Proyecto 1 – Parte 2 – BI

María Paula González Escallón
Jessica A. Robles Moreno
Juan Esteban Vergara

Tabla de contenido

1. El equipo

Nuestro equipo está conformado por tres integrantes:

- María Paula González Escallón:
  - María Paula es la encargada liderar el proyecto. Se encarga de manejar las preentregas de cada miembro del equipo, asignar tareas y de resolver conflictos. También hace parte del equipo de ingenieros de software encargados de la aplicación final, ella hizo parte del proceso de implementación de la aplicación, su interfaz y su integración con el modelo analítico escogido en la primera parte del proyecto.
  - Las horas de trabajo fueron 10.
  - Debido a su colaboración en el equipo y a las tareas realizadas se ha decidido que a María Paula se le asignan 33.33 de los 100 puntos en total (100/3).

- Jessica A. Robles Moreno:
  - Jessica es la encargada de liderar el manejo de datos. Ella se encargó de manejar los datos necesarios para la creación del pipeline y la encargada de mejorar las métricas del algoritmo seleccionado. También hace parte del equipo de ingenieros de software encargados de la aplicación final, ella hizo parte del proceso de implementación de la aplicación, su interfaz y su integración con el modelo analítico escogido en la primera parte del proyecto.
  - Las horas de trabajo fueron 10.
  - Debido a su colaboración en el equipo y a las tareas realizadas se ha decidido que a Jessica se le asignan 33.33 de los 100 puntos en total (100/3).

- Juan Esteban Vergara
    - Juan es el ingeniero encargado del diseño y los resultados de la aplicación. Él se encargó de diseñar la aplicación, sus partes funcionales y aspecto visual. También se encargó de la realización del video y sus resultados.
    - Las horas trabajadas fueron 10.
    - Debido a su colaboración en el equipo y a las tareas realizadas se ha decidido que a Juan Esteban se le asignan 33.33 de los 100 puntos en total (100/3).

Nuestro equipo realizó las siguientes reuniones:

- Reunión de lanzamiento y planeación: Esta reunión se llevó a cabo de manera presencial el día 11 de noviembre del 2022. En esta reunión se definieron los roles de cada integrante, se leyó el enunciado del proyecto y se realizó la lluvia de ideas para tener la base de cómo empezar el proyecto.
- Reunión de seguimiento: Esta reunión se llevó a cabo a través de zoom el día 12 de noviembre del 2022. En esta reunión, por un lado, se implementó una interfaz, la cual fue diseñada teniendo en mente el usuario objetivo. Además, por el otro lado se realizó un pipeline que fue integrado con el modelo analítico de la primera parte del proyecto 1.
- Reunión de finalización: Esta reunión se llevó a cabo a través de zoom el día 13 de noviembre del 2022. En esta reunión, en primer lugar, se terminó el documento solicitado. Al finalizar la reunión, se verificó que todos los puntos solicitados en el enunciado estén implementados y subidos al repositorio, con enlace en la wiki, con el fin de realizar la entrega pertinente en Bloque Neón.

2. Comprensión del negocio y enfoque analítico

Descripción de usuario/rol de la organización	La aplicación diseñada para este proyecto será usada por los miembros directivos de la rama de recursos en línea y nuevas tecnologías de la organización “Suicide Awareness Voices of Education” (SAVE) bajo la gerencia de Jennifer Owens (Program Manager). La aplicación ira a la par con las nuevas tecnologías creadas alrededor de la prevención de suicidio en redes sociales como herramienta que soporta la línea de acción para la prevención del suicidio.
Conexión entre aplicación y modelo de negocio	SAVE es una organización que se enfoca en las practicas activas para la prevención del suicidio a través de la sensibilización publica, educación y equipamiento para salvar vidas. Una buena cantidad de recursos del negocio van orientados al soporte virtual para las personas que requieren ayuda

	inmediata, la aplicación desarrollada brinda capacidad de reacción al negocio a casos tempranos o sin identificar de una forma activa (En vez de esperar a que las personas busquen ayuda, se pueden encontrar a las personas que lo necesitan) siendo potencialmente una pieza clave en el negocio captando la población objetivo y ayudándola de forma más temprana.
Importancia de la aplicación para el rol que la usa	Los miembros encargados de los recursos tecnológicos de la organización ya poseen un soporte para redes sociales para prevenir intentos de suicidio (preventtheattempt.com) pero es para una captación pasiva de casos de ayuda, requieren que la persona afectada busque la ayuda dentro de las redes sociales. Pero los miembros encargados con el uso de la aplicación que creamos pueden pasar a una captación activa de casos en las redes sociales, añadiendo el uso de la aplicación implementada con el pipeline entrenado a la cadena de proceso de ayuda como herramienta inicial.

