ACTIVIDAD DE APRENDIZAJE DE LA UNIDAD 3

TÓPICOS AVANZADOS DE SOFTWARE

LUIFER VILLA LOZANO
COD: 7502010031

ROBERTO CARLOS ANGULO TINOCO
COD:7502010009

VINCENT ARTURO ROLONG MARQUEZ
COD:7502010020

TUTOR
CARLOS CACERES OCHOA

UNIVERSIDAD DE CARTAGENA FACULTAD DE INGENIERÍAS

PROGRAMA DE INGENIERÍA DE SOFTWARE

10/11/2023

INTRODUCCIÓN

Título: Aplicación de técnicas de ciencia de datos en la predicción de precios de viviendas

Objetivo:

El objetivo de esta actividad es que los estudiantes apliquen técnicas de ciencia de datos para predecir los precios de viviendas basándose en un conjunto de datos de entrenamiento.

Materiales necesarios:

- Documentación https://www.w3schools.com/datascience/

Instrucciones:

- Del siguiente vinculo:https://www.w3schools.com/datascience/
- Hacer un informe y la practica en https://colab.research.google.com/, y escoger dos de las siguientes tecnicas:
- DS Linear RegressionDS
- Regression Table
- DS Regression Info
- DS Regression Coefficients
- DS Regression P-Value
- DS Regression R-Squared
- DS Linear Regression Case

LIBRERÍAS

Pandas
Esencial para la manipulación y análisis de datos en Python. Su estructura de datos principal, el DataFrame, facilita la organización y manipulación eficiente de conjuntos de datos. Permite realizar operaciones como filtrado, agregación y limpieza de datos de manera intuitiva, lo que la convierte en una herramienta indispensable para científicos de datos y analistas.

TensorFlow
Desarrollado por Google, es una librería de código abierto fundamental para el aprendizaje automático y la creación de redes neuronales. Su flexibilidad y escalabilidad lo convierten en una elección popular para tareas de predicción, clasificación y otras aplicaciones de inteligencia artificial. TensorFlow facilita la construcción y entrenamiento de modelos complejos, proporcionando herramientas avanzadas para la implementación eficiente de algoritmos de aprendizaje automático.

NumPy
NumPy es esencial para la computación numérica en Python. Proporciona estructuras de datos eficientes, como arreglos y matrices multidimensionales, que son fundamentales para realizar operaciones numéricas eficientes. Su capacidad para manejar grandes conjuntos de datos y realizar operaciones vectorizadas lo convierte en una herramienta clave para científicos e ingenieros que trabajan en campos como la física, la estadística y la ingeniería.

Matplotlib
Poderosa para la creación de gráficos y visualizaciones en Python. Su versatilidad permite generar una amplia variedad de gráficos, desde simples visualizaciones hasta representaciones más complejas. Matplotlib es esencial para comprender patrones, tendencias y distribuciones en los datos, facilitando la comunicación efectiva de los resultados en entornos científicos y de análisis de datos.

tqdm
Herramienta valiosa para mejorar la experiencia del usuario al realizar bucles o tareas prolongadas, ya que proporciona barras de progreso visuales. Su integración sencilla en bucles permite monitorear el avance de tareas, lo que es especialmente útil en operaciones que pueden llevar tiempo. Tqdm mejora la interactividad y la comprensión del tiempo requerido para completar tareas, facilitando la gestión eficiente de procesos largos en el desarrollo de proyectos.

DESARROLLO

LINK :
https://colab.research.google.com/drive/1SyNM0wd4UHi_t3rWVQ-t9Q6Ti4LozatO?authuser=1#scrollTo=Vd8JfjCVwQpc

La actividad se centra en predecir los precios de alquiler de viviendas a partir del tamaño de estas, empleando técnicas de ciencia de datos. Se utiliza una red neuronal implementada con TensorFlow para este propósito.

A continuación se explicará el paso a paso que seguimos para el desarrollo de la misma.

ENTRENAMIENTO DE LA RED

1. IMPORTAR BIBLIOTECAS

```
import pandas as pd
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from tqdm import tqdm
```

Aquí, se importan las bibliotecas necesarias, como Pandas para la manipulación de datos, TensorFlow y Keras para la construcción y entrenamiento de la red neuronal, NumPy para operaciones numéricas, Matplotlib para visualizaciones y scikit-learn para la escala de datos y la división del conjunto de datos.

