UNIVERSIDAD DE LOS ANDES
FACULTAD DE INGENIERIA
DEPATAMENTO DE INGENIERIA DE SISTEMAS Y COMPUTACIÓN
INTELIGENCIA DE NEGOCIOS

PROYECTO 1- ETAPA 2
Apoyo a la detección de intentos de suicidio

Grupo 4
Daniela Ricaurte - 201822966
Melissa Contreras - 202011876
Julián Mora - 202012747

Profesor
María Del Pilar Villamil

Bogotá, Octubre 19 de 2022

Tabla de Contenidos

a) Contexto de salud mental. Apoyo a la detección de intentos de suicidio a partir de información recolectada de Reddit a nivel de comunidades que sufren de depresión o han intentado suicidarse. Los datos originales los pueden encontrar en este enlace (https://www.kaggle.com/datasets/nikhileswarkomati/suicidewatch?resource=download) . Sin embargo, los datos a utilizar en el proyecto no son los mismo y los encuentra disponibles en la sección proyecto de BloqueNeón curso Unificado de BI.

0. Trabajo en equipo

Nombre	Rol	Tareas	Tiempo asignado	Tiempo dedicado
Daniela Ricaurte	- Líder de proyecto - Ingeniera de datos	- Gestión del proyecto. - Construcción del modelo analítico: Perfilamiento, limpieza, tokenización, codificación, normalización, elaboración del pipeline.	7 horas	7 horas

Julián Mora	- Ingeniero de datos - Ingeniero de Software: Diseño y resultados	Diseño de la aplicación y generación del video. - Construcción del modelo analítico: Desarrollo del algoritmo y elaboración del pipeline	7 horas	7 horas
Melissa Contreras	Ingeniera de Software: Aplicación final Diseño y resultados	Construcción y diseño de la aplicación. Implementación de la persistencia.	7 horas	7 horas

Nombre	Retos enfrentados	Formas planteadas para resolverlos
Melissa Contreras	Crear la aplicación y que pudiera devolver las predicciones en el Front.	Después de varias horas de trabajo, desarrollar un modelo efectivo de back, front y persistencia, tomando en cuenta el .joblib que elaboró Julián.
Julián Mora	Desarrollamos un pipeline para el algoritmo Naive Bayes, la mayor dificultad fue encontrar el pipeline que arrojara los resultados más adecuados.	Realizar los dos pipelines en compañía de Daniela, con tiempo de anticipación, para poder encontrar el modelo que de las predicciones más adecuadas.
Daniela Ricaurte	Desarrollamos un pipeline para el algoritmo SVM, sin embargo, es demorado y realizar correcciones se tarda.	Probar primero con menos datos, para ver si está funcionando correctamente para luego cuando se sabe que todo está bien, usar los datos completos.

Reuniones

Tema por tratar	Fecha
Reunión de lanzamiento y planeación	11 noviembre 2022
Reunión de ideación	12 noviembre 2022
Reuniones de seguimiento	13 noviembre 2022
Reunión de finalización	13 octubre 2022

Repartición de los 100 puntos

- Julián Mora: 33,33 puntos.
- Melissa Contreras: 33,33 puntos.
- Daniela Ricaurte: 33,33 puntos.

Puntos de mejora
Comenzar a hacer los trabajos con mayor antelación, e intentar resolver los problemas en conjunto para así ser más efectivos y aprendemos mucho mejor.

