UNIVERSIDAD DEL VALLE DE GUATEMALA

CC3105 - Machine learning engineering

Sección 10

Ejercicio 5

SEBASTIAN ARISTONDO PEREZ 20880

PABLO DANIEL GONZALEZ RAMOS 20362

JOSE DANIEL GONZALEZ CARRILLO 20293

GUATEMALA, 27 de agosto de 2024

Usamos django rest para generar la API. Creamos dos servicios, uno para entreno y otro para predicción.

```
import os
import logging

from rest_framework import status
from urllib.parse import urlparse

import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
import pickle


base_path = os.getcwd()
data_path=os.path.normpath(base_path+os.sep+'data')
pickle_path=os.path.normpath(base_path+os.sep+'models')
log_path=os.path.normpath(base_path+os.sep+'logs')

if not os.path.exists(log_path):
    os.makedirs(log_path)

log_file = os.path.join(log_path, 'training.log')
logging.basicConfig(
    filename=log_file,
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s',
)

class Training:

    def accuracy_measures(self, y_test, y_pred):
        accuracy = accuracy_score(y_test, y_pred)
        logging.info(f"Precisión del modelo: {accuracy * 100:.2f}%")

        # Mostrar el reporte de clasificación
```

```
        logging.info("Reporte de clasificación:")
        logging.info(classification_report(y_test, y_pred))

        # Mostrar la matriz de confusión
        logging.info("Matriz de confusión:")
        logging.info(confusion_matrix(y_test, y_pred))

        return accuracy

    def best_features(self, pipeline, X):
        selector = pipeline.named_steps['selector']
        mask = selector.get_support()  # Array booleano de las
características seleccionadas
        selected_features = X.columns[mask]
        logging.info("Características seleccionadas: %s",
selected_features)

    def train(self, request):
        return_dict=dict()
        try:
            data = pd.read_csv(data_path + '/train_data.csv')
            data['categoria_precio'] = pd.qcut(data['Price'], q=4,
labels=['Bajo', 'Medio Bajo', 'Medio Alto', 'Alto'])
            df_numerico =
data.select_dtypes(include=['number']).dropna(axis=1)
            df_numerico = df_numerico.drop(columns=['Longtitude',
'Lattitude'])
            X = df_numerico.drop(columns=['Price'])
            y = data['categoria_precio']
            X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)
            pipeline = Pipeline([
                ('scaler', StandardScaler()),
                ('selector', SelectKBest(score_func=f_classif, k=7)),
                ('classifier', RandomForestClassifier(n_estimators=100,
random_state=42))
            ])
            pipeline.fit(X_train, y_train)
            y_pred = pipeline.predict(X_test)

            accuracy = self.accuracy_measures(y_test, y_pred)
            self.best_features(pipeline, X)
```

```
            pickle_file =
os.path.normpath(pickle_path+os.sep+'model.pkl')
            pickle.dump(pipeline, open(pickle_file, 'wb'))

            selector = pipeline.named_steps['selector']
            selected_columns = X.columns[selector.get_support()]
            columns_file = os.path.normpath(pickle_path + os.sep +
'selected_columns.pkl')
            with open(columns_file, 'wb') as f:
                pickle.dump(selected_columns.tolist(), f)

            return_dict['response'] = 'Model Trained Successfully'
            return_dict['status']=status.HTTP_200_OK
            return return_dict

        except Exception as e:
            return_dict['response']="Exception when training the
module: "+str(e.__str__)
            return_dict['status']=status.HTTP_500_INTERNAL_SERVER_ERROR
            return return_dict
```

```
# prediction.py
import os
import pickle
import pandas as pd

base_path = os.getcwd()
pickle_path = os.path.normpath(base_path + os.sep + 'models')

class Predictor:
    def __init__(self):
        # Cargar el modelo y las columnas seleccionadas al inicializar
la clase
        self.model = self.load_model()
        self.selected_columns = self.load_selected_columns()

    def load_model(self):
        try:
            pickle_file = os.path.normpath(pickle_path + os.sep +
'model.pkl')
            with open(pickle_file, 'rb') as f:
                model = pickle.load(f)
            return model
```

