PROYECTO – ETAPA 1

Juan José Córdoba: 201922105

Andrés Peña: 201913766

Laura Isabela Martínez Galindo: 202012613

Temática: Salud Mental

Problema de negocio:

El tabú alrededor del suicidio o los problemas relacionados con la salud mental hace que para las personas con estos problemas sea más difícil hablarlo. Con esto en mente, muchas de estas personas utilizan las redes sociales para expresar estos sentimientos. Reddit, una red social de foros, contiene una gran variedad de comunidades que comentan este tipo de situaciones y en algunos casos sus participantes sufren de depresión o han intentado suicidarse.

Enfoque analítico:

Una herramienta muy útil para predecir cuales usuarios tienen depresión o han intentado suicidarse es el Machine Learning. En este caso se van a utilizar técnicas de Analítica de Textos (AT) para poder analizar comentarios de usuarios en Reddit que cumplen con estas características para poder entrenar un modelo que prediga si una persona, dependiendo de sus comentarios en esta red social, padece de alguna de estas enfermedades.

Roles:

- *Líder de proyecto:* Juan José Córdoba
- *Líder de negocio:* Laura Isabela Martínez
- *Líder de datos y analítica:* Andrés Peña

Reuniones de equipo:

Reunión de lanzamiento y planeación: 10 oct 2022 (35 mins)

El día 10 de octubre nos reunimos para leer el enunciado del proyecto y poder definir los roles y las tareas que iba a realizar cada integrante en el grupo. Con la asignación de los roles nos definimos las tareas que se van a realizar en el proyecto.

Reunión de ideación: 12 oct 2022 (2 hrs)

En esta reunión nos pusimos de acuerdo al explorar los datos de Salud mental, para poder darle una interpretación correcta al problema del negocio y ayudar al Líder de datos y analítica a completar una de las primeras partes del proyecto que es el "Entendimiento y preparación de los datos"

Reunión de seguimiento: 12 oct 2022 (1 hr)

En esta reunión solucionamos un bug que se presentó y que estaba siendo bloqueante a la hora de perfilar los datos que vamos a utilizar para los algoritmos de

ML. Por esta razón, la reunión duró tanto y se utilizó también para ver el avance en el proyecto.

Reunión de seguimiento 2: 14 oct 2022 (20 mins)

Fue una reunión para poder establecer el avance de cada integrante en la decisión que se había tomado para la implementación de los algoritmos a realizar.

Preparación y entendimiento de los datos:

Los datos, dado el tema que escogímos están compuestos por una estructura de 3 columnas las cuales son: Unnamed 0, que representa el ID del registro; text, que representa el texto a analizar; class, que representa la categoría asignada al texto que puede estar entre (Suicide, Non-suicide).

El texto y la categoría está en el idioma de ingles

Pasos para la preparación de los datos:

1. Stopwords: Definimos las stopwords (palabras que no tienen un significado) para el lenguaje de inglés ya que este es el lenguaje que, en este caso, usan los textos dentro del dataframe.

Para realizar este procedimiento, debemos importar previamente la librería nltk. Esta librería nos provee funciones para trabajar con data del lenguaje humano.