1. Comprensión del negocio y enfoque analítico.

Definición de los objetivos y criterios de éxito desde el punto de vista del negocio.
Determinación del enfoque analítico para alcanzar los objetivos del negocio.
Descripción de cómo el requerimiento de negocio es resuelto por el enfoque analítico propuesto, para lo cual debe diligenciar la tabla que se presenta a continuación:

Oportunidad/problema Negocio	Debido al incremento exponencial de los casos de depresión a lo largo de los años, nos vemos enfrentando una crisis de salud mental, concentrada en personas jóvenes. Esto nos presenta la oportunidad de identificar a aquellos que son más vulnerables, para brindar apoyo oportuno y evitar el suicidio.
Enfoque analítico (Descripción del requerimiento desde el punto de vista de aprendizaje de máquina)	Identificar patrones en los textos de personas que necesitan ayuda, pero no saben cómo pedirla. A través de modelos de lenguaje natural.
Organización y rol dentro de ella que se beneficia con la oportunidad definida	Cualquier empresa de redes sociales puede beneficiarse de estos modelos, ya que siempre debe de ser prioridad para estos medir el bienestar de sus usuarios.
Técnicas y algoritmos para utilizar	Máquinas de Vectores de Soporte (SVM) y como técnica de vectorización se va a usar TF-IDF Algoritmo Naive Bayes Algoritmo Label Spreading

2. Construcción del modelo analítico

Algoritmo seleccionado: Algoritmo Naive Bayes

Estudiantes encargados: Daniela Ricaurte y Julián Mora

Descripción del algoritmo:

Este algoritmo de clasificación, tal como su nombre lo indica, está basado en el teorema de probabilidad condicionada propuesto por el matemático Thomas Bayes. Entender este teorema es fundamental para comprender cómo trabaja el algoritmo, por lo cual se hará una explicación resumida.

En términos generales, este teorema vincula la probabilidad de un evento A dado un evento B, con la probabilidad de el evento B dado el evento A. Es decir, gracias a conjuntos de datos conocidos, por ejemplo, la probabilidad de A, dado B, podemos concluir que una hipótesis B sea verdadera si algún evento A ha sucedido.

La fórmula del teorema de Bayes es la siguiente, para el caso anterior:

$$P(B|A) = \frac{P(A|B) * P(B)}{P(A)}$$

Los conjuntos de datos son los siguientes:

- **P(B)** es el a priori, la probabilidad de que la hipótesis B sea verdadera antes de la evidencia.
- **P(A|B)** es el likelihood, probabilidad de que la hipótesis A sea verdadera dados los datos B.
- **P(A)** es el likelihood marginal o evidencia, la probabilidad de observar los datos A.
- **P(B|A)** es el a posteriori, la probabilidad final, en la cual B es verdadera dado A. Es la consecuencia lógica de haber usado un conjunto de datos, un likelihood y una a priori.

Resultados de la evaluación cuantitativa:

```
In [52]: y_pred = classifier.predict(Test_X)
         print(classification_report(Test_Y,y_pred))
```

```
              precision    recall  f1-score   support

  non-suicide       0.96      0.84      0.90     33024
      suicide       0.82      0.95      0.88     25686

     accuracy                           0.89     58710
    macro avg       0.89      0.90      0.89     58710
 weighted avg       0.90      0.89      0.89     58710
```

Pruebas

```
In [53]: examples = ['My name is Daniela','I want to kill myself']
         example_counts = vectorizer.transform(examples)
         predictions = classifier.predict(example_counts)
         predictions

Out[53]: array(['non-suicide', 'suicide'], dtype='<U11')
```

Ilustración 0: Evaluación cuantitativa del modelo

Justificación de la selección del modelo:

Este modelo fue seleccionado por distintas razones. En primer lugar, la aplicación que tiene el teorema de bayes a problemas donde se quiere conocer la probabilidad de las causas dados los efectos.
Por poner caso, uno de los ejemplos más conocidos sobre la aplicación del teorema de Bayes es conocer la probabilidad de que un paciente con síntomas X tenga una enfermedad Y, cuando lo que normalmente se conoce es el porcentaje de pacientes de la enfermedad Y que tienen síntomas X. Este ejemplo puede ser escalado a nuestro caso de interés pues necesitamos identificar, a partir de las actitudes que tienen los usuarios en los foros de la fuente (síntomas X), cuales de ellos pueden estar atentando contra su integridad física (enfermedad Y).
Retomando lo anterior, con el enfoque analítico una vez establecido, investigamos sobre las aplicaciones del teorema de Bayes a aprendizaje automático, con el fin de cumplir los requisitos del caso. En este punto, identificamos que a través de Scikit-Learn, librería utilizada en el curso, se pueden implementar métodos enfocados al aprendizaje supervisado basados en aplicar el teorema de Bayes. Al analizar ejemplos de su uso, notamos que es bastante común que sea aplicado a la clasificación de texto, con el objetivo de detectar spam, dadas ciertas palabras clave. De esta manera, llegamos a la idea de llevar a cabo un modelo en el cual se detecte la probabilidad de que un mensaje se deba catalogar como “riesgoso para la integridad de un individuo” o en otras palabras relacionadas al contexto “suicida”, dadas algunas palabras que se repitan y presenten indicios de esto.
Así, el modelo podría asemejarse al siguiente, para dar una idea general de cómo sería su implementación, sin especificar los pronósticos:
Para el problema de clasificación de texto, S1 se interpretaría como “Peligroso” y S2 como “No peligroso”, pues ambos eventos se excluyen mutuamente. Además, M sería la ocurrencia de alguna palabra en los textos, como por ejemplo “muerte”, así M1 indicaría que “Muerte” hace parte del texto y M2 indicaría lo contrario.

- **P(S1)** es la probabilidad a priori, que un mensaje sea peligroso sin conocer el contenido del texto.
- **P(M1|S1)** es el likelihood.
- **P(M1)** es el likelihood marginal.
- **P(S1|M1)** es la probabilidad a posteriori.

$$P(S1|M1) = \frac{P(M1|S1) * P(S1)}{P(M1)}$$

Así, la ecuación anterior representaría la probabilidad de que un mensaje deba ser catalogado como peligroso dado que la presencia de palabras como “Muerte” sea verdadero.

3. Construcción del pipeline

Para elaborar el pipeline que automatiza los pasos realizados sobre los datos, utilizamos dos algoritmos distintos para probar cómo eran las predicciones.
En primer lugar, intentamos con el algoritmo SVM, sin embargo, después de realizar el pipeline y hacer las pruebas, nos dimos cuenta de que las predicciones no eran adecuadas, así como se puede ver en la *ilustración 1*.

```
Out[62]: Pipeline(steps=[('vect', CountVectorizer()), ('tfidf', TfidfTransformer()),
                         ('clf', SVC())])

In [65]: # Clasificamos los datos recientes
         predicted = p2.predict(Test_X)

In [66]: print("SVM Accuracy Score -> ",accuracy_score(predicted, Test_Y)*100)

         SVM Accuracy Score ->  94.07426332822347

In [67]: confusion_matrix(Test_Y, predicted)

Out[67]: array([[31643,  1477],
                [ 2002, 23588]], dtype=int64)

In [68]: import matplotlib.pyplot as plt
         from sklearn.metrics import confusion_matrix, classification_report, precision_score, recall_score, f1_score, accuracy_score
         from sklearn.metrics import plot_confusion_matrix

In [70]: print(classification_report(Test_Y, predicted))

                       precision    recall  f1-score   support

                    0       0.94      0.96      0.95     33120
                    1       0.94      0.92      0.93     25590

             accuracy                           0.94     58710
            macro avg       0.94      0.94      0.94     58710
         weighted avg       0.94      0.94      0.94     58710


In [102]: oom but my walls are slowly falling down and itâ€™s only a matter of time before i snap and finally end it once and for all']

In [103]: new_output = text_clf.predict(new_input)

In [104]: new_output

Out[104]: array([0])
```

Ilustración 1: Resultados modelo SVM

Debido a las predicciones del algoritmo SVM, decidimos utilizar el algoritmo Naive Bayes, para el cual los datos ingresados por el usuario eran congruentes con la clasificación de textos dada por el Excel con todos los detalles. Un ejemplo de cómo funcionan las predicciones del notebook se puede ver en la *ilustración 2*.