2. CARGAR Y PREPARAR LOS DATOS

```
data = pd.read_csv("Dataset.csv", sep=",")
data = data[["Size", "Rent"]]
X = np.array(data["Size"], dtype=float)
Y = np.array(data["Rent"], dtype=float)
scaler = MinMaxScaler()
X = scaler.fit_transform(X.reshape(-1, 1))
Y = scaler.fit_transform(Y.reshape(-1, 1))
```

Se carga el conjunto de datos desde un archivo CSV y se seleccionan las columnas "Size" y "Rent". Luego, se escalan los datos utilizando MinMaxScaler para normalizarlos entre 0 y 1.

3. DIVISIÓN DEL CONJUNTO DE DATOS

```
X_train, X_val, Y_train, Y_val = train_test_split(X, Y, test_size=0.2,
random_state=42)
```

Se divide el conjunto de datos en conjuntos de entrenamiento y validación.

4. CONSTRUIR EL MODELO DE RED NEURONAL

```
capa1 = tf.keras.layers.Dense(units=128, activation='relu',
input_shape=[1], kernel_regularizer=tf.keras.regularizers.l2(0.01))
capa2 = tf.keras.layers.Dense(units=64, activation='relu')
capa3 = tf.keras.layers.Dense(units=32, activation='relu')
capa_salida = tf.keras.layers.Dense(units=1)
modelo = tf.keras.Sequential([capa1, capa2, capa3, capa_salida])
```

Se define un modelo secuencial de red neuronal con tres capas ocultas y una capa de salida.

5. COMPILAR EL MODELO

```
modelo.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
    loss="mean_squared_error",
    metrics=['mae']
)
```

Se compila el modelo especificando el optimizador, la función de pérdida y las métricas que se deben monitorear durante el entrenamiento.

6. ENTRENAR EL MODELO

```
with tqdm(total=500) as pbar:
    entrenamiento = modelo.fit(X_train, Y_train, epochs=500,
batch_size=32, verbose=False, validation_data=(X_val, Y_val),
callbacks=[tf.keras.callbacks.LambdaCallback(on_epoch_end=lamb
da epoch, logs: pbar.update(1))])
```

Se entrena el modelo durante 500 épocas con un tamaño de lote de 32, y se utiliza tqdm para visualizar el progreso.

7. GUARDAR EL MODELO Y LOS PESOS

```
modelo.save('RedNeuronal.h5')
modelo.save_weights('Weights.h5')
```

Se guardan el modelo y los pesos entrenados.

8. VISUALIZAR LAS MÉTRICAS DE ENTRENAMIENTO

```
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.xlabel("Ciclos de entrenamiento")
plt.ylabel("Pérdida (MSE)")
plt.plot(entrenamiento.history["loss"], label="Entrenamiento")
plt.plot(entrenamiento.history["val_loss"], label="Validación")
plt.legend()

plt.subplot(1, 2, 2)
plt.xlabel("Ciclos de entrenamiento")
plt.ylabel("Error Absoluto Medio (MAE)")
plt.plot(entrenamiento.history["mae"], label="Entrenamiento")
plt.plot(entrenamiento.history["val_mae"], label="Validación")
plt.legend()

plt.show()
```

Se generan gráficos que muestran cómo evolucionan la pérdida y el error absoluto medio durante el entrenamiento.

9. REALIZAR PREDICCIONES

```
prediction = modelo.predict(np.array([user_input]))
prediction = scaler.inverse_transform(prediction)[0][0]
```

Se utiliza el método predict del modelo para realizar una predicción. En este caso, se pasa como entrada un arreglo NumPy que contiene el valor proporcionado por el usuario (user_input). Es importante notar que el valor de entrada debe tener la misma estructura que los datos utilizados durante el entrenamiento del modelo, por lo que se envuelve en un arreglo para que coincida con la forma esperada por la red neuronal.

