Documentación Prueba Técnica 1

Prueba

Construye un modelo de identificación de entidades nombradas usando las bibliotecas que consideres más apropiadas. Para el desarrollo de este reto debes usar datos abiertos gubernamentales. A continuación encontraras algunas fuentes de datos recomendadas.

Tesis

Tratados

Comunicados

En tu script debes indicar claramente la fuente de datos que est´as utilizando.

Solución

Los datos recomendados se intentaron probar pero el scraping era bloqueado asi que se usó otro dataset del gobierno de la Ciudad de México ya que los federales no se podían descargar

Dataset: https://datos.cdmx.gob.mx/dataset/carpetas-de-investigacion-fgj-de-la-ciudad-de-mexico

A continuación, se proporciona una explicación exhaustiva del código paso a paso:

Importar las bibliotecas necesarias:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
```

Las bibliotecas utilizadas son:

pandas: se utiliza para trabajar con estructuras de datos tabulares y manipular conjuntos de datos.

train_test_split de sklearn.model_selection: se utiliza para dividir los datos en conjuntos de entrenamiento y prueba.

LabelEncoder de sklearn.preprocessing: se utiliza para convertir las etiquetas de texto en números.

DecisionTreeClassifier de sklearn.tree: se utiliza para crear un modelo de clasificación basado en árboles de decisión.

accuracy_score y confusion_matrix de sklearn.metrics: se utilizan para evaluar el rendimiento del modelo.

matplotlib.pyplot como plt: se utiliza para visualizar la matriz de confusión.

Especificar la ruta del conjunto de datos:

dataset_path = "datos_delitos.csv"

La variable dataset_path contiene la ruta del archivo CSV que contiene los datos delitos.

Leer el conjunto de datos:

data = pd.read_csv(dataset_path)

El conjunto de datos se lee desde el archivo CSV y se almacena en la variable data como un DataFrame de Pandas.

Explorar los datos:

print(data.head())

Se imprime una vista previa de los primeros registros del conjunto de datos para tener una idea de su estructura y contenido.

Eliminar columnas innecesarias o irrelevantes:

data = data.drop(['ao_hechos', 'fecha_hechos', 'hora_hechos', 'ao_inicio', 'mes_inicio', 'fecha_inicio', 'hora_inicio'], axis=1)

Se eliminan las columnas innecesarias o irrelevantes del DataFrame utilizando el método drop(). Las columnas que se eliminan son 'ao_hechos', 'fecha_hechos', 'hora_hechos', 'ao_inicio', 'mes_inicio', 'fecha_inicio' y 'hora_inicio'.

Manejar valores faltantes o nulos:

data = data.dropna()

Se eliminan las filas que contienen valores faltantes o nulos utilizando el método dropna() del DataFrame.

Convertir la variable objetivo en números usando LabelEncoder:

le = LabelEncoder()

data['mes_hechos'] = le.fit_transform(data['mes_hechos'])

Se crea una instancia de LabelEncoder y se ajusta a la columna 'mes_hechos' del DataFrame. Esto asigna un número único a cada valor distinto en la columna 'mes_hechos'. Los valores numéricos se utilizan para entrenar el modelo.

Separar características y variable objetivo:

X = data.drop('delito', axis=1)

y = data['mes_hechos']

Las características se almacenan en la variable X, eliminando la columna 'delito'. La variable objetivo se almacena en la variable y, que es la columna 'mes_hechos'.

Dividir los datos en conjuntos de entrenamiento y prueba:

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

Se utiliza la función train_test_split para dividir los datos en conjuntos de entrenamiento y prueba. Se asigna el 80% de los datos para entrenamiento (X_train y y_train) y el 20% para prueba (X_test y y_test). El parámetro test_size=0.2 indica que el conjunto de prueba será el 20% del tamaño total de los datos. El parámetro random_state=42 se utiliza para garantizar la reproducibilidad de la división de datos.

Crear un modelo de clasificación basado en árboles de decisión:

model = DecisionTreeClassifier()

Se crea una instancia del clasificador DecisionTreeClassifier como model.

Entrenar el modelo utilizando los datos de entrenamiento:

model.fit(X_train, y_train)

Se entrena el modelo utilizando los datos de entrenamiento (X_train y y_train) con el método fit().

Realizar predicciones en el conjunto de prueba:

y_pred = model.predict(X_test)

Se utilizan los datos de prueba (X_test) para realizar predicciones utilizando el modelo entrenado. Las predicciones se almacenan en y_pred.

Calcular la precisión del modelo:

accuracy = accuracy_score(y_test, y_pred)

print("Precisión del modelo:", accuracy)

Se calcula la precisión del modelo comparando las etiquetas reales del conjunto de prueba (y_test) con las predicciones del modelo (y_pred) utilizando la función accuracy_score. La precisión se almacena en accuracy y se imprime en pantalla.

Calcular y mostrar la matriz de confusión:

conf_matrix = confusion_matrix(y_test, y_pred)

print("Matriz de confusión:")

print(conf_matrix)

Se calcula la matriz de confusión utilizando la función confusion_matrix, que compara las etiquetas reales (y_test) con las predicciones del modelo (y_pred). La matriz de confusión se almacena en conf_matrix y se imprime en pantalla.

Mostrar la matriz de confusión en forma de gráfico:

plt.imshow(conf_matrix, cmap='Blues')

plt.title("Matriz de Confusión")

plt.colorbar()

plt.xlabel("Clases Predichas")

plt.ylabel("Clases Verdaderas")

plt.show()

La matriz de confusión se muestra en forma de gráfico utilizando imshow() de matplotlib.pyplot. Se agrega un título, una barra de color, etiquetas de ejes y se muestra el gráfico en pantalla.