Actividad - Ajuste de redes neuronales

September 10, 2023

1 Actividad: Ajuste de redes neuronales

Alfonso Pineda Cedillo | A01660394

Fecha de entrega: 8 de Septiembre de 2023

1.1 Ejercicio 1

Instrucciones:

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 100,000 habitantes
- `MR`: asesinatos por cada 100,000 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 con ingresos por debajo del nivel de pobreza

Para este conjunto de datos:

1. Evalúa con validación cruzada un modelo pereceptrón multicapa para las variables que se te asignaron para este ejercicio.
2. Agrega al conjunto de datos columnas que representen los cuadrados de las variables predictoras (por ejemplo, M^2, W^2), así como los productos entre pares de variables (por ejemplo, $P \times S$, $M \times W$). Evalúa un modelo perceptrón multicapa para este nuevo conjunto de datos.
3. Viendo los resultados de regresión, desarrolla una conclusión sobre los siguientes puntos:

   A. ¿Consideras que el modelo perceptrón multicapa es efectivo para modelar los datos del problema? ¿Por qué?

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

Último número de matrícula: 4

Variables a utilizar: Variable dependiente `MR`, variables independientes: `M`, `W`, `H`, y `P`.

1.1.1 Solución Ejercicio 1

Importamos las librerías necesarias para el ejercicio.

[38]:
```
import pandas as pd
from sklearn.neural_network import MLPRegressor
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
from sklearn.model_selection import cross_val_predict, GridSearchCV
import numpy as np
```

Importamos el Dataset a utilizar.

[2]:
```
data = pd.read_csv('crime_data.csv')
```

Definimos la variable de respuesta y las variables predictoras correspondientes.

[13]:
```
X = data[['M', 'W', 'H', 'P']]
y = data['MR']
```

Paso 1: Evaluamos con validación cruzada un modelo pereceptrón multicapa para las variables asignadas para este ejercicio.

Primero creamos un modelo de Perceptrón Multicapa (MLPRegressor) con dos capas ocultas, cada una con 100 neuronas, utilizando la configuración `hidden_layer_sizes=(100, 100)`. Este modelo se entrena con todas las observaciones del conjunto de datos.

[33]:
```
# Entrenamos el modelo con todas las observaciones (dos capas de 100 neuronas)
clf = MLPRegressor(hidden_layer_sizes=(100, 100), max_iter=10000)
clf.fit(X, y)
```

[33]:
```
MLPRegressor(hidden_layer_sizes=(100, 100), max_iter=10000)
```

Posteiromente, se evalúa el modelo con validación cruzada utilizando el método `cross_val_score` de la librería `sklearn.model_selection`. Esto implica predecir los valores de la variable objetivo (`MR`) y calcular métricas de regresión para evaluar el rendimiento.

[35]:
```
# Evaluamos el modelo (dos capas de 100 neuronas)
y_pred = cross_val_predict(MLPRegressor(hidden_layer_sizes=(100, 100),␣
 ↪max_iter=10000), X, y)
```

A continuación, definimos los rangos para el número de capas ocultas (`num_layers`) y el número de neuronas en cada capa (`num_neurons`). Luego, se crea una lista `layers` que contiene todas las combinaciones posibles de capas y neuronas a probar.

Utilizamos `GridSearchCV` para encontrar la mejor configuración de capas y neuronas utilizando validación cruzada con 5 divisiones (`cv=5`). El objetivo es optimizar el rendimiento del modelo encontrando la configuración que minimiza el error de regresión.

Por último, imprimimos el mejor estimador encontrado por `GridSearchCV`, que representa la configuración óptima de capas y neuronas para el modelo de Perceptrón Multicapa.

[36]:
```
# Número óptimo de capas y neuronas
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])

# Usamos GridSearchCV para encontrar la mejor configuración de capas y neuronas
clf = GridSearchCV(MLPRegressor(max_iter=10000), {'hidden_layer_sizes':␣
 ↪layers}, cv=5)
clf.fit(X, y)
print("Mejor estimador encontrado:")
print(clf.best_estimator_)
```

```
Mejor estimador encontrado:
MLPRegressor(hidden_layer_sizes=[70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70,
                                 70, 70, 70, 70],
             max_iter=10000)
```

Utilizamos `cross_val_predict` nuevamente para evaluar el modelo con la configuración óptima encontrada en el paso anterior.

[37]:
```
# Evaluamos el modelo encontrando el número óptimo de capas y neuronas
clf = GridSearchCV(MLPRegressor(max_iter=10000), {'hidden_layer_sizes':␣
 ↪layers}, cv=5)
y_pred = cross_val_predict(clf, X, y, cv=5)
```

Por último, obtenemos las métricas de regresión para evaluar el rendimiento del modelo.

