1.) Introduction

a.) Problem statement

700000 people die from suicide every year. Some of these usually send a message to social media before they commit the act. They usually send messages on boards like Reddit suicide watch or Facebook suicide group. These messages go undetected and the individual proceeds with the dreadful act.

b.) Goal

The goal of our service is to catch content related to suicide, monitor individual history, and prevent the user from acting on it.

2.) Dataset

The data we use comes from the Kaggle website. The website link: https://www.kaggle.com/datasets/nikhileswarkomati/suicide-watch. The dataset is composed of various text classified as suicide or non-sucide.

3.) Data Cleaning and Data Wrangling

We have 232074 rows of data and 3 columns

The following columns in our dataset: 'Unnamed: 0', 'text', 'class'

Unnamed: 0: It the id column

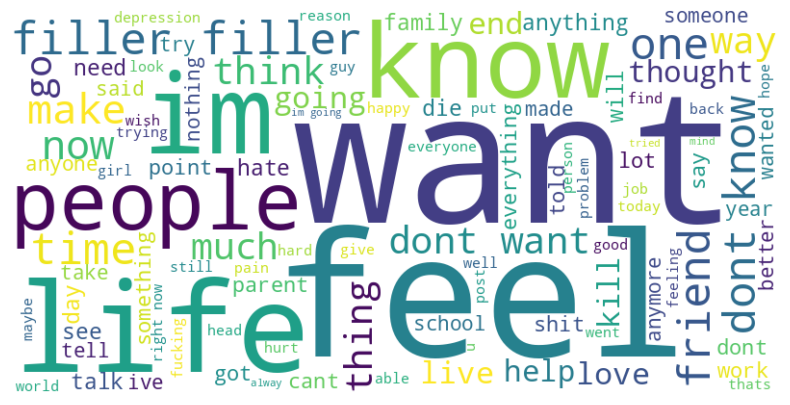
Text: The suicide content.

Class: The classification of the suicide content whether it suicide or non-suicide

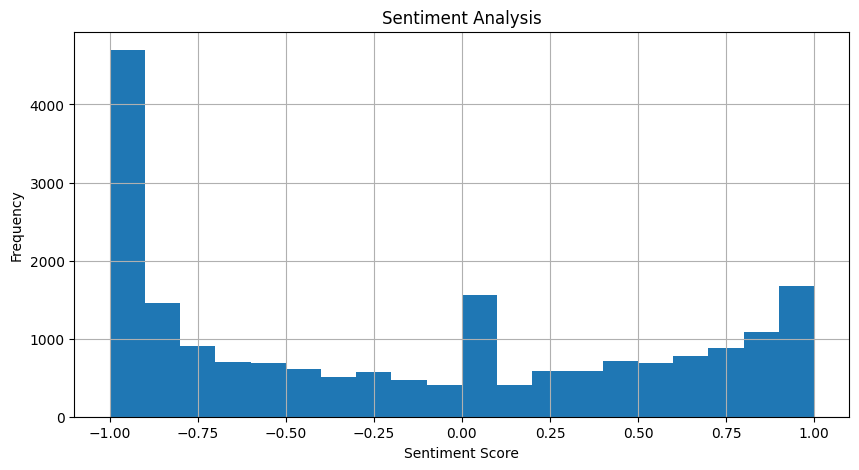
We did some data cleaning with the content. We check for null values. We minimize the number of rows to 20000 because of the total amount of computed resources we have available. We check for the average sizes of each context, We also check for the total amount of words in the collection. We also create new columns called “cleaned\_text” and “filtered\_tokens”. Column “cleaned\_text” contains the same content as the “text” column but without punctuation and stop words. The filtered\_tokens is the tokenized version of the cleaned\_text columns.

