

**Proyecto analítica de textos en contexto de salud mental - Entrega 1**

Santiago Pardo Bravo

Juan Sebastián Pinzón Roncancio

Daniel Felipe Reyes Ramírez

Universidad de Los Andes

Inteligencia de Negocios ISIS-3301

19/10/2022

## Índice

<b>Comprensión de negocio y enfoque analítico</b>	<b>2</b>
<b>Entendimiento y preparación de los datos</b>	<b>3</b>
Stopwords	3
TweetTokenizer	4
OneHotEncoding	4
BOW (CountVectorizer)	5
TF-IDF (TfidfVectorizer)	5
<b>Modelado y evaluación</b>	<b>6</b>
Random Forest	6
Linear SVC	7
Nearest Centroid	9
<b>Resultados</b>	<b>10</b>
<b>Referencias</b>	<b>12</b>

## Comprensión de negocio y enfoque analítico

<p>Oportunidad/problema Negocio</p>	<p>Las EPS con servicios de apoyo psicológico podrían ayudarse de la analítica de los textos extraídos de los Tweets con el fin de predecir en los comentarios que realizan sus pacientes posibles casos de suicidio y optimizar así el proceso de asignación de citas.</p> <p>En un primer momento, entrenaremos el modelo de ML con los comentarios extraídos de Twitter para luego en ejecución sea capaz de detectar y predecir cuando una frase o comentario puede ser etiquetado como posible caso de suicidio. De esa manera, se anticipará este problema con anterioridad y se le puede dar una solución de manera más eficiente y rápida.</p> <p>Lo que lograríamos con el proceso anterior, es que podríamos utilizar dicho modelo ML de predicción para priorizar los pacientes que tengan mayor probabilidad de terminar en suicidio al momento de la asignación de citas médicas de la siguiente forma: Los pacientes que durante la consulta estipulen con mayor frecuencia frases que el modelo etiquete como caso de suicidio se les prioriza la asignación de su cita. En ese sentido, la EPS puede reducir costos al tratar oportunamente casos graves relacionados a los causantes del suicidio.</p>
<p>Enfoque analítico (Descripción del requerimiento desde el punto de vista de aprendizaje de máquina)</p>	<p>El principal requerimiento de la oportunidad/problema del negocio es, según las frases y comentarios que dicen los pacientes durante sus consultas en psicología, predecir si es un posible caso o no de suicidio. Y, de esa manera, realizar la asignación de citas lo más óptima posible priorizando aquellos casos que sean etiquetados como suicidio.</p> <p>Para ello, el modelo de ML requiere en sus métricas principalmente un valor alto para la exactitud (Accuracy) y el Recall -pues se debe priorizar la detección de pacientes con intenciones de suicidio-. De esta forma nos aseguramos que el</p>

	porcentaje de predicciones correctas sea el mayor, ya que eso permitirá una efectiva optimización en el proceso de asignación de citas.			
Organización y rol dentro de ella que se beneficia con la oportunidad definida	La organización que principalmente se beneficia del modelo de ML es la EPS, específicamente aquellas que atienden casos de salud mental relacionados a enfermedades mentales que conlleven al suicidio. Por otro lado, el rol dentro de dicha organización que se beneficiaría principalmente sería el área financiera, ya que si se logra optimizar el proceso de asignación de citas, se puede ayudar a solucionar oportunamente los casos más graves y así reducir los costos en los que podría incurrir una EPS al dejar un caso agravarse más.			
Técnicas y algoritmos a utilizar	- Técnica para vectorización: BOW & TF-IDF	-Algoritmo 1: Random Forest	-Algoritmo 2: Linear SVC	-Algoritmo 3: Nearest Centroid Classifier

### Entendimiento y preparación de los datos

Para la preparación de los datos utilizamos los siguientes paquetes de la librería *NLTK*: *Stopwords*, *TweetTokenizer*. Por otro lado, para la columna categórica ‘class’ realizamos un *OneHotEncoding*. Finalmente, para la vectorización de los datos usamos los siguientes paquetes de la librería *Scikit-learn*: *CountVectorizer* y *TfidfVectorizer*. A continuación entraremos a revisar el uso de cada uno.