[40]:
```
# Calculamos el Error Cuadrado Medio (MSE)
mse = mean_squared_error(y, y_pred)
print("Error Cuadrado Medio (MSE):", mse)

# Calculamos el Error Absoluto Medio (MAE)
mae = mean_absolute_error(y, y_pred)
print("Error Absoluto Medio (MAE):", mae)

# Calculamos el Coeficiente de Determinación (R^2)
r2 = r2_score(y, y_pred)
print("Coeficiente de Determinación (R^2):", r2)
```

```
Error Cuadrado Medio (MSE): 85.06261472594268
Error Absoluto Medio (MAE): 4.075668724690002
Coeficiente de Determinación (R^2): 0.24465428251132648
```

Paso 2: Agregamos al conjunto de datos columnas que representen los cuadrados de las variables predictoras, así como los productos entre pares de variables.. Evaluamos un modelo perceptrón multicapa para este nuevo conjunto de datos.

[41]:
```
data['M2'] = data['M'] ** 2
data['W2'] = data['W'] ** 2
data['H2'] = data['H'] ** 2
data['P2'] = data['P'] ** 2

data['MxW'] = data['M'] * data['W']
data['MxH'] = data['M'] * data['H']
data['MxP'] = data['M'] * data['P']
data['WxH'] = data['W'] * data['H']
data['WxP'] = data['W'] * data['P']
data['HxP'] = data['H'] * data['P']
```

Definimos las variables predictoras actualizadas y la variable objetivo.

[42]:
```
X_updated = data[['M', 'W', 'H', 'P', 'M2', 'W2', 'H2', 'P2', 'MxW', 'MxH',␣
  ↪'MxP', 'WxH', 'WxP', 'HxP']]
y_updated = data['MR']
```

Creamos un modelo de Perceptrón Multicapa (MLPRegressor) con dos capas ocultas, cada una con 100 neuronas, utilizando la configuración `hidden_layer_sizes=(100, 100)`. Este modelo se entrena con todas las observaciones del conjunto de datos.

[43]:
```
clf = MLPRegressor(hidden_layer_sizes=(100, 100), max_iter=10000)
clf.fit(X_updated, y_updated)
```

[43]:
```
MLPRegressor(hidden_layer_sizes=(100, 100), max_iter=10000)
```

Posteiromente, se evalúa el modelo con validación cruzada utilizando el método `cross_val_score` de la librería `sklearn.model_selection`. Esto implica predecir los valores de la variable objetivo (`MR`) y calcular métricas de regresión para evaluar el rendimiento.

[44]:
```
y_pred_updated = cross_val_predict(MLPRegressor(hidden_layer_sizes=(100, 100),␣
  ↪max_iter=10000), X_updated, y_updated)
```

Por último, obtenemos las métricas de regresión para evaluar el rendimiento del modelo.

[45]:
```
# Calculamos el Error Cuadrado Medio (MSE)
mse_updated = mean_squared_error(y_updated, y_pred_updated)
print("Error Cuadrado Medio (MSE) con variables adicionales:", mse_updated)

# Calculamos el Error Absoluto Medio (MAE)
mae_updated = mean_absolute_error(y_updated, y_pred_updated)
print("Error Absoluto Medio (MAE) con variables adicionales:", mae_updated)

# Calculamos el Coeficiente de Determinación (R^2)
r2_updated = r2_score(y_updated, y_pred_updated)
print("Coeficiente de Determinación (R^2) con variables adicionales:",␣
  ↪r2_updated)
```

```
Error Cuadrado Medio (MSE) con variables adicionales: 72433.3941178236
Error Absoluto Medio (MAE) con variables adicionales: 131.323474661441
Coeficiente de Determinación (R^2) con variables adicionales: -642.1997679161514
```

1.1.2 Conclusiones Ejercicio 1

A. ¿Consideras que el modelo perceptrón multicapa es efectivo para modelar los datos del problema? ¿Por qué?

El modelo de perceptrón multicapa sin variables adicionales tiene un R^2 de 0.245, lo que significa que solo explica el 24.5% de la varianza en los datos. Esto indica que el modelo no es muy efectivo para modelar la relación entre las variables predictoras y la variable objetivo (crímenes violentos por cada 100,000 habitantes). Además, el MAE de 4.08 es relativamente alto, lo que indica que el modelo tiene un error promedio significativo en sus predicciones.

Por otro lado, el modelo de perceptrón multicapa con variables adicionales tiene un rendimiento aún peor, con un MAE y MSE muy altos. Esto sugiere que la inclusión de variables adicionales (cuadrados y productos entre variables) ha empeorado el rendimiento del modelo, lo que indica un posible problema de sobreajuste.

