Machine Learning Aplicado : Regresión con keras

1. Resumen: El trabajo consiste en el desarrollo de una regresión por medio de una perceptrón (red neuronal simple) sobre un conjunto de datos housing.csv para estimar el valor de la casa de acuerdo al ingreso medio de una casa.

Librerías: Keras, numpy, pandas, sklearn, matplotlib

Objetivo: Codificar y utilizar el framework de deep learning keras para el modelado de una regresión.

2. Descripción de Datos:

Los datos pertenecen a las casas encontradas en un distrito de California y algunas estadísticas resumidas sobre ellas basadas en los datos del censo de 1990.

3. Desarrollo de Actividad

Para cada uno de los siguientes ítems, adjuntar imagen del código realizado.

1. Importación de librerías

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import keras
from keras.models import Sequential
from keras.layers import Dense
```

2. Lectura de dataframe

```
from google.colab import drive

# Monta Google Drive
drive.mount('/content/drive')

# Especificar la ruta del archivo subido
file_path = '/content/drive/MyDrive/Trabajos/MLaplicado/BostonHousing.csv'

# Leer el archivo CSV
df = pd.read_csv(file_path)
df.head()
```

3. Explique el procesamiento y estandarización de datos

```
# Estandarización de las características numéricas
scaler = StandardScaler()
numeric_cols = df.select_dtypes(include=[float, int]).columns
df[numeric_cols] = scaler.fit_transform(df[numeric_cols])
```

La estandarización es un proceso común en el preprocesamiento de datos para asegurarse de que todas las características tengan la misma escala. En este caso, se utiliza la clase StandardScaler de scikit-learn para centrar las características alrededor de cero y escalarlas para tener una desviación estándar unitaria.

```
# Calcular el rango intercuartil solo para columnas numéricas
numeric_cols = df.select_dtypes(include=[float, int]).columns
Q1 = df[numeric_cols].quantile(0.25)
Q3 = df[numeric_cols].quantile(0.75)
IQR = Q3 - Q1

# Definir un umbral para identificar los valores extremos
threshold = 0.7
lower_threshold = Q1 - threshold * IQR
upper_threshold = Q3 + threshold * IQR

# Eliminar las filas con valores extremos
df = df[~((df[numeric_cols] < lower_threshold) | (df[numeric_cols] > upper_threshold)).any(axis=1)]
```

En este extracto de código se calcula el rango intercuartil (IQR) para las columnas numéricas de un conjunto de datos. El IQR es una medida de dispersión estadística que se utiliza para identificar la variabilidad en los datos. Se calcula como la diferencia entre el tercer cuartil (Q3) y el primer cuartil (Q1) de un conjunto de datos. Luego, se define un umbral multiplicativo (en este caso, 0.7) para identificar valores extremos. Los valores que están por debajo del cuartil Q1 menos el umbral multiplicativo del IQR y por encima del cuartil Q3 más el umbral multiplicativo del IQR se consideran valores extremos. Finalmente, se eliminan las filas que contienen al menos un valor extremo en las columnas numéricas.

4. Explique la división de conjunto de datos para entrenamiento, validación y test

```
# Extraer la característica 'rm' (número promedio de habitaciones por vivienda) y el objetivo 'medv' (valor medio de las viviendas ocupadas por sus propietarios
X = df[['rm']].values
y = df['medv'].values
```

El código se encarga de extraer dos variables de interés de un conjunto de datos. En particular, selecciona la característica 'rm', que representa el número promedio de habitaciones por vivienda, y el objetivo 'medv', que indica el valor medio de las viviendas ocupadas por sus propietarios. Luego, convierte estas variables en matrices numpy para su posterior uso en un modelo de aprendizaje automático.

```
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

X_train = X_train.reshape(-1, 1)
X_val = X_val.reshape(-1, 1)
X_test = X_test.reshape(-1, 1)
```

El código realiza una división de los datos en tres conjuntos: entrenamiento, validación y prueba. En primer lugar, divide el conjunto de características (X) y el objetivo (y) en dos partes, una para entrenamiento y otra combinada para validación y prueba, utilizando la función train_test_split. Luego, toma esa segunda parte y la divide nuevamente en dos partes iguales, una para validación y otra para prueba. Después, modifica la forma de las características para que tengan una sola columna, debido a que el modelo de aprendizaje automático espera una matriz bidimensional de características.

5. Explique cada una de las capas del modelo de regresión por medio de un perceptrón. Crear un modelo secuencial con *keras.models* y capas densas con *keras.layers*

```
model = Sequential()
model.add(Dense(1, input_dim=1, kernel_initializer='uniform', activation='linear'))
model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mean_squared_error'])
```

- *Capa de Entrada (Dense):*
  Esta es la primera capa del modelo, donde se especifica el número de entradas que tiene el modelo. En este caso, input_dim=1 indica que hay una sola característica de entrada.
- *Capa de Salida (Dense):*
  Esta es la capa de salida del modelo, donde se especifica el número de neuronas de salida. Dado que estamos realizando una regresión y queremos predecir un único valor, se utiliza una sola neurona de salida (Dense(1)).
  La función de activación linear se utiliza comúnmente en capas de salida de modelos de regresión, ya que no aplica ninguna transformación a la salida de la capa.

6. Explique el resumen de capas de la red.