#### Stopwords

Uno de los principales pasos para preparar los datos para su uso en modelos de predicción de ML es filtrar aquellos tokens (palabras o símbolos) que podrían ser relevantes, de aquellos que son usados comúnmente y no representan información

## PROYECTO ANALÍTICA DE TEXTOS

importante para el modelo, por lo que podrían ser ignorados. Es por eso que usamos la librería *Stopwords*, que nos permite fácilmente saber cuales son las palabras (tokens) en inglés que no aportan valor al modelo ML ya que aparecen con alta frecuencia y podrían ser ignoradas.

```
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))
```

### **TweetTokenizer**

Ya que los datos de la columna *text* vienen directamente como frases enteras, debemos dividir esas frases en partes más pequeñas llamadas tokens. Estos tokens generalmente son unidades atómicas de palabras. Este proceso se hace identificando los límites de cada palabra. Ahora bien, recordemos que el dataset fue obtenido de Twitter, por lo que las frases pueden contener emoticones o Hashtags, los cuáles ni son Stopwords ni Tokens por sí solos. En ese sentido, el paquete *TweetTokenizer* nos permite tokenizar las palabras teniendo en cuenta los hashtags como parte de ellas y los emoticones como un sólo token en vez de dividirlo por caracteres.

```
def tokenizer(text):
    tt = TweetTokenizer()
    return tt.tokenize(text)
```

### **OneHotEncoding**

Como lo mencionamos previamente, para la columna categórica *class* usamos un *OneHotEncoding* para asignarle un valor numérico a cada categoría. Le asignamos 1 para suicidio y 0 para no suicidio.

```
def text(text):  
    if('suicide'==text):  
        return 1  
    if('non-suicide'==text):  
        return 0  
    return None  
df['class'] = df['class'].apply(text)
```

### **BOW (*CountVectorizer*)**

La vectorización con BOW radica básicamente en que a cada palabra en el vocabulario  $V$  se le asigna un índice entero y posteriormente un vector de longitud  $V$  con todos en 0, para cada frase se le asigna en la posición correspondiente al índice de la palabra en el vector la frecuencia de veces con las que aparece en la frase.

Esto se realizará usando de tokenizer el TweetTokenizer que habíamos creado antes y usando las Stopwords estipuladas previamente. El proceso se realiza tanto para el conjunto  $X$  de entrenamiento como de test.

```
bow = CountVectorizer(tokenizer = tokenizer, stop_words = stop_words, lowercase =  
True)  
X_bow_train = bow.fit_transform(X_train)  
X_bow_test = bow.transform(X_test)
```

### **TF-IDF (*TfidfVectorizer*)**

La vectorización con TF-IDF es muy similar a la *BOW*. Su diferencia radica en que en esta se aplican dos ecuaciones matemáticas que relacionan su frecuencia tanto en el token que se está analizando como su frecuencia en el toda la frase. Igualmente, realiza lo mismo pero a nivel de todas las frases.

De la misma manera que en *BOW*, realizaremos la tokenización usando el *TweetTokenizer* creado previamente junto al *Stopwords* generado. Esto lo haremos tanto con el conjunto *X* de entrenamiento como de prueba.

```
tfidf = TfidfVectorizer(tokenizer = tokenizer, stop_words = stop_words, lowercase =
True)
X_tfidf_train = tfidf.fit_transform(X_train)
X_tfidf_test = tfidf.transform(X_test)
```

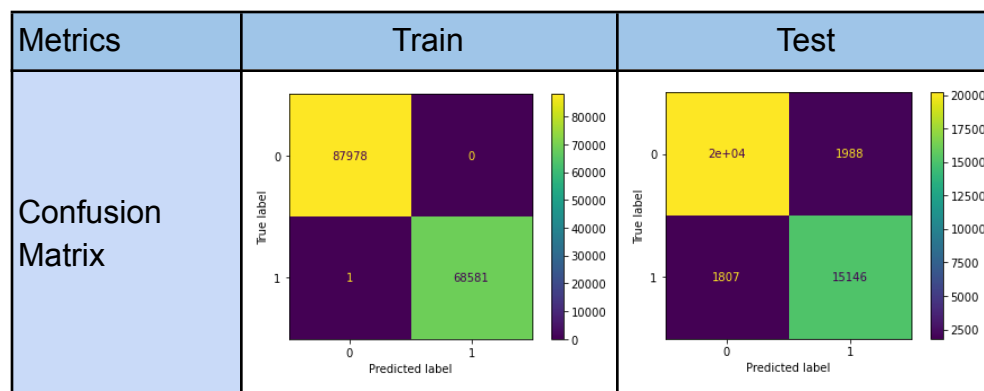
## Modelado y evaluación

Para este apartado decidimos usar los siguientes 3 algoritmos: Random Forest, Linear SVC y Nearest Centroid. Los 3 los probaremos tanto en los datos generados con BOW como con los generados por TF-IDF.

### Random Forest

Decidimos usar este primer algoritmo dado a que es más exacto que un sólo árbol de decisión, no necesita un manejo exhaustivo de hiper-parámetros y logra resolver el problema de overfitting de los árboles de decisión. De esa manera, logramos usar la lógica que presenta un árbol de decisión para entrenar el modelo sin necesidad de tener los problemas que conlleva el usarlos únicamente.

