Andres Molina – 201923434

Josue Rivera –201914138

Jaime Alfonso- 202116525

Proyecto 1 Entrega 2 Inteligencia de Negocios

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

Con respecto al proceso de esta primera parte, utilizamos la limpieza de datos de la primera etapa. En este caso tomamos los datos ya organizados en tokens y sentimiento, y a estos datos les aplicamos el modelo que nos dio los mejores resultados en la primera etapa, el Random Forest. El modelo y la vectorizacion de los datos los transformamos a archivos .joblib para poder hacer uso de ellos a la hora de desarrollar la aplicación. Una vez que se tiene el modelo y la vectorización de los datos en formato .joblib, se procede a la construcción de una API REST con Django. La API consta de tres endpoints: uno para predecir el sentimiento de un texto ingresado, otro para entrenar el modelo y el último para obtener las métricas del modelo.

El endpoint para predecir el sentimiento de un texto utiliza el modelo y la vectorización previamente guardados en archivos .joblib para realizar la predicción. Para ello, se envía una petición HTTP POST con un JSON que contiene el texto a analizar. La API procesa el texto, lo vectoriza y realiza la predicción con el modelo cargado previamente.

El endpoint para entrenar el modelo se encarga de recibir un conjunto de datos y entrenar el modelo con ellos. Primero se limpian los datos con el mismo proceso que se utilizó en la primera etapa, luego se vectorizan y finalmente se entrena el modelo con los datos procesados. Una vez que el modelo está entrenado, se guardan los archivos .joblib actualizados para su uso en futuras predicciones, así, se va entrenando cada vez más el modelo a medida que se va utilizando.

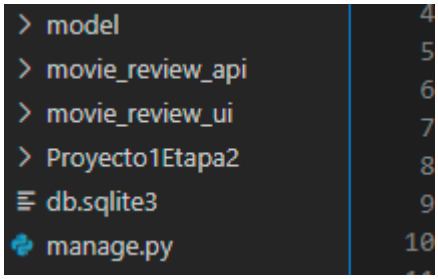
Por último, el endpoint para obtener las métricas del modelo simplemente carga un archivo .json que contiene las métricas previamente calculadas en el proceso de entrenamiento del modelo.

En resumen, el proceso de automatización del proceso de preparación de datos, construcción del modelo, persistencia del modelo y acceso por medio de API consiste en la creación de una API REST con Django que utiliza el modelo previamente entrenado y vectorización previamente realizada para realizar predicciones de sentimiento de texto. Además, la API permite la actualización del modelo con nuevos datos y la obtención de métricas del modelo.

2. Desarrollo de la aplicación y justificación.

Para el desarrollo de la aplicación decidimos utilizar el framework Django debido a sus múltiples funcionalidades para el diseño web mediante Python y, principalmente, a las facilidades que presenta a la hora de realizar la integración con nuestro modelo de analítica en específico (Random Forest), las cuales se verían reducidas al mínimo si utilizáramos otros Frameworks (en especial aquellos no enfocados al desarrollo en Python).

La se divide en 4 partes principales que son: la aplicación principal, el api, el modelo y por último la ui, además de la base de datos y el archivo utilizado para correr el servicio.



Gran parte de los archivos dentro de las carpetas son generados automáticamente por Django, siendo que los cambios se harían, primero que todo, en el archivo url.py de la carpeta Proyecto1Etapa2, añadiendo las urls dedicadas a la comunicación con el api y la ui. Además, en esta misma carpeta, se modificaría el archivo setting.py, añadiendo las aplicaciones nuevas en installed apps incluyendo el rest framework (esencial para el desarrollo del api), y, se referencian las urls principales de la aplicación en general.