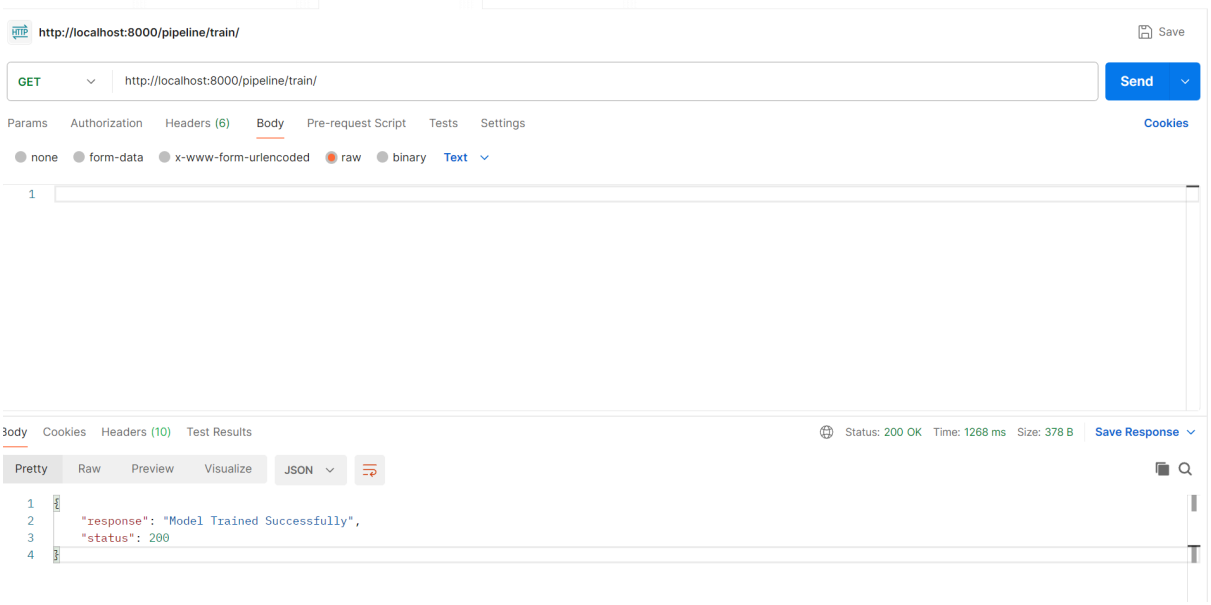
```
        except Exception as e:
            raise Exception(f"Error loading model: {str(e)}")

    def load_selected_columns(self):
        try:
            columns_file = os.path.normpath(pickle_path + os.sep +
'selected_columns.pkl')
            with open(columns_file, 'rb') as f:
                selected_columns = pickle.load(f)
            return selected_columns
        except Exception as e:
            raise Exception(f"Error loading selected columns:
{str(e)}")

    def predict(self, data):
        """
        Método para predecir una o muchas observaciones.
        `data` puede ser un diccionario de una observación o una lista
de diccionarios.
        """
        try:
            if isinstance(data, dict):  # Si es un solo diccionario,
convertir a DataFrame
                input_data = pd.DataFrame([data])
            elif isinstance(data, list):  # Si es una lista de
diccionarios, convertir a DataFrame
                input_data = pd.DataFrame(data)
            else:
                raise ValueError("Invalid data format. Must be a dict
or list of dicts.")

            # Usar solo las columnas seleccionadas para la predicción
            input_data = input_data[self.selected_columns]

            # Realizar la predicción
            predictions = self.model.predict(input_data)
            return predictions.tolist()  # Devolver la predicción como
lista para mayor flexibilidad
        except Exception as e:
            raise Exception(f"Error during prediction: {str(e)}")
```