Dado que durante el preprocesamiento de los datos se utilizó MinMaxScaler para normalizar tanto las características de entrada como las etiquetas de salida entre 0 y 1, es necesario invertir esta transformación para obtener la predicción en la escala original. La función inverse_transform de scaler realiza esta inversión. Luego,

se accede al valor predicho específico en la posición [0][0] del arreglo resultante, ya que el resultado es un arreglo multidimensional y se extrae el valor escalar de la predicción.

PRUEBA DE LA RED

1. CARGAR EL MODELO Y LOS PESOS

```
from tensorflow.keras.models import load_model

modelo = load_model('RedNeuronal.h5')
modelo.load_weights('Weights.h5')
```

Se utiliza load_model de TensorFlow para cargar el modelo desde el archivo 'RedNeuronal.h5'. Luego, load_weights carga los pesos del modelo desde el archivo 'Weights.h5'. Esto asegura que el modelo esté en el mismo estado en el que fue guardado después del entrenamiento.

2. SOLICITAR ENTRADA AL USUARIO

```
user_input = float(input("Ingresa el tamaño de la vivienda (normalizado) para predecir el precio del alquiler: "))
```

Se solicita al usuario que ingrese el tamaño de la vivienda (normalizado) para el cual se desea realizar la predicción. El valor ingresado se convierte a tipo float para asegurar la compatibilidad con el modelo.

3. REALIZAR PREDICCIÓN CON EL MODELO CARGADO

```
prediction = modelo.predict(np.array([user_input]))
prediction = scaler.inverse_transform(prediction)[0][0]
```

Se utiliza el modelo cargado para predecir el precio del alquiler para el tamaño de la vivienda proporcionado por el usuario. La entrada del usuario se convierte en un arreglo NumPy y se pasa al método predict. Posteriormente, se invierte la escala de la predicción utilizando scaler.inverse_transform para obtener el resultado en la escala original.

4. MOSTRAR LA PREDICCIÓN

print("La predicción es: " + str(prediction))

Se imprime la predicción calculada y se muestra al usuario el precio estimado del alquiler para la vivienda proporcionada en base al modelo previamente entrenado.

VISUALIZACIÓN DE LOS DATOS

1. IMPORTAR BIBLIOTECAS

import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt

Se importan las bibliotecas necesarias: Pandas para la manipulación de datos, statsmodels para el análisis de regresión y Matplotlib para la visualización de gráficos.

2. CARGAR Y EXPLORAR DATOS

data = pd.read_csv("Dataset.csv", parse_dates=["Posted On"], sep=",")

Los datos se cargan desde un archivo CSV y se especifica que la columna "Posted On" contiene fechas.

3. VISUALIZACIÓN DE DATOS

```
plt.figure(figsize=(12, 4))
plt.subplot(1, 3, 1)
plt.scatter(data["BHK"], data["Rent"])
plt.xlabel("Número de habitaciones (BHK)")
plt.ylabel("Precio del alquiler (Rent)")
plt.title("BHK vs. Rent")

plt.subplot(1, 3, 2)
plt.scatter(data["Size"], data["Rent"])
plt.xlabel("Tamaño (Size)")
plt.ylabel("Precio del alquiler (Rent)")
plt.title("Size vs. Rent")

plt.subplot(1, 3, 3)
furnishing_groups = data.groupby("Furnishing
Status")["Rent"].mean()
furnishing_groups.plot(kind="bar")
plt.xlabel("Estado de amueblamiento")
plt.ylabel("Precio promedio del alquiler (Rent)")
plt.title("Furnishing Status vs. Rent")
```

Se crean tres subgráficos para visualizar las relaciones entre las variables. Dos gráficos de dispersión muestran cómo el número de habitaciones (BHK) y el tamaño (Size) se relacionan con el precio del alquiler (Rent). El tercer gráfico de barras muestra el precio promedio del alquiler según el estado de amueblamiento.

4. PREPARACIÓN DE DATOS PARA EL MODELO DE REGRESIÓN

```
data = data[["BHK", "Rent"]]
data.dropna(inplace=True)

x = data["BHK"]
y = data["Rent"]

x = sm.add_constant(x)
```

Se seleccionan las columnas relevantes y se eliminan las filas con valores nulos. Se prepara la variable independiente (x) y se agrega una constante para el término independiente en el modelo de regresión.