In [18]:
```
#classifier.fit(Train_X, Train_Y)
```

In [19]:
```
y_pred = model.predict(Test_X)
print(classification_report(Test_Y,y_pred))
```

```
              precision    recall  f1-score   support

 non-suicide       0.96      0.84      0.90     33024
     suicide       0.82      0.96      0.89     25686

    accuracy                           0.89     58710
   macro avg       0.89      0.90      0.89     58710
weighted avg       0.90      0.89      0.89     58710
```

In [20]:
```
y_pred = p2.predict(Test_X)
print(classification_report(Test_Y,y_pred))
```

```
              precision    recall  f1-score   support

 non-suicide       0.96      0.84      0.90     33024
     suicide       0.82      0.96      0.89     25686

    accuracy                           0.89     58710
   macro avg       0.89      0.90      0.89     58710
weighted avg       0.90      0.89      0.89     58710
```

Pruebas

In [21]:
```
examples = ['i want to destroy myselffor once everything was starting to feel okay again but it all came tumbling down and i
example_counts = vectorizer.transform(examples)
predictions = classifier.predict(example_counts)
predictions
```

Out[21]:
```
array(['suicide', 'suicide'], dtype='<U11')
```

Se puede observar como reconoce palabras críticas y categoriza los textos según las encuentra

In [22]:
```
new_input = ['i want to destroy myselffor once everything was starting to feel okay again but it all came tumbling down and i
```

In [23]:
```
new_output = p2.predict(new_input)
```

In [24]:
```
new_output
```

Out[24]:
```
array(['suicide'], dtype='<U11')
```

Ilustración 2: Resultados modelo Naive Bayes

4. Construcción de la aplicación

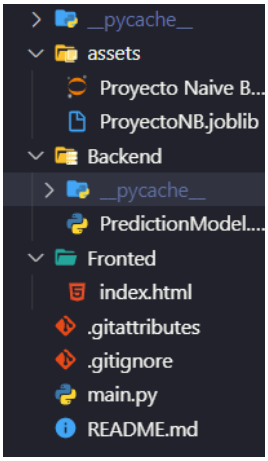

Ilustración 3: División del proyecto

Como se ve en la *ilustración 3*, dentro de la aplicación se dividió en tres directorios principales, assets, Backend y Frontend.
En assets, se guarda el notebook y el archivo .joblib obtenido. Dentro de Backend, se realizan las predicciones teniendo en cuenta el algoritmo Naive Bayes. Finalmente, dentro de Frontend se guarda el archivo html en el cual se hace el despliegue de la aplicación.
Por otra parte, en el archivo main.py se guarda el código para la construcción del API de la aplicación, tal como se ve en la *ilustración 4.*

main.py

```python
import string
import json
from fastapi import FastAPI, Request, Response
from fastapi.staticfiles import StaticFiles
from fastapi.templating import Jinja2Templates
from fastapi.responses import HTMLResponse
from Backend.PredictionModel import Model

app = FastAPI()
app.mount("/Fronted",StaticFiles(directory = "Fronted"), name = "Fronted")
templates = Jinja2Templates(directory="Fronted")

@app.get("/",response_class=HTMLResponse)
def read_root(request: Request):
    return templates.TemplateResponse("index.html", {"request":request, "message": "hola"})

@app.post("/calcularModelo")
async def calcularModelo(request: Request):
    input = await request.json()
    d = make_predictions(input["data"])
    result = {"data": d}
    encode = json.dumps(result)
    return Response(content=encode,media_type="application/json")

def make_predictions (texto: string):
    model = Model()
    result = model.make_predictions(texto)
    if result == "non-suicide":
        result = "no suicida"
    else:
        result = "suicida"

    res = "La frase de considera " + result
    return res
```

