE: 80420.05778311426   MAE: 211.065184562885
```

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.

In [5]:
```
df['M2'] = df['M'] ** 2
df['W2'] = df['W'] ** 2
df['S2'] = df['S'] ** 2
df['P2'] = df['P'] ** 2
df['MW'] = df['M'] * df['W']
df['MS'] = df['M'] * df['S']
df['MP'] = df['M'] * df['P']
df['WS'] = df['W'] * df['S']
df['WP'] = df['W'] * df['P']
df['SP'] = df['S'] * df['P']
```

In [6]:
```
x2 = np.array(df[['M', 'W', 'S', 'P', 'M2', 'W2', 'S2','P2','MW','MS
','MP','WS','WP','SP']])
y2 = np.array(df['VR'])
n_features = 14
```

In [7]:
```
scaler = StandardScaler()
X2_scaled = scaler.fit_transform(x2)
```

In [8]:
```
clf = MLPRegressor(hidden_layer_sizes=(20, 20), max_iter=20000)
clf.fit(X2_scaled, y)
y_pred = clf.predict(X2_scaled)
print('MSE: ', mean_squared_error(y, y_pred))
print("MAE: ", mean_absolute_error(y, y_pred))
kf = KFold(n_splits=5)
mse_cv = []
mae_cv = []
for train_index, test_index in kf.split(x, y):
    # Training phase
    x_train = X2_scaled[train_index, :]
    y_train = y[train_index]
    clf_cv = MLPRegressor(hidden_layer_sizes=(20, 20), max_iter=20000)
    clf_cv.fit(x_train, y_train)
    # Test phase
    x_test = X2_scaled[test_index, :]
    y_test = y[test_index]
    y_pred = clf_cv.predict(x_test)
    mse_i = mean_squared_error(y_test, y_pred)
    mse_cv.append(mse_i)
    mae_i = mean_absolute_error(y_test, y_pred)
    mae_cv.append(mae_i)

print('MSE:', np.average(mse_cv), '  MAE:', np.average(mae_cv))
```

```
MSE:  0.615828330096692
MAE:  0.3314057222664132
MSE: 174794.3032661454   MAE: 245.06165990274872
```

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?

En base a los resultados obtenidos, puedo concluir que el modelo perceptrón multicapa tiene más efectividad en el modelado de datos de ambos problemas (con las variables originales y con los cuadrados de las variables predictoras). Esto lo veo desde el momento que las métricas de error (MSE y MAE) tienen una gran mejora con los cuadraros de las variables predictoras y se debe porque el modelo perceptrón tiene una capacidad de aprender y explotar relaciones no lineales en los datos. Además, gracias a que puede estudiar patrones complejos en los datos de entrenamiento puede ser más efectivo para predecir datos nuevos o desconocidos.

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

En comparación a los datos obtenidos del modelo lineal de la actividad de **Problemas de regresión** y los de perceptrón multicapa, este último es el que resulta mejor. Porque el modelo perceptrón multicapa es más flexible y puede adaptarse a la complejidad inherente de los datos de criminalidad, que pueden estar influenciados por múltiples factores interconectados de manera no lineal y resulta más preferible debido a su capacidad para modelar relaciones complejas y no lineales como había mencionado en la anterior respuesta.

Ejercicio 2 (50 puntos)

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:

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

In [9]:
```
df2 = np.loadtxt('M_5.txt')
x = df2[:,1:]
y = df2[:,0]
```

In [10]:
```
clf = MLPClassifier(hidden_layer_sizes=(20, 20, 20, 20, 20), max_iter=
1000)
clf.fit(x, y)
y_pred = cross_val_predict(MLPClassifier(hidden_layer_sizes=(20, 20, 2
0, 20, 20),
max_iter=10000), x, y)
print(classification_report(y, y_pred))
```

```
              precision    recall  f1-score   support

         1.0       0.89      0.92      0.91        90
         2.0       0.65      0.68      0.66        90
         3.0       0.96      0.97      0.96        90
         4.0       0.97      0.98      0.97        90
         5.0       0.93      0.90      0.92        90
         6.0       0.66      0.63      0.65        90
         7.0       0.95      0.93      0.94        89

    accuracy                           0.86       629
   macro avg       0.86      0.86      0.86       629
weighted avg       0.86      0.86      0.86       629
```

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

In [11]:
```
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=[90], max_iter=10000)
```

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.

In [12]:
```
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       0.90      0.92      0.91        90
         2.0       0.64      0.74      0.69        90
         3.0       0.96      0.96      0.96        90
         4.0       0.96      0.97      0.96        90
         5.0       0.92      0.90      0.91        90
         6.0       0.78      0.62      0.69        90
         7.0       0.97      1.00      0.98        89

    accuracy                           0.87       629
   macro avg       0.87      0.87      0.87       629
weighted avg       0.87      0.87      0.87       629
```

4-. Contesta lo siguientes:

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

Si, se mostro una mejora importante porque cuando utilizabamos 5 capas de 20 neuronas el modelo tenia un accuracy promedio del 84% pero, cuando realice la busqueda para encontrar el número óptimo de capas y neuronas el modelo alcanzo un accuracy promedio del 87%. Quiza no sea el gran porcentaje, no obstante, si represento una mejora en el rendimiento del modelo. En parte si era el resultado de lo que esperaba porque es común que la optimización de la arquitectura de la red neuronal mejore el rendimiento, sin embargo, esperaba que el resultado de la precisión alcanzara un 90% al menos. Igual comprendo que el resultado siempre depende de la complejidad y naturaleza de los datos asi que los resultados obtenidos fueron decentes.

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

Entre los principales inconvenientes que considero que hay al encontrar el tamaño óptimo de la red son el costo computacional, el riesgo de sobreajuste y la dificultad para la interpretación. De hecho, cuando corrí el código del paso 3 si tardo alrededor de 20 minutos en ejecutar y esto se debió a que como tal la búsqueda de hiperparámetros resulto computacionalmente costosa y aunque ayuda no siempre es práctico. Acorde a lo del riesgo de sobreajuste, cuando una red demasiado grande se adapta en exceso a los datos de entrenamiento puede obtener un alto rendimiento en ellos pero un bajo rendimiento en datos no vistos. Además, redes neuronales grandes y complejas pueden ser difíciles de interpretar, lo que dificulta la comprensión de cómo toman decisiones.

Desde mi punto de vista encontrar el tamaño óptimo siempre sera desafiante porque implica equilibrar la complejidad del modelo con la capacidad de generalización, recursos computacionales y la interpretabilidad del modelo.