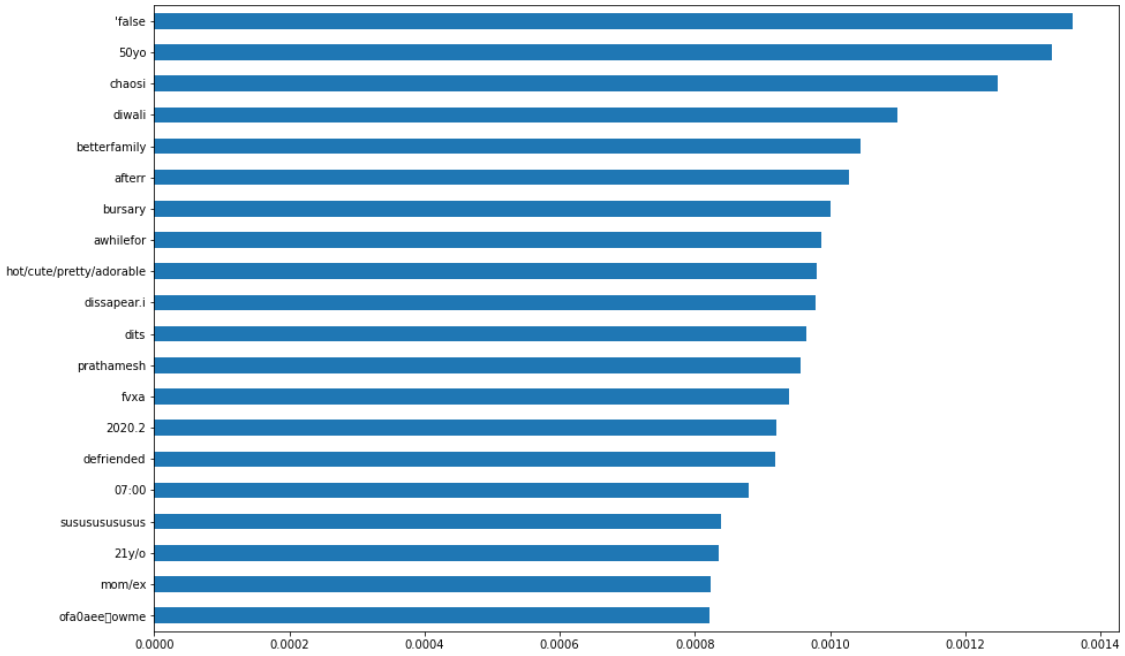
Después, descargamos el paquete punk y las stopwords para luego guardar en una variable las stopwords en el lenguaje de inglés.

2. *Tokenizer:* Para la tokenización, definimos una función llamada tokenizer y usamos word_tokenize, la cual es una clase que nos ofrece NLTK para poder dividir las palabras en tokens que conforman un texto.
3. *Lectura de datos:* Usamos la librería pandas para leer los datos que están en formato csv.
4. *Entendimiento de los datos:* Usamos la función data.shape para saber el número de filas y columnas que tiene el dataset. También usamos el data.columns para saber los nombres de las columnas y su tipo de dato.
5. *Eliminar columnas:* Decidimos eliminar la primera columna (Unnamed: 0) ya que no genera ningún valor al modelo que vamos a realizar.
6. *Identificar la clasificación de la columna "class":* Aquí queremos ver el porcentaje que hay en el dataset de las dos categorias de la columna 'class', es decir, de non-suicide y suicide.
7. *Selección del modelo para vectorizar:* Seleccionamos el modelo TfidfVectorizer ya que en este caso tiene menor costo y es más rápido, en comparación con el modelo de BoW. No obstante, se implementaron ambos y y se hicieron pruebas, que luego fueron eliminados porque los resultados de los modelos entrenados eran levemente inferiores a los utilizados con TF-IDF

Nota: Recordando que el vectorizer es una estructura que nos permite tener un índice más organizado de los tokens que se encontraron en las frases y su ocurrencia en el dataset y los diferentes registros. Permitiendo al modelo hacer un entrenamiento para encontrar patrones y poder clasificar un texto.

8. *Separar en datasets:* Dividimos los datos en Train y Test dataset.
9. *Uso del TfidfVectorizer:* Aplicamos el dataset Train al TfidfVectorizer.
10. *Verificación de la longitud del vocabulario:* Obtenemos la longitud del vocabulario del dataset usando en parametro max_df en la clase TfidfVectorizer ya que este reduce la cantidad de veces que una palabra aparece en el dataset no usándola como un token.