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

Tomando en cuenta que el modelo lineal obtuvo los siguientes resultasdos:

Error Cuadrado Medio (MSE): 37.37
Error Absoluto Medio (MAE): 3.35
Coeficiente de Determinación (R^2): 0.668

Podemos conluir que el modelo lineal es claramente mejor para los datos de criminalidad en este caso. Hay varias razones para esto:

- **Mejor rendimiento en las métricas:** El modelo lineal tiene un R^2 mucho más alto, lo que significa que explica una mayor proporción de la variabilidad en los datos. Además, el MSE y MAE son significativamente más bajos en el modelo lineal, lo que indica que las predicciones del modelo lineal son más precisas.
- **Simplicidad:** El modelo lineal es más simple e interpretable que el perceptrón multicapa. No incluye términos cuadráticos o productos entre variables, lo que hace que sea más fácil entender cómo cada variable predictora afecta a la variable objetivo.
- **Menos riesgo de sobreajuste:** El modelo lineal tiene menos riesgo de sobreajuste y generaliza mejor a datos no vistos.

1.2 Ejercicio 2

Instrucciones:

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

   A. Opten los hiperparámetros óptimos de capas y neuronas de la red.

   B. Con los hiperparámetros óptimos, ajusta el modelo con todos los datos.
4. Contesta lo siguiente:

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

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

Penúltimo número de matrícula: 9

Conjunto de datos a utilizar: `M_5.txt`

1.2.1 Solución Ejercicio 2

Importamos las librerías necesarias para el ejercicio.

[58]:
```
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import classification_report
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import GridSearchCV
import numpy as np
```

Importamos el Dataset a utilizar.

[53]:
```
data = np.loadtxt('M_5.txt')
```

Separamos las características de la clase (`y`) y las etiquetas (`X`).

[54]:
```
X = data[:, 1:]  # Ignorar la segunda columna (se ignora la primera columna también ya que es la clase)
y = data[:, 0]   # La primera columna es la clase
```

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

Para este primer paso, definiremos un modelo de MLP con una estructura fija. Utilizaremos 5 capas ocultas, cada una con 20 neuronas. Esto se especificará utilizando el parámetro `hidden_layer_sizes` del clasificador MLPClassifier.

[55]:
```
clf = MLPClassifier(hidden_layer_sizes=(20, 20, 20, 20, 20), max_iter=10000)
```

A continuación, aplicaremos la validación cruzada para evaluar el rendimiento del modelo en diferentes divisiones de los datos. Utilizaremos la validación cruzada estratificada con 5 pliegues (`StratifiedKFold`) para garantizar que cada clase tenga una representación adecuada en cada pliegue.

Durante cada iteración de la validación cruzada, dividiremos los datos en conjuntos de entrenamiento y prueba. Luego, entrenaremos el modelo de MLP en el conjunto de entrenamiento y lo evaluaremos en el conjunto de prueba. Registraremos las predicciones y las etiquetas reales para cada pliegue.

[57]:
```
# Realizamos validación cruzada
kf = StratifiedKFold(n_splits=5, shuffle=True)
cv_y_test = []
cv_y_pred = []

for train_index, test_index in kf.split(X, y):
    # Fase de entrenamiento
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    clf_cv = MLPClassifier(hidden_layer_sizes=(20, 20, 20, 20, 20),␣
 ↪max_iter=10000)
    clf_cv.fit(X_train, y_train)

    # Fase de prueba
    y_pred = clf_cv.predict(X_test)
    cv_y_test.append(y_test)
    cv_y_pred.append(y_pred)
```

Al finalizar la validación cruzada, generaremos un informe de clasificación que incluirá métricas de evaluación como la precisión, la recuperación y la puntuación F1 para cada clase y un resumen general del rendimiento del modelo.