3. Entendimiento y preparación de los datos

Los datos que nos brinda la organización para generar el modelo es un csv con tres columnas y 195.700 registros. La primera columna hace referencia al número asociado al texto que no nos interesa para hacer nuestro modelo ya que no aporta información relevante. La segunda columna (Text) hace referencia al comentario hecho por una persona en la comunidad de Reddit. Finalmente, la última columna (class) asocia el comentario de Reddit a dos etiquetas, non-suicide y suicide, haciendo referencia a si el comentario demuestra o no comportamientos o intenciones suicidas.

Al revisar la calidad de estos datos nos encontramos con que ningún registro es nulo, por lo tanto, no es necesario hacer ajustes ni eliminar registros nulos. También al revisar la cantidad de etiquetas de cada clase observamos que el número de textos etiquetados como ‘non-suicide’ es de 110165, y los etiquetados con ‘suicide’ son 85535. Pese a que los datos tienden a tener más registros con textos con tendencias no suicidas, la cantidad de etiquetas suicidas esta medianamente balanceada con la otra clase, por ende, no es necesario generar registros con SMOTE.

Sin embargo, se encontró que en la columna textos hay varios comentarios que cuentan con caracteres especiales como enter, signos de puntuación y

espacios. Para esto se decidió hacer una limpieza de esta columna eliminando todos estos caracteres especiales y de paso convirtiendo todas las letras a minúscula con el fin de homogenizar el texto y que el algoritmo no clasifique erróneamente palabras con significado igual por un mal entrenamiento con palabras minúsculas y mayúsculas.

Otra observación sobre los datos administrados por la organización es que hay muchos comentarios con palabras repetidas, las cuales al momento de tokenizar se ignoran. Esto debido a que para nuestros modelos se generan predicciones a partir de las palabras encontradas y su significado, no por la cantidad de veces que estas aparezcan en el comentario, ya que puede generar problemas en el algoritmo al momento de identificar que comentarios son suicidas y cuales no lo son.

Nuestros algoritmos implementaron el método CountVectorizer como método para volver las palabras números y que sean mejor entendidas por los algoritmos. Lo que hace esta función es crear vectores donde cada posición representa la cantidad de repeticiones tienen cada palabra de nuestro vocabulario. En nuestro caso debido a la limpieza hecha previamente, estos vectores van a estar rellenos de números binarios.

4. Desarrollo del modelo y la aplicación

Con el desarrollo y los resultados obtenidos en la anterior entrega de este proyecto se concluyó que, entre los 3 algoritmos, LinearSVC, Naive Bayes y SGD Classifier, aquel que arroja los mejores resultados para el objetivo del proyecto es LinearSVC. Debido a esto, se escogió trabajar con este algoritmo al momento de desarrollar la aplicación.

Un desafío que se encontró al momento de exportar el modelo escogido es que anteriormente se pasaba por parámetro a la función CountVectorizer una función para la tokenización. Sin embargo, al exportar este modelo y usarlo en otro archivo, este no podía utilizar la función que se pasaba por parámetro. Es por esto que decidimos eliminarla y en vez de esto usar el tokenizador que viene por defecto en CountVectorizer.

Con respecto al modelo y los datos implementados no se hicieron muchos cambios. El primer cambio que se hizo fue incorporar el uso de pipelines al modelo. Se hizo con el objetivo de poder manejar el modelo de una manera más sencilla para poder exportarlo y trabajar con este desde la aplicación. Para la creación del pipeline se realizaron 3 pasos:

1. Se utiliza CountVectorizer para agrupar los datos en una matriz donde cada posición representa la cantidad de repeticiones de una palabra. Esta función por defecto separa los datos a partir de espacios en blanco, vuelve todos los caracteres a minúsculas y elimina los signos de puntuación y los caracteres especiales.
2. Después se utiliza la función TfidfTransformer, que lo que hace es volver una matriz a una representación tf-idf. Esta aumenta matriz representa la multiplicación del número de veces que esta la palabra en un comentario por

el inverso de en cuantos documentos. Esto se ve representado en la siguiente ecuación:

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

Al hacer esto se está representando la importancia de una palabra en los comentarios, de tal manera que si la palabra se repite muchas veces en muchos comentarios se asumen que no es una palabra importante para el sentido de este. Sin embargo, si este se repite muchas veces en un comentario, pero aparece en pocos comentarios se interpreta como una palabra importante para el significado del contexto. (Dutta, 2021)

3. Por último, se realiza la ejecución del modelo escogido. En nuestro caso este es LinearSVC (Linear Support Vector Classifier). Lo que hace este algoritmo es implementar un kernel lineal para encontrar el mejor hiperplano que divide y categoriza los datos dados en el entrenamiento. De esta manera, cuando ingresen datos sin clasificar ya tendrá una referencia de como categorizar los nuevos datos. (sklearn.svm.LinearSVC, s. f.)