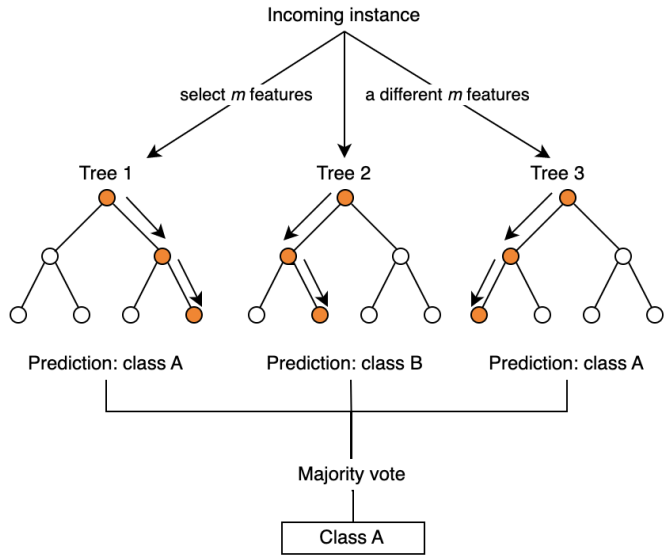
Feature importances:



Técnicas y algoritmos a utilizar:

- *Random Forest:* Andrés Peña

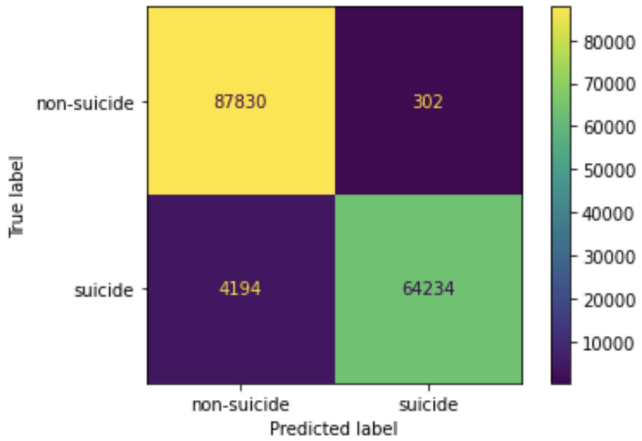
El objetivo de este algoritmo es crear varios árboles de decisión pequeños (estimadores), y combinar sus predicciones para obtener una predicción más acertada.



1. Importar Random Forest Classifier de sklearn.ensemble
2. Crear el objeto de RandomForestClassifier con un random_state de 3. A este random_state se le asigna un valor para asegurar que la aleatoriedad no afecte la precisión del modelo.
3. Entrenar el modelo usando los training sets.
4. Predecir la respuesta para el train y test dataset.
5. Imprimir las matrices de confusión y los scores.

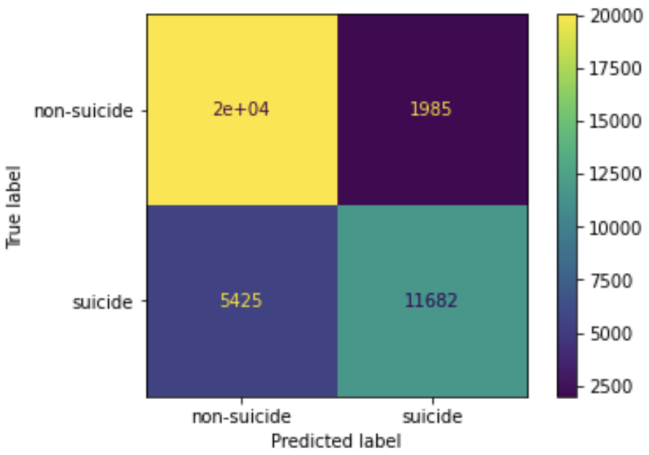
NOTA: Para poder entrenar el RandomForest se hizo uso de varías implementaciones de un GridSearchCV el cual en todas y cada uno de sus resultados arrojó hiperparámetros que empeoraban el módelo con relación a los hiperparámetros, mencionados anteriormente. Adicionalmente, con el objetivo de reducir complejidad y tiempo de la búsqueda de hiperparámetros, se redujó la cantidad de datos a ¼ del dataset (50K aprox).

Train



La mayoría de los datos fueron clasificados correctamente, como podemos ver en la matriz, muy poca cantidad de ellos fueron falsos negativos (4194) y falsos positivos (302).