```
model.summary()
```

Este es un modelo secuencial de una sola capa densa. La capa densa tiene una salida de forma (None, 1), lo que significa que produce una sola salida para cada entrada. Tiene 2 parámetros entrenables (pesos y sesgos), lo que se refleja en el total de parámetros del modelo, que es 2. No hay parámetros no entrenables en este modelo.

7. Explique cada argumento del método de entrenamiento del modelo perceptron

```
history = model.fit(X_train, y_train, epochs=150, batch_size=32, validation_data=(X_val, y_val), verbose=1)
```

- X_train: Es el conjunto de características de entrenamiento.
- y_train: Son las etiquetas correspondientes a las características de entrenamiento.
- epochs=150: Este argumento especifica el número de épocas, es decir, el número de veces que el modelo pasará por todo el conjunto de datos de entrenamiento durante el entrenamiento. En este caso, se están realizando 150 épocas.
- batch_size=32: Define el tamaño del lote, es decir, cuántos ejemplos de entrenamiento se utilizan antes de que los gradientes se calculen y se apliquen al modelo durante una iteración de entrenamiento. Aquí, se está utilizando un tamaño de lote de 32.
- validation_data=(X_val, y_val): Este argumento especifica el conjunto de datos de validación, que se utilizará para evaluar el rendimiento del modelo después de cada época de entrenamiento. Aquí, X_val son las características de validación y y_val son las etiquetas correspondientes.
- verbose=1: Determina la cantidad de información que se mostrará durante el entrenamiento. Un valor de 1 significa que se mostrarán los detalles del progreso del entrenamiento en cada época.

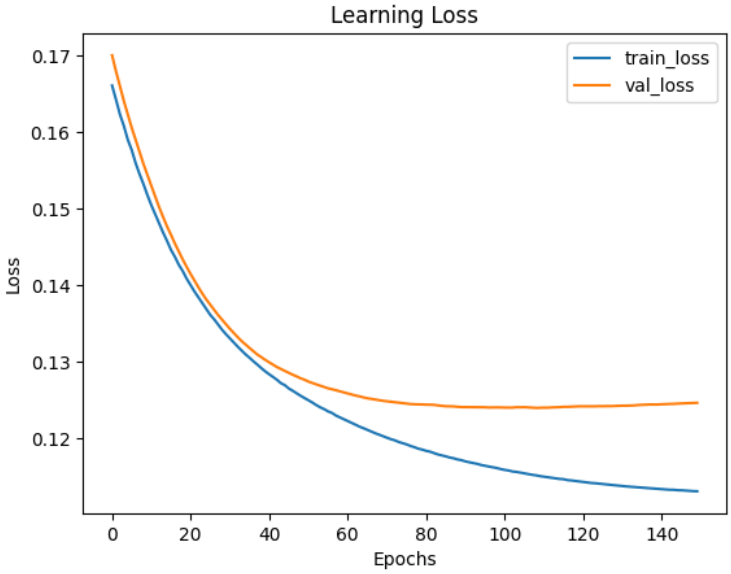
8. Explique los resultados de la evaluación del modelo perceptron

```
test_loss, test_mse = model.evaluate(X_test, y_test)
print(f'Loss: {test_loss}, MSE: {test_mse}')
```

Estos resultados indican que durante la evaluación del modelo, se obtuvo una pérdida (loss) y un error cuadrático medio (MSE) de aproximadamente 0.0621. Esta métrica sugiere que el modelo está realizando bastante bien la tarea para la cual fue entrenado, ya que tanto la pérdida como el MSE son bajos, lo que indica que la diferencia entre las predicciones del modelo y los valores reales es mínima.

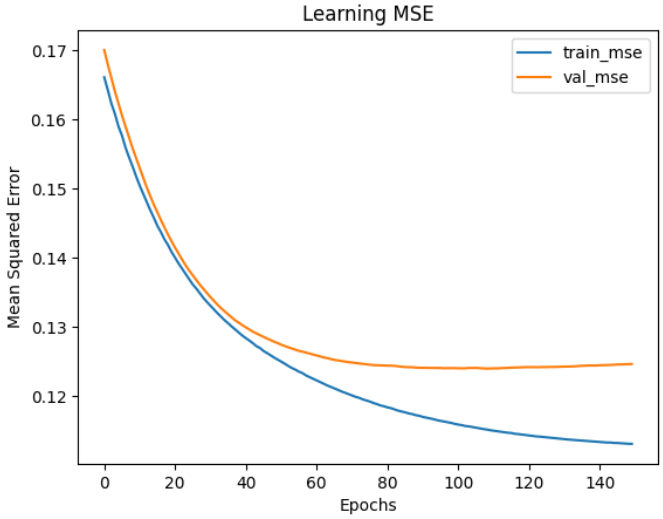
9. Gráfica de pérdida de aprendizaje durante el entrenamiento

```
plt.plot(history.history['loss'], label='train_loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.title('Learning Loss')
plt.show()
```



10. Gráfica del MSE de aprendizaje durante el entrenamiento

```
plt.plot(history.history['mean_squared_error'], label='train_mse')
plt.plot(history.history['val_mean_squared_error'], label='val_mse')
plt.xlabel('Epochs')
plt.ylabel('Mean Squared Error')
plt.legend()
plt.title('Learning MSE')
plt.show()
```



11. Explique el comportamiento de ambos gráficos.

Gráfica de pérdida de aprendizaje: Muestra cómo disminuye la pérdida a medida que avanza el entrenamiento. Idealmente, la pérdida de entrenamiento y validación disminuyen y se estabilizan, indicando que el modelo está aprendiendo sin sobreajustar.

Gráfica del MSE de aprendizaje: Similar a la pérdida, muestra el error cuadrático medio (MSE) durante el entrenamiento. Una MSE más baja indica un mejor ajuste del modelo. Si

las curvas de validación comienzan a aumentar mientras la curva de entrenamiento sigue disminuyendo.