Ilustración 4: Elaboración del API

5. Despliegue final

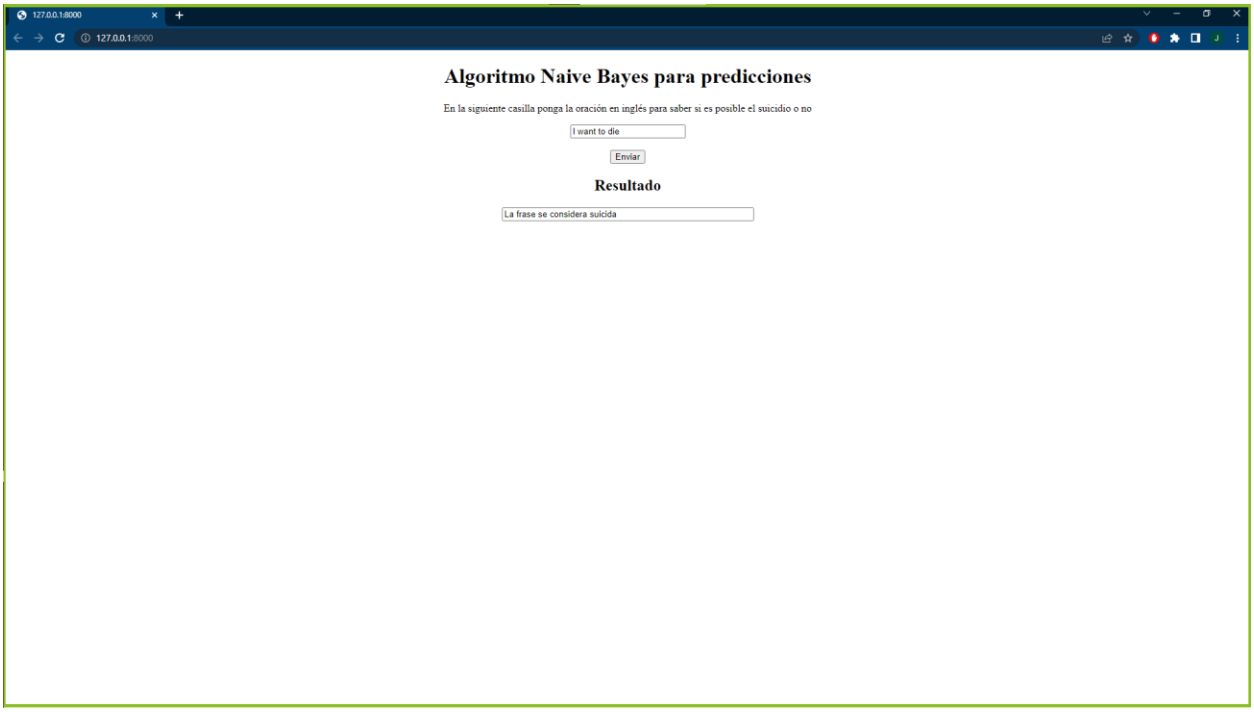

Ilustración 5: Aplicación final

6. Referencias

Bedi, G. (2018). *Simple guide to Text Classification(NLP) using SVM and Naive Bayes with Python*. Medium. Accedido el 18 de octubre de 2022, de https://medium.com/@bedigunjit/simple-guide-to-text-classification-nlp-using-svm-and-naive-bayes-with-python-421db3a72d34.

Naive Bayes Classifier in Machine Learning - Javatpoint. Accedido el 18 de octubre de 2022, de https://www.javatpoint.com/machine-learning-naive-bayes-classifier#:~:text=Na%C3%AFve%20Bayes%20Classifier%20is%20one,the%20probability%20of%20an%20object.

Brownlee, J. (2020). *Naive Bayes for Machine Learning.* Accedido el 18 de octubre de 2022, de https://machinelearningmastery.com/naive-bayes-for-machine-learning/