Test



Aquí podemos observar que hay mayor cantidad de datos categorizados incorrectamente, comparando con la matriz del train.

Scores:

Train

Precision: 0.9953204413040784

Recall: 0.9387093002864324

F1: 0.9661863361511387

Test

Precision: 0.8547596400087802

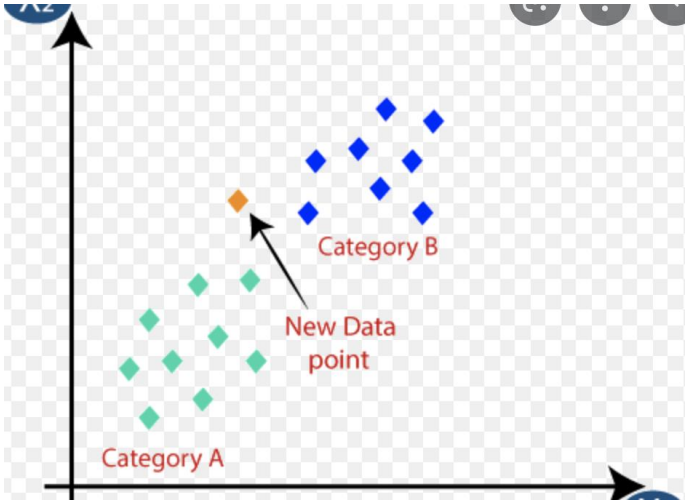
Recall: 0.6828783538902203

F1: 0.759212322090076

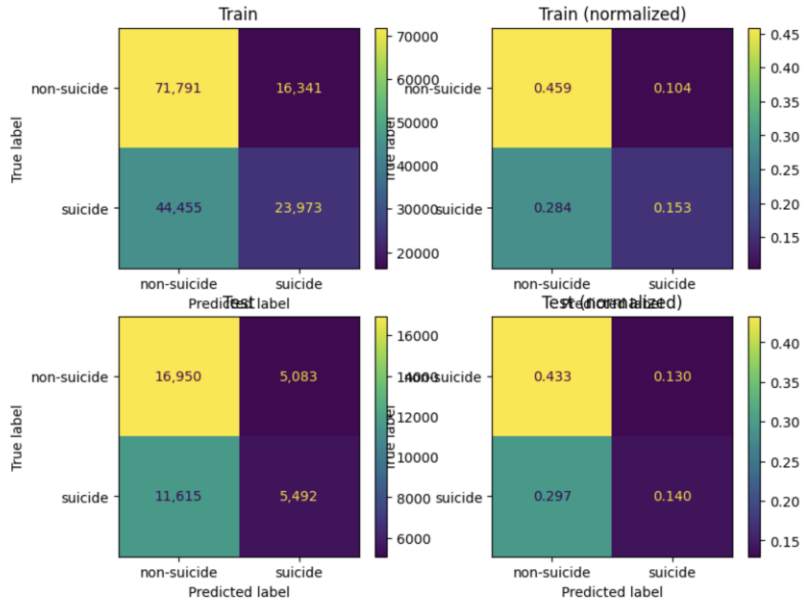
Se escogió porque este algoritmo es uno de los más adecuados para manejar datos con ruido de alta dimensión en Text Classification. Si lo comparamos con el Árbol de Decisión, este reduce el overfitting en los estimadores y ayuda a mejorar la precisión.

- *K-Nearest Neighbors (KNN):* Juan José Córdoba

La función es identificar los vecinos más cercanos de un punto dado para poder categorizar ese punto.



1. Importar de sklearn.neighbors, KNeighborsClassifier
2. Crear el objeto KNeighborsClassifier con n_neighbors = 5. Este es la cantidad de puntos vecinos a usar, 5 es el número por defecto.
3. Ajustar el modelo a los datos de entrenamiento.
4. Predecir la respuesta para el train y test dataset.
5. Precision de train y test dataset
6. Impresión de las matrices de confusión de las predicciones realizadas.