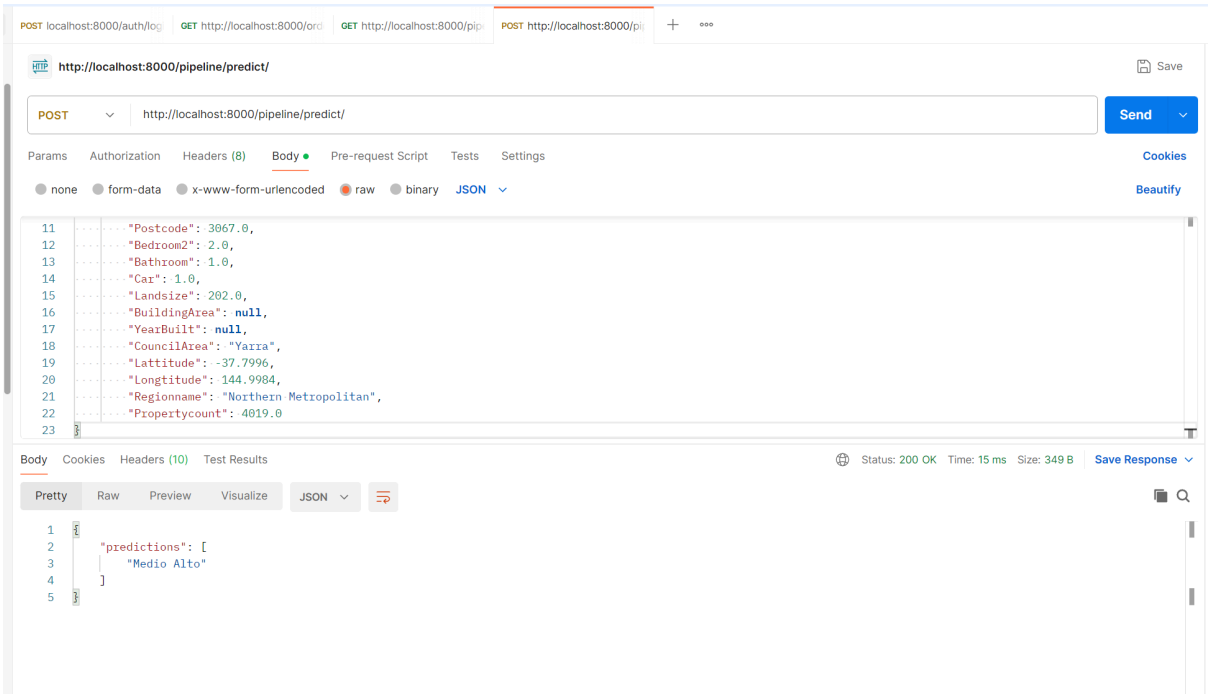
En el entreno se consumieron los datos que hemos estado usando en todos los labs y se realizó el pipeline de entrenamiento. En la predicción se usó el modelo generado en un .pkl.

Se crearon vistas para poder servir los endpoints:

```
 from django.shortcuts import render
from model_pipeline.services.training import Training
from model_pipeline.services.prediction import Predictor
from rest_framework.response import Response
from rest_framework import status
from rest_framework.views import APIView

predictor = Predictor()

class TrainModel(APIView):
    def get(self, request):
        training = Training()
        response_dict = training.train(request)
        print(response_dict)
        return Response(response_dict)

class PredictView(APIView):
    def post(self, request):
        try:
            # Obtener los datos del request
            data = request.data

            # Realizar la predicción utilizando el servicio de
predicción
            predictions = predictor.predict(data)

            response = {
                'predictions': predictions
            }

            return Response(response, status=status.HTTP_200_OK)
        except Exception as e:
            return Response({'error': str(e)},
status=status.HTTP_500_INTERNAL_SERVER_ERROR)
```

Este fue el resultado de los endpoints





Además, se creó un dockerfile para el proyecto

```
# Python runtime as a image
FROM python:3.10
# install dependencies
COPY requirements.txt requirements.txt
RUN pip install --no-cache-dir -r requirements.txt
#Mounts the application code to the image andexpose 8post 8000
COPY . code
```

```
WORKDIR /code
EXPOSE 8000
# runs the production server
ENTRYPOINT ["python", "manage.py"]
CMD ["runserver", "0.0.0.0:8000"]
```