   Al momento de implementar el algoritmo utilizamos los parámetros que vienen por defecto. Entre estos se debe resaltar loss='squared_hinge', max_iter=10000 y penalty='l2'. Squared_hinge representa la información utilizada para la función loss en el algoritmo. 'l2' representa la penalización, y esta es utilizada ya que la mayoría de nuestros datos no son dispersos y queremos que la penalización sea constante. De manera similar, se utiliza squared_hinge ya que este es utilizado con l2 (en datos no dispersos). Aumentamos el número de iteraciones debido a que debido a la cantidad de datos se necesitan más iteraciones para obtener resultados más exactos.

Para hacer el entrenamiento y la evaluación del algoritmo se separaron la totalidad de los datos dados por la empresa como contexto. Ya que a pesar de que es una cantidad elevada de datos, nuestro modelo no tenía demoras de más de 1 minuto para ejecutarlo y consideramos que se obtienen resultados más acordes a la realidad al manejar la mayor cantidad de datos posibles.

Después de haber entrenado el modelo este se exporta como un archivo .joblib para poder ser utilizado al momento de ejecutar la aplicación.

Para desarrollar la aplicación se utilizó el web framework de alto nivel de Django, que permite el desarrollo de páginas web escalables, seguras y sostenibles. Se realizó en Django ya que permite una implementación sencilla y en el lenguaje Python. Este lenguaje se nos facilita para este proyecto ya que el modelo de clasificación y el pipeline fueron hechos con este también, y el manejo del pipeline desde la aplicación se facilita al tener acceso a la librería para el manejo de pipelines (joblib).

Para la implementación de la aplicación se realizó una aplicación web que contiene la interfaz y la lógica necesaria para la organización. Se realizó una interfaz simple donde se puede observar las instrucciones para la predicción de textos y una

sección para ingresar comentarios y posteriormente ser clasificados como suicidas o no suicidas.

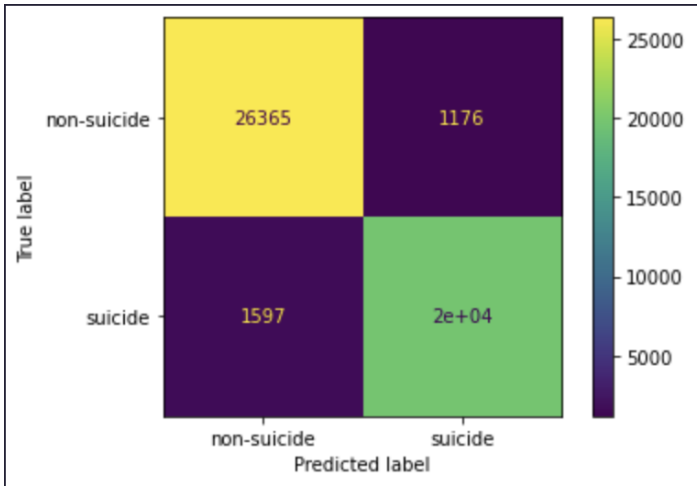
Para la interfaz gráfica se hizo uso de html y bootstrap, que nos facilita widgets para una implementación interactiva y sencilla de la interfaz. Para la persistencia de los datos se utilizó la base de datos por defecto que viene incorporada con los proyectos creados en Django, sqlite3. Por último, para la lógica y la interacción entre las aplicaciones dentro de la página web se hizo uso de modelos de Django para la conexión entre la persistencia y la lógica de la aplicación, se hizo uso de los archivos url.py y view.py para la conexión entre funciones lógicas y redireccionamiento entre las aplicaciones y sus páginas web. Por último, la lógica de la aplicación se encuentra en el archivo logic_inputs.py. Donde se hacen las funciones de crear(create) y solicitar (get).

Se decidió que la predicción de los textos se iba a hacer antes de guardar los textos en la base de datos. Esto se hace para que al guardar los textos se puedan asociar a un resultado/clasificación al momento de ser guardados en la base de datos y poder mantener un registro de todas las predicciones hechas y sus resultados.