A partir de estas matrices podemos analizar que en el train hubo más falsos negativos que verdaderos positivos.

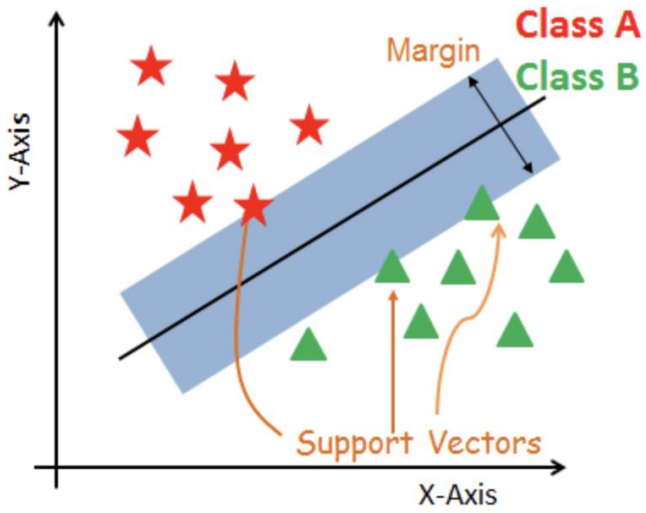
Por otro lado, en el test hubo más falsos negativos y falsos positivos que verdaderos positivos.

Se escogió este algoritmo ya que está entre los recomendados de implementar cuando se está trabajando con Text Classification. Además, la fase entrenamiento es

rápida ya que KNN tiene lazy learning, es decir, no se aprende una función en especial del training data sino que, se memoriza el training dataset.

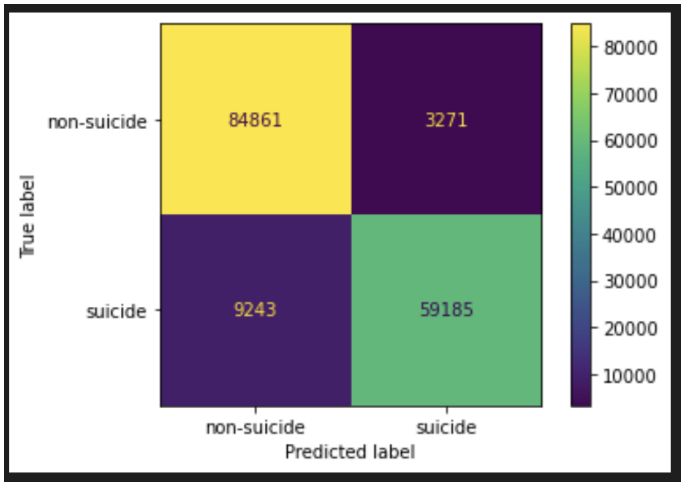
- *Support Vector Machines (SVM):* Laura Isabela Martínez

El propósito de este algoritmo es construir de manera iterativa, un hiperplano óptimo que separe el dataset en diferentes clases.



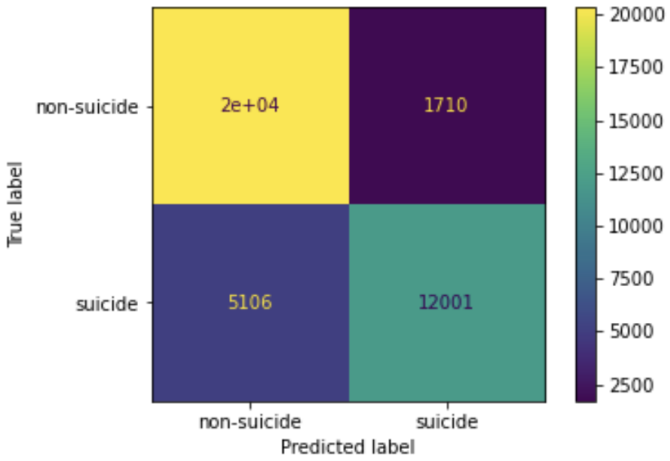
1. Importar el módulo de Support Vector Machines
2. Crear el objeto de Support vector llamando a Support Vector Classification que pertenece al módulo de Support Vector Machines. Y como parámetro, un kernel lineal ya que este se usa cuando hay una gran cantidad de features en el data set, en este caso como está trabajando con Text classification, lo más conveniente es usar el kernel lineal.
3. Entrenar el modelo usando los training sets.
4. Predecir la respuesta para el train y test dataset.
5. Imprimir las matrices de confusión y los scores.