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('', include('movie_review_ui.urls')),
    path('api/', include('movie_review_api.urls')),
    path('admin/', admin.site.urls),
]
```

Por el lado del api, lo principal seria modificar el archivo views, que es lo que contiene todo lo relacionado al pipeline. Primero se desarrolla la función encargada del procesamiento del texto ingresado por medio de la interfaz gráfica (tokenizacion, lematizacion, tc.), luego se encuentra la funcionalidad que se encarga de proporcionar las diferentes métricas de calidad del modelo, seguida de las 2 principales funciones de la aplicación: la que se encarga de la prediccion (predict) y la que se encarga del entrenamiento (train). Predict es la más involucrada con el pipeline siendo la que hace la vectorización del texto y la predicción final que indica si el comentario es positivo o negativo, y train es la que se encarga del entrenamiento del modelo, leyendo el archivo proporcionado para, posteriormente,

hacer la separación entre datos de entrenamiento y de prueba, realizar la distincion entre columna de tokens y de sentimiento, realizar el entrenamiento y acabar encontrando los resultados de este (exportendo por medio de joblib) junto con las métricas de calidad del modelo (exportadas al json que se utilizara en la función de métricas mencionada anteriormente.

```python
def process_text(text):

    text.replace('[^a-zA-Z ]', '')
    text.lower()

    # Tokenize text
    tokens = word_tokenize(text)

    # Remove stop words
    stop_words = stopwords.words('english')
    tokens = [token for token in tokens if token not in stop_words]

    # Stemming
    stemmer = LancasterStemmer()
    tokens = [stemmer.stem(token) for token in tokens]

    # Lemmatization
    lemmatizer = WordNetLemmatizer()
    tokens = [lemmatizer.lemmatize(token) for token in tokens]

    # Join tokens
    text = ' '.join(tokens)

    return text
```

```python
class ModelMetrics(views.APIView):
    with open('model\model_metrics.json', 'r') as f:
        model_metrics = json.load(f)

    # Return json with the model metrics
    def get(self, request):
        return Response(self.model_metrics, status=status.HTTP_200_OK)

class Predict(views.APIView):
    def post(self, request):
        predictions = []

        for entry in request.data:
            try:
                # Vectorize text
                vectorized_text = vectorizer.transform([entry['text']])

                # Predict
                prediction = model.predict(vectorized_text)[0]
                predictions.append({'text': entry['text'], 'sentimiento': prediction})

            except Exception as e:
                return Response({'error': f'Error al procesar el texto {str(e)}'}, status=status.HTTP_400_BAD_REQUEST)

        return Response(predictions, status=status.HTTP_200_OK)
```

```
X_count = vectorizer.fit_transform(X)

model.fit(X_count, Y)

# Import test data from model/processed_data.csv
db_route = 'model/processed_data_min.csv'
df_originales = pandas.read_csv(db_route, encoding = 'ISO-8859-1')

df_originales.dropna(inplace=True)

X_test = df_originales['tokens']
Y_test = df_originales['sentimiento']
X_count_test = vectorizer.transform(X_test)

predictions = model.predict(X_count_test)

# Create a json with the previous values
model_json = {
    'accuracy': accuracy_score(Y_test, predictions),
    'f1': f1_score(Y_test, predictions, average='weighted'),
    'precision': precision_score(Y_test, predictions, average='weighted'),
    'recall': recall_score(Y_test, predictions, average='weighted')
}

# Save the json in a file
with open('model/model_metrics.json', 'w') as outfile:
    json.dump(model_json, outfile)

# Save model with joblib
joblib.dump(model, 'model/modelo_random_forest.joblib')
joblib.dump(vectorizer, 'model/vectorizer.joblib')
```

Por último, en esta sección, se modifica el archivo urls.py para que contenga lo correspondiente a métricas predicciones y entrenamiento.