Para hacer la predicción fue necesario hacer uso de un API, esta se hizo con la ayuda de la librería fastAPI. Desde el archivo de lógica (logic_inputs.py) se hace envío de la petición a la API llamando a la función make_predictions ubicada en el archivo apiMain.py. Ya en esta función lo que se hace es leer la petición y obtener el texto que se quiere predecir. La clasificación de textos se hizo importando el pipeline con el modelo LinearSVC y haciendo la predicción sobre los textos ingresados por parámetro con ayuda de la función predict(). Ya por último se envían los resultados de la petición en formato json al archivo logic_inputs.py. Posteriormente este obtiene el resultado recibido y se hace la persistencia del texto y su clasificación para que el usuario pueda observar los resultados de la predicción y poder analizar si un comentario tiene tendencias suicidas o no. Esto puede ser utilizado para identificar si una persona está teniendo problemas de salud mental y poder actuar a tiempo para salvar vidas.

5. Resultados

Después de haber creado el pipeline y haber hecho fit sobre todos los datos de prueba lo que se hizo fue encontrar las métricas de clasificación del algoritmo. Primero se encontró que el “accuracy score” sobre los datos de entrenamiento fue 97.3429% y sobre los datos de prueba fue 94.3321%.



	precision	recall	f1-score	support
non-suicide	0.94	0.96	0.95	27541
suicide	0.94	0.93	0.93	21384
accuracy			0.94	48925
macro avg	0.94	0.94	0.94	48925
weighted avg	0.94	0.94	0.94	48925

Exactitud: 0.9433

Con respecto a la precision (VP/(VP+FP)) se puede observar que el porcentaje de casos positivos detectados es del 94% y el de negativos es de 94%. Esto nos indica que el algoritmo identifica de igual manera los casos que pueden representar comportamientos suicidas y los no suicidas.

Con recall (VP/(VP+FN)) los resultados nos informan la proporción de casos positivos que fueron correctamente identificados. En este caso para suicide es del 93% y para non-suicide del 96%, demostrando mejores resultados al identificar textos que no representan comportamientos suicidas. Debido a que el objetivo de la empresa es determinar los casos suicidas se recomienda aumenta el número de muestras de comentarios suicidas para que al momento de entrenar el algoritmo este entienda mejor en qué casos se puede clasificar un texto como suicida.

En general podemos observar que este algoritmo identifica mejor los casos no suicidas que los suicidas a partir del f1-score, ya que para los no suicidas es del 95% y para los suicidas es del 93%.

Con respecto a las pruebas hechas mediante el API se encontró que el modelo de predicción funciona bien en la mayoría de los casos. Sin embargo, existen ingresos de texto que pueden tender a tener comportamientos suicidas, pero no los detecta como estos

I hate myself	suicide
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa aaaaaaaaaaaaaaaaa	non-suicide
Maybe I want to die, maybe not. It depends on the day I'm having and the amount of work I have	suicide
Maybe I want to die, maybe not. It depends on the day I'm having and the amount of work I have	suicide
I love that meme, it's amazing	non-suicide
I hate my life	suicide
How do I download minions 2 free online?	non-suicide
Does someone has a gun?	suicide
hahaha	non-suicide
I cry every night before I go to bed 😔	non-suicide
eeeeeeeeeeeeeeeeeeeeeeeeoooooooooooooeooeeeeeeeeeOOOOOOOOOO	non-suicide
ello mate	non-suicide
I hate me	non-suicide
I hate myself	suicide

Se sabe que ningún modelo de predicción es perfecto, en este caso tampoco lo es. Es por esto que le recomendamos a la empresa tener en cuenta que el algoritmo puede tener errores y no basar sus decisiones totalmente en los resultados obtenidos del modelo. También otra sugerencia que les damos es que si quieren obtener mejores resultados deberían hacer un filtro de los datos con los que se hizo el entrenamiento del modelo, ya que al revisarlo se pudo observar que algunos de estos datos estaban etiquetados como no suicidas, pero al revisarlos estos pueden llegar a representar pensamientos suicidas, y como entrenamos al modelo con estos datos, si estos tienen errores el modelo va a aprender a clasificar con errores.

6. Referencias

Dutta, M. (2021, 14 julio). *Bag-of-words vs TFIDF vectorization –A Hands-on Tutorial*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/07/bag-of-words-vs-tfidf-vectorization-a-hands-on-tutorial/