Train



A partir de esta matriz de confusión, podemos concluir que gran cantidad de los datos fueron categorizados correctamente comparando con el número de falsos positivos y verdaderos negativos.

Test



Igualmente, en el test hubo mayor cantidad de datos que se clasificaron correctamente.

Scores

Train

Precision: 0.9476271294991674

Recall: 0.8649237154381247

F1: 0.9043886189297394

Test

Precision: 0.8752826197943258

Recall: 0.7015256912375051

F1: 0.7788305535725875

Se escogió este algoritmo ya que es uno de los mejores y más usados para Text Classification. Esto es porque el aprendizaje de SVM es independiente de la dimensionalidad del espacio de las features. También usa protección de overfitting, lo cual le da el potencial para trabajar con grandes cantidades de features.

Conclusiones

- Al comparar los resultados de los tres modelos anteriores podemos darnos cuenta de que el más acertado es Support Vector Machines ya que su F1 fue el más alto, es decir, tiene mayor precisión y recall que los demás algoritmos.
- Este modelo puede que sea el más acertado entre los tres ya que, como anteriormente lo mencionábamos, es uno de los mejores algoritmos para usar cuando estamos trabajando con Text Classification.

- El peor modelo fue KNN ya que este algoritmo no es eficaz al tratar con grandes cantidades datos, esto se debe a que el costo de calcular la distancia de un punto a cada punto vecino es muy grande. Además, es sensible al ruido en los datos.

Conclusiones del negocio:

Los modelos entrenados pueden ser una gran ayuda para el negocio, teniendo en cuenta que los resultados son altos y son bastante efectivos a la hora de predecir. Adicionalmente, si bien las predicciones no son un 100% exactas, para el negocio resulta ser una herramienta muy útil, debido a que se podría utilizar para tratar de prevenir suicidas, en el sentido de que una equivocación en la predicción del modelo no resulta en alguna acción grave a realizar. Es decir, si tu le ofreces acompañamiento a una persona que fue predicha como un posible mensaje de “Suicide” no le va a afectar en nada.

Ahora bien, es necesario llegar a una mayor eficiencia en el modelo para poder evitar tener datos categorizados como “non-suicide” y que realmente sean “suicide” esto se puede lograr mediante un entrenamiento constante del modelo y una revisión de los datos categorizados, debido a que en nuestro entendimiento de los datos consideramos que hay varios textos que estarían mal clasificados por la posible aparición de una palabra que puede ser categorizada como “mala”

Referencias

- Navlani, A. (2019, diciembre). *Support Vector Machines with Scikit-learn Tutorial*. DataCamp. Recuperado 19 de octubre, de https://www.datacamp.com/tutorial/svm-classification-scikit-learn-python
- ProjectPro. (2022, 27 junio). *Machine Learning NLP Text Classification Algorithms and Models*. Recuperado 19 de octubre de 2022, de https://www.projectpro.io/article/machine-learning-nlp-text-classification-algorithms-and-models/523
- Great Learning Team. (2022, 9 septiembre). *Random forest Algorithm in Machine learning | Great Learning*. Great Learning Blog: Free Resources what Matters to shape your Career! Recuperado 19 de octubre de 2022, de https://www.mygreatlearning.com/blog/random-forest-algorithm/
- Techopedia. (2017, 14 marzo). *K-Nearest Neighbor (K-NN)*. Techopedia.com. Recuperado 19 de octubre de 2022, de https://www.techopedia.com/definition/32066/k-nearest-neighbor-k-nn