Actividad: Ajuste de redes neuronales

Joel Isaias Solano Ocampo | A01639289

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

In [183…
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier, MLPRegressor
from sklearn.model_selection import StratifiedKFold, cross_val_predict, GridSearchCV,
from sklearn.metrics import accuracy_score, classification_report,  mean_squared_error
```

In [170…
```
df = pd.read_csv('crime_data.csv')
df[:5]
```

Out[170]:

	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

In [171…
```
df = df.drop(['State', 'VR'], axis=1)
df.head()
```

Out[171]:

	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

In [173…
```
x = np.array(df[['M', 'W', 'H', 'P', 'S']])
y = np.array(df['MR'])
n_features = 5
```

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

In [174…
```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(x)
```

In [175…
```
# 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)
```

Out[175]:
```
MLPRegressor(hidden_layer_sizes=(10, 10), max_iter=10000, random_state=0)
```

In [177…
```
# K-fold cross-validation
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=200000)
  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)
  mae_i = mean_absolute_error(y_test, y_pred)
  mse_cv.append(mse_i)
  mae_cv.append(mae_i)
print('MSE CV: ', np.average(mse_cv))
print('MAE CV: ', np.average(mae_cv))
```

```
MSE CV:  29.19029279637747
MAE CV:  2.7376483154126747
```

1. 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 [178…
```
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']
```

In [179…
```
x2 = np.array(df[['M', 'W', 'H', 'P', 'S', 'M2', 'W2', 'H2', 'P2', 'S2', 'MW', 'MH',
y2 = np.array(df['MR'])
#y2 = y2.astype(int)
n_features = 20
```

In [180…
```
scaler = StandardScaler()
X2_scaled = scaler.fit_transform(x2)
```

In [181…
```
# Train classifier with all the available observations
clf = MLPRegressor(hidden_layer_sizes=(20, 20), max_iter=200000)
clf.fit(X2_scaled, y)
```

Out[181]:
```
MLPRegressor
MLPRegressor(hidden_layer_sizes=(20, 20), max_iter=200000)
```

In [182…
```
# K-fold cross-validation
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=200000)
  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)
  mae_i = mean_absolute_error(y_test, y_pred)
  mae_cv.append(mae_i)
```

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

```
MSE CV:  32.140640989582096
MAE CV:  3.1198891360626235
```

1. 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?** El modelo perceptron multicapa es efectivo para modelar los datos de este problema, los resultados de MSE y MAE son muy parecidos a los resultados obtenidos en la actividad de *Problemas de regresion*, donde aplicamos la validacion cruzada de una manera muy similar y ambos resultados tanto en MSE y MAE son muy parecidos, por lo que podemos concluir que el modelo es efectivo para modelar los datos del problema.
- **¿Qué modelo es mejor para los datos de criminalidad, el lineal o el perceptrón multicapa? ¿Por qué?** Justamente reiterando en la actividad anterior *Problemas de regresion*, el modelo lineal tuvo un R^2 de: -1.20. Esto quiere decir que el modelo lineal no es bueno para modelar los datos, por lo que el modelo perceptron multicapa es mejor para los datos de criminalidad, ya que el R^2 no tiene sentido en este modelo y se comporta de una forma similar al modelo cuadratico, tambien aplicado en la actividad anterior.

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.

In [164…
```
df = np.loadtxt('M_5.txt')
df[:5]
```

Out[164]:
```
array([[ 1.        ,  1.        ,  0.15991005, ...,  0.87855426,
         1.63603899,  1.60896884],
       [ 1.        ,  1.        , -1.03964581, ...,  0.67610431,
         0.75054922,  1.04006595],
       [ 1.        ,  1.        , -1.41164407, ...,  0.36203866,
         1.59177889,  1.53300746],
       [ 1.        ,  1.        , -2.6459736 , ...,  2.5321943 ,
         0.34460084, -0.867472  ],
       [ 1.        ,  1.        , -1.69286003, ...,  1.38472232,
         0.45231775,  0.7562994 ]])
```

In [165…
```
x = df[:,1:]
y = df[:,0]
```

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

In [166…
```
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, 20, 20, 20),
max_iter=10000), x, y)
print(classification_report(y, y_pred))
```

```
              precision    recall  f1-score   support

         1.0       0.94      0.87      0.90        90
         2.0       0.60      0.66      0.62        90
         3.0       0.97      0.97      0.97        90
         4.0       0.95      0.93      0.94        90
         5.0       0.88      0.89      0.88        90
         6.0       0.68      0.67      0.67        90
         7.0       0.98      0.99      0.98        89

    accuracy                           0.85       629
   macro avg       0.86      0.85      0.85       629
weighted avg       0.86      0.85      0.85       629
```

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

In [167…
```
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)
```

1. 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 [168…
```
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	0.92	0.89	0.90	90
2.0	0.65	0.61	0.63	90
3.0	0.97	0.96	0.96	90
4.0	0.96	0.97	0.96	90
5.0	0.91	0.94	0.93	90
6.0	0.65	0.68	0.66	90
7.0	0.96	0.97	0.96	89
accuracy			0.86	629
macro avg	0.86	0.86	0.86	629
weighted avg	0.86	0.86	0.86	629

1. Contesta lo siguientes:

- **¿Observas alguna mejora importante al optimizar el tamaño de la red? ¿Es el resultado que esperabas? Argumenta tu respuesta.** Podria argumentar que si existe una mejora al optimizar el tamaño de la red pero definitivamente no podria argumentar que es una mejora importante, mas bien una mejora insignificativa. El modelo con 5 capas de 20 neuronas obtuvo un accuracy en F1 de 0.85 mientras que el modelo con el tamaño optimo de neuronas y capas obtuvo un accuracy en F1 de 0.86. Y si vemos los mismos valores de F1 score en las diferentes clases, podemos ver que son muy similares, por lo que no existe una mejora importante al optimizar el tamaño de la red.
- **¿Qué inconvenientes hay al encontrar el tamaño óptimo de la red? ¿Por qué?** Curiosamente, haber encontrado el tamaño optimo de neuronas y capas no mejoro considerablemente el modelo, ya que el modelo con 5 capas de 20 neuronas obtuvo un accuracy en F1 de 0.85 mientras que el modelo con el tamaño optimo de neuronas y capas obtuvo un accuracy en F1 de 0.86. A pesar de grandes similitudes en las diferentes clases en cuanto al valor de F1 score, el haber encontrado el tamaño optimo hizo que el modelo tardara aproximadamente 7 minutos en correr mientras que el otro solo duro 5.5 segundos; una diferencia muy significativa.