- **Evaluación BOW:**



## PROYECTO ANALÍTICA DE TEXTOS

Accuracy	99.9994%	90.3040%
Precision	100%	88.3973%
Recall	99.9985%	89.3411%
F1	99.9993%	88.8667%

- **Evaluación TF-IDF:**

Metrics	Train	Test
Confusion Matrix		
Accuracy	95.6343%	88.6331%
Precision	99.9984%	90.0512%
Recall	90.0353%	82.9175%
F1	94.7557%	86.3373%

### Linear SVC

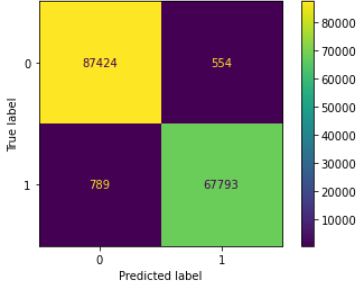
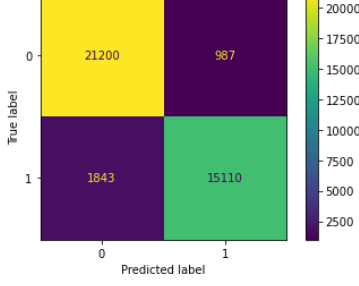
. Dado a que la data de los textos se encuentra en forma de vectores debido al proceso de vectorización BOW o TF-IDF por el que pasaron, este tipo de algoritmo podría tener un buen desempeño. Recordemos que el funcionamiento de este algoritmo básicamente es aplicar una función linear kernel para comparar dichos vectores, como aplica una penalización el algoritmo buscará que la función lineal sea lo más precisa al momento de dividir 2 o más conjuntos posibles de clasificación. Ahora bien, se hizo la variación de los hiperparametros que consideramos podrían tener buen impacto en el entrenamiento del modelo lineal. Estos fueron el parámetro “C” de



## PROYECTO ANALÍTICA DE TEXTOS

regulación, el cual en la documentación se indica que la fuerza de esta regulación es inversa al valor de C, por lo cual, como se buscaba una mayor regulación que la de defecto (para mejorar el proceso de comparación de los vectores) se disminuyó el valor hasta 0.58. Inicialmente se empezó reduciendo este valor, sin embargo, desde cierto punto, se observó que podía generar penalización y overfitting en los resultados, así pues, con la técnica de prueba y error, se estimó que 0.58 era el valor que maximiza los valores métricos del modelo. Por último, se variaron tanto las iteraciones como el valor de tolerancia para finalizar el algoritmo. Las iteraciones se aumentaron para asegurar que el modelo convergiera dada la tolerancia en todos los casos, y, teniendo en cuenta esto, la tolerancia se disminuyó para obtener una aproximación más certera.

- **Evaluación BOW:**

Metrics	Train	Test
Confusion Matrix		
Accuracy	99.1422%	92.7895%
Precision	99.1894%	93.8484%
Recall	98.8496%	89.1878%
F1	99.0192%	91.4562%

- **Evaluación TF-IDF:**

Metrics	Train	Test
---------	-------	------

Confusion Matrix		
Accuracy	97.6271%	94.5248%
Precision	97.0180%	94.5816%
Recall	97.5821%	92.6680%
F1	97.2994%	93.6150%

### Nearest Centroid

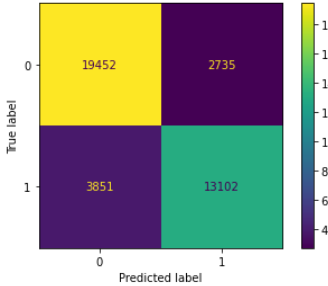
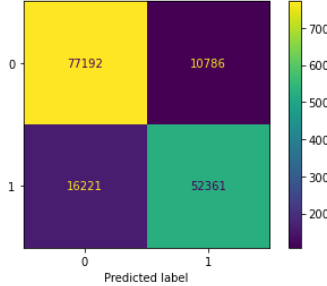
Este tercer algoritmo es interesante por la misma razón del anterior: Ya que estamos manejando datos en forma de vectores. Recordemos que el algoritmo Nearest Centroid usa la clasificación por cercanía de un centroide para un cluster. Por lo cuál, tenemos nuestro vector  $X$  con  $n$  características (features) y queremos determinar la clase de ese vector a partir de un conjunto de  $k$  clases (en este caso 2) puede ser útil y eficiente a la hora de entrenamiento, pues se comporta bien para funciones con tendencias de convexidad.