```
urlpatterns = [
    path('', ModelMetrics.as_view(), name='api'),
    path('metrics', ModelMetrics.as_view(), name='ModelMetrics'),
    path('predict', Predict.as_view(), name='Predict'),
    path('train', Train.as_view(), name='Train'),
]
```

Con respecto a la UI, se incluye en el views la función de procesamiento de texto sin ningún cambio con respecto a la anterior, la función home encargada de renderizar la petición, la función predict con algunos cambios para enfocarse más en recibir el texto para la predicción (formulario) y mostrar la probabilidad de acierto de la predicción, una función para manejar los errores, y por último la funcionalidad encargada de mostrar la predicción en pantalla (showPrediction) que despliega las métricas ya calculadas en el entrenamiento e indicar de forma explícita si el resultado de la predicción es que el comentario ingresado en el formulario es positivo o negativo.

```
def home(request):
    return render(request, 'home.html')
```

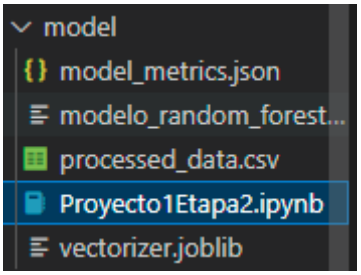
```
def error(request, error):
    return render(request, 'error.html', {'error': error})
```

```python
def predict(request):

    if request.method == 'POST':
        probabilidad = 0
        prediccion = 0

        # Obtener los datos del formulario
        text = request.POST['mensaje_para_prediccion']
        text = process_text(text)

        try:

            # Vectorizar el texto
            text_vectorized = vectorizer.transform([text])

            prediccion = model.predict(text_vectorized)[0]
            probabilidad = model.predict_proba(text_vectorized)[0][prediccion]
            probabilidad = int(probabilidad * 100)

            return redirect('showPrediction', prediccion, probabilidad)
        except Exception as e:
            return redirect('error', e)

    return render(request, 'predict.html')
```

```python
def showPrediction(request, prediction, probability):

    with open('model\model_metrics.json', 'r') as f:
        model_metrics = json.load(f)

    accuracy = float(model_metrics['accuracy']) * 100
    precision = float(model_metrics['precision']) * 100
    recall = float(model_metrics['recall']) * 100
    f1_score = float(model_metrics['f1']) * 100

    if prediction == 0:
        prediction = 'Clasificacion negativa'
    else:
        prediction = 'Clasificacion positiva'

    context = {
        'accuracy': accuracy,
        'precision': precision,
        'recall': recall,
        'f1_score': f1_score,
        'prediction': prediction,
        'probability': probability
    }
    return render(request, 'show_prediction.html', context)
```

Por último, y al igual que en la sección anterior, se modifica el urls.py para incluir los paths necesarios que en este caso serían los relacionados con la predicción, la visualizacion de los resultados de esta y de sus métricas, y, la visualización cuando ocurre un error.