[59]:
```
# Imprimir informe de clasificación
print(classification_report(np.concatenate(cv_y_test), np.
 ↪concatenate(cv_y_pred)))
```

```
              precision    recall  f1-score   support

         1.0       0.93      0.90      0.92        90
         2.0       0.64      0.68      0.66        90
         3.0       0.98      0.96      0.97        90
         4.0       0.96      0.97      0.96        90
```

```
           5.0       0.90      0.90      0.90        90
           6.0       0.72      0.69      0.70        90
           7.0       0.98      1.00      0.99        89

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

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

Primero, definimos un rango de posibles números de capas (num_layers) y neuronas (num_neurons). En este caso, estamos explorando desde 1 hasta 19 capas en incrementos de 5, y desde 10 hasta 100 neuronas en incrementos de 20.

[60]:
```
num_layers = np.arange(1, 20, 5)
num_neurons = np.arange(10, 110, 20)
layers = []
```

Generamos todas las combinaciones posibles de capas y neuronas utilizando un bucle anidado. Esto nos proporciona una lista de configuraciones de capas ocultas que queremos probar.

[61]:
```
# Generar todas las combinaciones posibles de capas y neuronas
for l in num_layers:
    for n in num_neurons:
        layers.append(l * [n])
```

Creamos un clasificador de perceptrón multicapa con búsqueda de cuadrícula (`GridSearchCV`). Este clasificador se ajustará automáticamente con diferentes configuraciones de capas y neuronas y seleccionará la mejor configuración basada en la validación cruzada.

[62]:
```
clf = GridSearchCV(MLPClassifier(max_iter=10000), {'hidden_layer_sizes':␣
 ↪layers}, cv=5)
```

Entrenamos el clasificador en nuestros datos de entrada (`x`) y etiquetas de salida (`y`) utilizando la función `fit` e imprimimos la mejor configuración encontrada en términos de capas y neuronas ocultas.

[63]:
```
clf.fit(X, y)
print("Número óptimo de capas y neuronas:", clf.best_estimator_.
 ↪hidden_layer_sizes)
```

```
Número óptimo de capas y neuronas: [50]
```

Posteriormente, evaluamos el modelo utilizando la validación cruzada y almacenamos las predicciones en `y_pred`. Finalmente, imprimimos un informe de clasificación que muestra métricas de rendimiento como precisión, recuperación y puntuación F1 para cada clase y un resumen general. Esto nos ayuda a evaluar el rendimiento del modelo con la configuración óptima encontrada.

[64]:
```
# Evaluar el modelo encontrando la configuración óptima de capas y neuronas
y_pred = cross_val_predict(clf, X, y, cv=5)

# Imprimir el informe de clasificación
print(classification_report(y, y_pred))
```

```
              precision    recall  f1-score   support

         1.0       0.94      0.90      0.92        90
         2.0       0.63      0.68      0.65        90
         3.0       0.96      0.97      0.96        90
         4.0       0.97      0.97      0.97        90
         5.0       0.90      0.91      0.91        90
         6.0       0.72      0.68      0.70        90
         7.0       0.98      0.98      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
```

Paso 3: Preparamos el modelo perceptrón multicapa para producción.

[66]:
```
# Obtener los mejores hiperparámetros encontrados
best_hidden_layer_sizes = clf.best_params_['hidden_layer_sizes']

# Crear y entrenar el modelo con todos los datos
clf_final = MLPClassifier(hidden_layer_sizes=best_hidden_layer_sizes)
clf_final.fit(X, y)
```

```
/Users/alfonsopineda/Library/Python/3.9/lib/python/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:691:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
```

[66]:
```
MLPClassifier(hidden_layer_sizes=[50])
```

1.2.2 Conclusiones Ejercicio 2

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

No, no se observa una mejora importante en los resultados al optimizar el tamaño de la red. Ambos modelos tienen un rendimiento muy similar en términos de precisión y F1-score promedio ponderado, con valores de alrededor del 0.87 en ambas métricas. Esto sugiere que, en este caso particular, aumentar el número de capas y neuronas no condujo a una mejora significativa en el rendimiento del modelo.

Esto podría no ser el resultado esperado, ya que normalmente se espera que un modelo más grande (con más capas y neuronas) tenga la capacidad de aprender representaciones más complejas de los datos y, por lo tanto, mejore el rendimiento. Sin embargo, en la práctica, esto puede depender en gran medida de la naturaleza de los datos y la complejidad del problema. En algunos casos, un modelo más pequeño puede ser suficiente para obtener buenos resultados, como parece ser el caso aquí.

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

Encontrar el tamaño óptimo de una red neuronal, es decir, el número adecuado de capas y neuronas, puede ser un desafío debido a varios inconvenientes:

- **Costo computacional:** Probar todas las combinaciones posibles de hiperparámetros (número de capas y neuronas) puede ser computacionalmente costoso y llevar mucho tiempo, especialmente si se realiza mediante validación cruzada.
- **Sobreajuste:** Aumentar el tamaño de la red puede llevar al sobreajuste (overfitting), donde el modelo se adapta demasiado a los datos de entrenamiento y no generaliza bien a nuevos datos. Esto puede empeorar el rendimiento en conjuntos de datos no vistos.
- **Datos:** Un modelo grande requiere más datos para entrenarse de manera efectiva. Si el conjunto de datos es pequeño, un modelo grande puede tener un rendimiento deficiente debido a la falta de ejemplos de entrenamiento.
- **Interpretación:** Un modelo grande puede ser más difícil de interpretar y comprender. Esto puede dificultar la depuración y el ajuste del modelo.