- **Evaluación BOW:**

Metrics	Train	Test
Confusion Matrix		

Accuracy	74.0342%	74.4021%
Precision	83.7058%	84.1039%
Recall	50.5687%	50.4336%
F1	63.0483%	63.0554%

- **Evaluación TF-IDF:**

Metrics	Train	Test
Confusion Matrix		
Accuracy	83.1732%	82.7497%
Precision	82.7303%	82.9192%
Recall	77.2843%	76.3480%
F1	79.9146%	79.4981%

## Resultados

Recordemos que a nivel de negocio nos importaba priorizar la métrica accuracy ya que nos permite conocer si el modelo está logrando clasificar correctamente tanto los casos positivos como los negativos de suicidio - y, de esta forma, realizar correctamente priorización de citas-. En ese sentido, después de realizar el proceso de probar los 3 algoritmos usando las distintas técnicas de vectorización (BOW y TF-IDF) podemos concluir que la mejor métrica la logró el algoritmo Linear SVC usando la

## PROYECTO ANALÍTICA DE TEXTOS

técnica de vectorización TF-IDF. Logrando con el conjunto de prueba un accuracy de 94.5248% y un F1 de 93.615%.

Ahora bien, en terminos de los objetivos de una eps de preservar la salud y vida humana, estaríamos hablando de que tambien se busca tener una metrica alta en el Recall, pues este permite definir que todos los casos que en realidad sí son suicidio, sean detectados. Así pues, el modelo escogido también presentó el mayor valor frente al resto de modelos en esta métrica, con un Recall de 92.668%.

Para el negocio esto supone unas métricas muy buenas ya que a partir de ellas las EPS se pueden apoyar muy fuertemente en la toma de decisiones para la asignación de citas, e incluso, automatizar el proceso para que un bot sea capaz de tomar la decisión de qué tan prioritaria debería ser una cita respecto a las declaraciones y frases que diga el paciente durante sus sesiones. De esa manera, lograr reducir costos atendiendo y dando una pronta solución a los casos más importantes y de mayor gravedad.

Ahora bien, respecto a las estrategias de negocio a futuro, se le recomienda al negocio la extracción de más datasets clasificados de textos con tendencias de suicidio. Lo anterior permitirá por un lado mejorar las métricas de testeo bajo las cuales se fundamenta la solidez del modelo; por el otro lado, le permitirá al modelo aumentar su colección de tokens y palabras con relevancia en los textos suicidas. Pues como bien se sabe, el lenguaje está evolucionando constantemente, por lo cual es necesario seguir entrenando el modelo con todo tipo de expresiones que surjan con el tiempo. Del mismo modo, el negocio debería emplear su propia recolección de datos a partir de la

## PROYECTO ANALÍTICA DE TEXTOS

información textual que se pueda manejar de los pacientes de la eps, estos deberían ser etiquetados e implementados en el entrenamiento también.

### Distribución del trabajo grupal

- **Líder de proyecto:** Santiago Pardo. Se encargó de distribuir la carga de trabajo de producción del código para todos los miembros del equipo. De igual forma, también distribuyó las tareas respectivas a los demás entregables, tales como el documento, el video y la presentación.
- **Líder de negocio:** Daniel Reyes. Se encargó en reunión con los demás miembros del grupo de identificar la oportunidad y problema de negocio que se podría realizar usando el dataset elegido.
- **Líder de datos:** Santiago Pardo. Se encargó de gestionar los datos que se usaron para los modelos.
- **Líder de analítica:** Juan Sebastián Pizón. Se encargó de verificar los modelos realizados y determinar la calidad de estos, junto a las pruebas de métricas realizadas y apoyo en la decisión del mejor modelo.

### Referencias

Cardellino, F. (28 de Abril de 2021). *Cómo funcionan los clasificadores Naive Bayes: con ejemplos de código de Python*. Obtenido de FreeCodeCamp: <https://www.freecodecamp.org/espanol/news/como-funcionan-los-clasificadores-naive-bayes-con-ejemplos-de-codigo-de-python/>

DataTechNotes. (01 de 07 de 2020). *Classification Example with Linear SVC in Python*. Obtenido de DataTechNotes:

<https://www.datatechnotes.com/2020/07/classification-example-with-linear-svm-in-python.html>

Gupta, M. (Abril de 2021). *Text vectorization algorithms in NLP*. Obtenido de Medium:

<https://medium.com/data-science-in-your-pocket/text-vectorization-algorithms-in-nlp-109d728b2b63>

Prince, M. (1 de Octubre de 2019). *Stop Words and Tokenization with NLTK*.

Obtenido de Medium:

<https://medium.com/@muddaprince456/stop-words-and-tokenization-with-nltk-70c4cccc5e0e>

*Python NLTK | nltk.TweetTokenizer()*. (12 de Mayo de 2022). Obtenido de Geeks for Geeks: <https://www.geeksforgeeks.org/python-nltk-nltk-tweettokenizer/>

Section.io. (11 de Diciembre de 2020). *Introduction to Random Forest in Machine Learning*. Obtenido de Section:

<https://www.section.io/engineering-education/introduction-to-random-forest-in-machine-learning/>