Santiago Campo 201921995

Camilo Falla 201821059

Pablo Pastrana 201822920

Proyecto Clasificador de Películas – Parte 2

Grupo 19

1. Proceso de automatización de preparación de datos, construcción del modelo, persistencia del modelo y acceso por medio de un API.

Preparación de datos y construcción del modelo:

Llevamos a cabo la preparación de datos y construcción del modelo a través de la creación de una clase Modelo() la cual carga el modelo creado a través de joblib y prepara los datos de manera automática para ser consumidos por el modelo. Para esto llevamos a cabo la creación de un Pipeline que contiene un vectotizador y un tfid converter los cuales transforman los datos ingresados por el usuario. Además, llevamos a cabo transformaciones para lematizar el texto, eliminar valores repetidos y stop words.

A continuación se puede evidenciar parte del pipeline usado en la clase Modelo():

```
vectorizer = CountVectorizer(max_features=18000, min_df=5, max_df=0.7, stop_words=stopwor
#X = vectorizer.fit_transform(documents).toarray()


tfidfconverter = TfidfTransformer()
#X = tfidfconverter.fit_transform(X).toarray()

steps = [("vect", vectorizer),("tfid", tfidfconverter)]
pipeline = Pipeline(steps)

X = pipeline.fit_transform(documents)

return self.model.predict(X)[-1]
```

Persistencia del modelo

Hicimos uso de la librería joblib para persistir el modelo en el sistema de manera que se pueda usar cuando se haga un llamado al endpoint. De esta manera, hicimos uso del método load() de la librería para recuperar el modelo creado en la primera etapa.

```
class Modelo:
    def __init__(self):
        self.model = load("ModeloFinal.joblib")
```

Acceso por medio de una API

Hicimos uso de la librería FastAPI con el fin de crear una API REST con la cual pueda interactuar el usuario de nuestra aplicación. Con FastAPI creamos un endpoint de tipo POST en el cual el usuario puede acceder al modelo creado y llevar a cabo predicciones con los datos suministrados.

```
3   from fastapi import FastAPI
4
5   from DataModel import DataModel
6
7   from FinalModel import Modelo
8
9   app = FastAPI()
10
11
12  @app.post("/predict/")
13  def predict(dataModel: DataModel):
14      modelo = Modelo()
15      resp = modelo.predict(dataModel.descripcion)
16      return resp
17
```

2. Desarrollo de la aplicación y justificación.

Decidimos crear una aplicación web para que el cliente pueda llevar a cabo clasificaciones de manera automática y muy eficaz. El rol del usuario de nuestra aplicación es el Gerente de Marketing de una organización de streaming. Creemos que este cliente puede hacer uso de nuestra aplicación para determinar cuáles películas y series tienen mayor acogida por los usuarios dependiendo de la cantidad de reseñas positivas que tengan de manera automática. Además, el Gerente de Marketing también podrá hacer uso de la aplicación para llevar a cabo marketing dirigido a los clientes al mostrarles exclusivamente reseñas positivas y que estos puedan llevarse una buena impresión del negocio.

De esta manera, para crear la aplicación web hicimos uso de la librería Dash en Python, la cual crea aplicaciones Flask de manera muy sencilla e intuitiva. Dicha librería nos permite escribir código html en Python para crear el layout de la aplicación.