```python
urlpatterns = [
    path('', home, name='home'),
    path('predict', predict, name='predict'),
    path('showPrediction/<int:prediction>/<int:probability>', showPrediction, name='showPrediction'),
    path('error/<str:error>', error, name='error')
]
```



La última sección de la aplicación seria la que contiene lo relacionado estrictamente con el modelo en sí, conteniendo el cuaderno utilizado para la creación del modelo de analítica, los datos proporcionados para entrenamiento y prueba, los joblibs resultantes del proceso de exportación del modelo y del vectorizador, y, el json con las métricas de calidad el cual se va actualizando con cada entrenamiento.

Decidimos dividirlo en estas 4 partes ya que sentimos que es la mejor forma de dividir las diferentes responsabilidades de cada una de las partes siendo que se pueden desarrollar por separado y sin embargo se pueden conectar fácilmente entre ellas, evitando así que si ocurre un error en alguna de ellas pueda afectar de manera grave a las otras. Así una se encarga exclusivamente del proceso de pipeline y el api, otra de todo lo que se le muestra al usuario, otra de lo relacionado con el modelo en términos de su creación y sus datos y la última que se encarga de unificar las anteriores (por así decirlo) de forma que sean una aplicación correctamente establecida.

3. Resultados

Se realizo la creación del pipeline y del api, para mostrar en ejecución la función de predicción de reseñas a partir del modelo dado y en si se pudo identificar que la longitud de una reseña llega a un punto en mientras más larga es afecta su probabilidad a crear un falso negativo y en si afecta la certeza de la predicción del modelo.

También se habló con Valery Calixto, nuestra pareja de estadística de manera que pudimos establecer la forma de mostrar los resultados de la primera etapa de manera que una persona sin conocimientos tan formales pudiera entender de manera más clara el funcionamiento del modelo. Esta es una de las razones por las cuales se creó una parte del api donde cualquier persona puede escribir una reseña y probar las habilidades y certeza de las predicciones del modelo.

Link del video: https://youtu.be/YpMoUfex0UY

Link de la wiki: https://github.com/amolinam08/InteligenciaDeNegociosG23/wiki

4. Trabajo en equipo

- Líder de proyecto: Andres Molina

Andrés Molina es el líder del proyecto y está a cargo de la gestión de este. Ha definido las fechas de reuniones y pre-entregables del grupo, y ha verificado las asignaciones de tareas para que la carga sea equitativa. Andres, estuvo pendiente por medio del grupo de WhatsApp para que se empezara a trabajar el proyecto con antelación. Se ha encargado de subir la entrega del grupo y ha tomado decisiones cuando no ha habido consenso, siempre buscando el beneficio del proyecto y del

equipo. Andrés ha sido un líder eficiente y ha fomentado un ambiente colaborativo y productivo entre los miembros del equipo.

- Líder de datos: Josue Rivera

El líder de datos en el proyecto se llama Josue Rivera. Su función principal es manejar y organizar los datos que se utilizarán en el proyecto, asegurándose de que estén disponibles para todos los miembros del equipo. Él también es responsable de asignar tareas relacionadas con el manejo y análisis de datos, y de asegurarse de que se completen dentro de los plazos establecidos. Josue trabaja en estrecha colaboración con el líder del proyecto y otros miembros del equipo para garantizar que los datos sean precisos, completos y relevantes para el proyecto. Además, se asegura de que se cumplan los estándares de privacidad y seguridad de los datos en todo momento.

- Líder de negocio: Jaime Alfonso

Jaime es el líder de negocio en el proyecto. Se asegura de que el problema o la oportunidad identificada se resuelva de manera efectiva y que esté alineado con la estrategia de negocio para el cual se está llevando a cabo el proyecto. También es responsable de garantizar que el producto se pueda comunicar adecuadamente y que se cumplan los objetivos del proyecto. Jaime tiene el papel crucial de contactar al grupo de expertos en estadística para evaluar el modelo, tanto a nivel cuantitativo como cualitativo, para garantizar la calidad del producto final. Además, trabaja en estrecha colaboración con el equipo técnico y el líder de proyecto para asegurarse de que el proyecto esté avanzando de acuerdo con los objetivos establecidos.

- Líder de Analítica: Josue Rivera

Josué es el líder de analítica del equipo, su responsabilidad es gestionar las tareas de análisis de datos y asegurarse de que los entregables cumplan con los estándares requeridos. Josué trabaja en conjunto con los otros líderes del equipo para identificar las mejores prácticas de análisis y asegurar que se estén aplicando correctamente en el proyecto. Además, Josué es el encargado de verificar que el equipo esté utilizando las herramientas y metodologías adecuadas para analizar los datos y encontrar la solución más efectiva al problema planteado.

- Pareja Asignada de Estadística: Valery Calixto

Valery, es la persona de estadística que nos fue asignada y con la que nos pudimos comunicar y reunir en esta segunda etapa para la aclaración de dudas y retroalimentación acerca de cómo se expresaba el funcionamiento del modelo y api, nos dio buena retroalimentación con respecto a los resultados del modelo y el despliegue del mismo.