5. AJUSTE DEL MODELO DE REGRESIÓN LINEAL

model = sm.OLS(y, x).fit()

Se utiliza el modelo de mínimos cuadrados ordinarios (OLS) de statsmodels para ajustar una regresión lineal a los datos.

6. IMPRIMIR RESUMEN DEL MODELO

print(model.summary())

Se imprime un resumen detallado del modelo, que incluye estadísticas como coeficientes, errores estándar, valores p, y estadísticas de ajuste.

7. MOSTRAR GRÁFICAMENTE EL MODELO

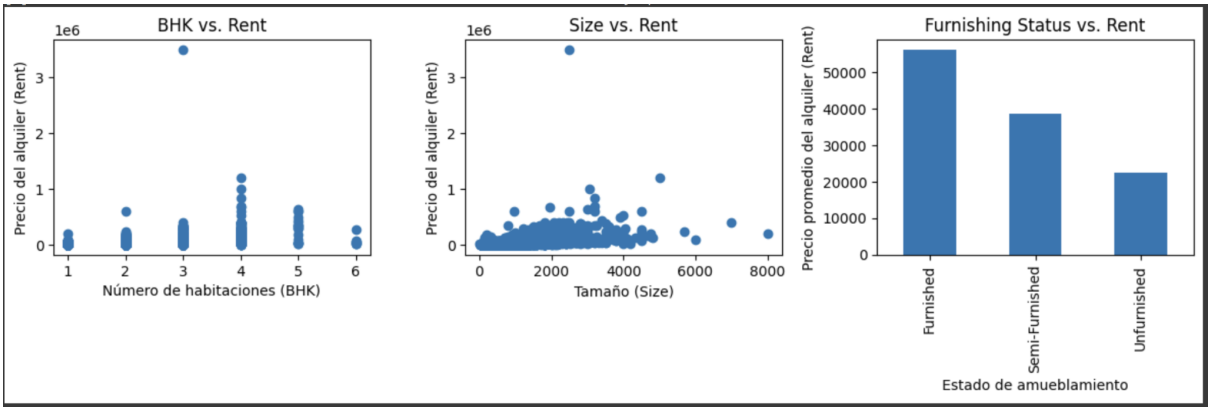
```
plt.tight_layout()
plt.show()
```

Se ajusta el diseño de las subgráficas y se muestra el gráfico.

Regression table

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   Rent   R-squared:                       0.137
Model:                            OLS   Adj. R-squared:                  0.137
Method:                 Least Squares   F-statistic:                     751.1
Date:                Fri, 10 Nov 2023   Prob (F-statistic):          1.21e-153
Time:                        03:08:48   Log-Likelihood:                -59853.
No. Observations:                4746   AIC:                         1.197e+05
Df Residuals:                    4744   BIC:                         1.197e+05
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const      -3.731e+04   2840.791    -13.134      0.000   -4.29e+04   -3.17e+04
BHK          3.47e+04   1266.020     27.407      0.000    3.22e+04    3.72e+04
==============================================================================
Omnibus:                    11523.724   Durbin-Watson:                   1.767
Prob(Omnibus):                  0.000   Jarque-Bera (JB):        230160879.095
Skew:                          24.832   Prob(JB):                         0.00
Kurtosis:                    1080.697   Cond. No.                         7.11
==============================================================================
```

Linear regression



CONCLUSIONES

Como conclusión tenemos que el uso de algoritmos y modelos como la regresión lineal o las redes neuronales en la ciencia de datos es una herramienta poderosa para predecir y modelar rentas. Estos modelos permiten predicciones basadas en datos históricos que son esenciales para que inquilinos, propietarios, inversores y planificadores urbanos tomen decisiones informadas en mercados cambiantes.

Otro aspecto importante es la capacidad de la ciencia de datos para analizar grandes volúmenes de datos en tiempo real, lo que permite realizar ajustes continuos en función de la dinámica del mercado. Esto permite una toma de decisiones rápida y adaptable a medida que evolucionan las condiciones del mercado.

Al final podemos ver que todos estos conceptos resultan en importantes aportes para nuestro futuro profesional y como ingenieros en general para todo tipo de industrias y necesidades futuras.