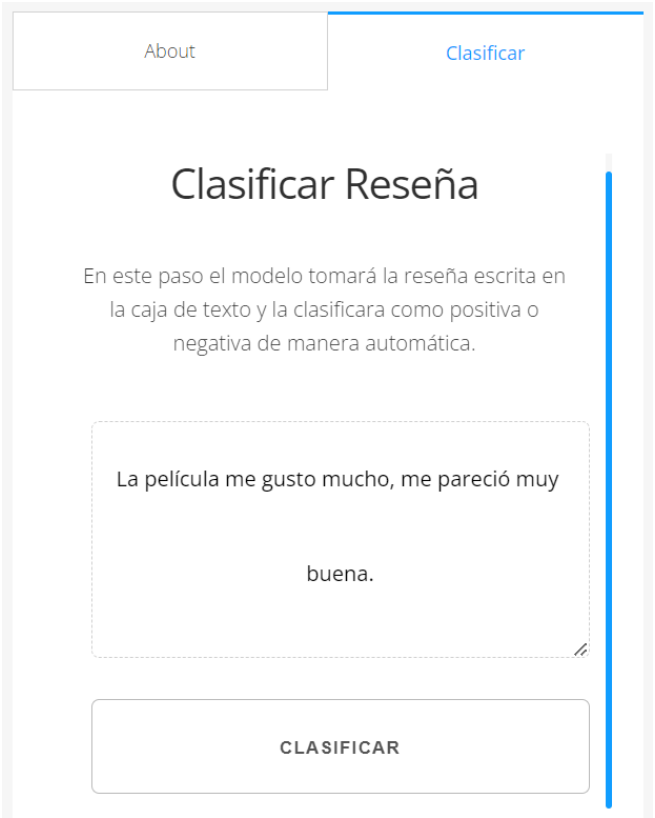
```
def layout():
    return html.Div(id='mhp-page-content', className='app-body', children=[
        dcc.Loading(parent_className='dashbio-loading', children=html.Div(
            id='mhp-graph-div',
        )),

        html.Div(id='manhattan-control-tabs', className='control-tabs', children=[
            dcc.Tabs(id='manhattan-tabs', value='what-is', children=[
                dcc.Tab(
                    label='About',
                    value='what-is',
                    children=html.Div(className='control-tab', children=[
                        html.H4(className='what-is', children='Descripción Herramienta'
                        html.P('Esta herramienta hace uso de técnicas de Machine Learni
                               'y de inteligencia artificial para clasificar reseñas de
                               'de manera automática y en segundos. '
                               'La herramienta tiene como próposito clasificar reseñas
                               ),
                        html.Br(), html.Br(),
                        html.Img(
                            src='{}'.format(
                                base64.b64encode(
                                    open(
                                        './assets/pelicula.png', 'rb'
                                    ).read()
                                ).decode()
                            ),
```

Por otro lado, la librería también nos permite crear peticiones a nuestra API para que los usuarios puedan tener acceso al modelo creado:

```
        def update_texto(n_clicks, value):
            if n_clicks != 0:

                url = 'http://127.0.0.1:8000/predict/'
                myobj = {'descripcion': value}

                x = requests.post(url, json = myobj)

                resp = "Positiva!"

                if "positivo" in x.text:
                    resp = "Positiva!"
                else:
                    resp = "Negativa!"

                return html.Div(children=[html.Br(),html.Br(),
                                          html.B("Reseña Clasificada"),
                                              html.P(value),
                                              html.Br(), html.Br(),
                                              html.B("Clasificacion"),
                                              html.Br(),
                                              html.P(resp),
                                              html.Br(),]
                                )
```

De este modo, creamos la siguiente interfaz web para el cliente propuesto. En la pagina principal encontramos una descripción de la herramienta y su funcionamiento.

En la siguiente pagina podemos ver la funcionalidad de clasificar una reseña ingresada por el usuario de manera automática. De esta manera el usuario puede ingresar una reseña que quiera clasificar y el modelo propuesto llevara a cabo la clasificación.



Los resultados se pueden visualizar en la parte derecha de la pantalla, el la cual se evidencia la reseña que se ingresó y la clasificación dada por el modelo.

De esta manera, el cliente puede llevar a cabo clasificaciones de manera rápida y automática con el fin de tener mayor conocimiento acerca de la acogida de sus películas y series, lo cual le permitirá tomar mejores decisiones de a donde llevar su contenido para que tenga mayor impacto. Además, consideramos que nuestro cliente puede aprovechar nuestra aplicación para identificar qué películas y series son más populares entre los usuarios según el número de reseñas positivas de forma automática. Además, el Gerente de Marketing puede utilizar la aplicación para realizar campañas de marketing dirigidas a los clientes, mostrándoles únicamente reseñas positivas con el objetivo de crear una buena impresión del negocio.

3. Trabajo en equipo

Roles asignados:

Líder de proyecto – Camilo Falla Moreno

Encargado de la gestión del proyecto, agendar las reuniones y del desarrollo de la API web y la aplicación web.

Ingeniero de datos – Santiago Campo

Encargado del desarrollo del modelo analítico de datos. Creador del pipeline final de datos.

Ingeniero de software – Pablo Pastrana

Encargado del correcto despliegue de la aplicación final. Revisor de la documentación de las librerías usadas.

- Repartiríamos 100 puntos de la siguiente manera;

Camilo Falla: 34

Santiago Campo: 33

Pablo Pastrana: 33

Punto a mejorar

Podríamos tener una mayor cantidad de reuniones para poder revisar la calidad final del proyecto a mayor detalle. Además, podríamos llevar a cabo revisiones con externos para así ver opiniones de otras personas y mejorar el diseño final.