4.) Data exploratory analysis:

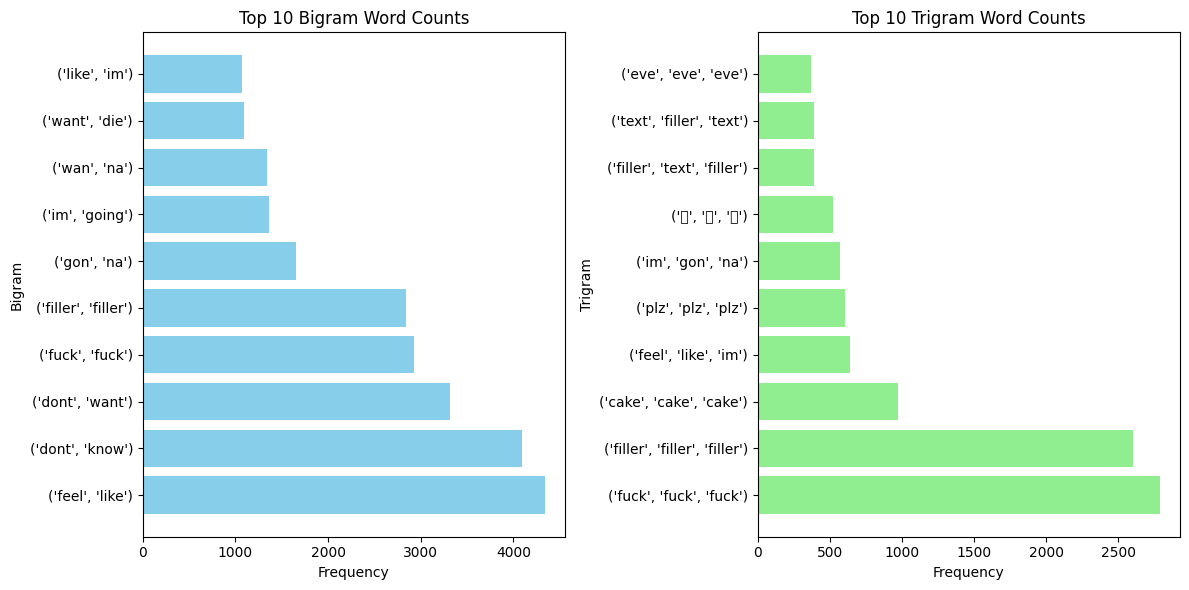
We decided to work on some graphs to analyze the content of the dataset. I built a word map to look at common words. The most common words are want, feel, and life. I am not surprised. Most suicidal contents have a phase like ending one's life or feeling or wanting to do something.



We also analyze the sentiment of the text. We have more negative sentiment then positive . Most content is negative, but only about half is suicidal content.



We also converted two more columns: “bigrams” and “trigrams”. We also want to see the most common bigrams and trigrams. The most common bigram is “feel like”. I wasn’t too surprised because most suicide context starts with what the individual feels like doing. It's like they are debating with themselves about their intentions. The most common trigrams are surprising but they end up being 'f\*ck', 'f\*ck', 'f\*ck. I know this is a common way to convey pain. I just didn’t expect it to rank so high.



5.) Deep analysis

We realize that we can’t throw this in a model using text. Machine can’t read text. So we create sentence embeddings of each text, which is a numerical representation of each sentence. I also map the class suicide. Non-suicide to 0 and suicide to 1. I decide to run this on six models, which is "Decision Trees Classifier", "Logistic Regression", "Support Vector Classifier", "Gradient Boosting Classifier", "Gaussian Naive Bayes", "Random Forest Classifier". I ran all model and got the following for each:

Random Forest Classifier:

Accuracy: 0.89575

Precision: 0.9023109243697479

Recall: 0.8814776808619805

F1 Score: 0.8917726446924474

Confusion Matrix:

[[1865 186]

[ 231 1718]]

Logistic Regression

Accuracy: 0.88625

Precision: 0.8720119521912351

Recall: 0.8984094407388404

F1 Score: 0.8850138994187516

Confusion Matrix:

[[1794 257]

[ 198 1751]]

Support Vector Classification

Accuracy: 0.90225

Precision: 0.8934343434343435

Recall: 0.9076449461262186

F1 Score: 0.9004835836090609

Confusion Matrix:

[[1840 211]

[ 180 1769]]

Decision Tree Classification

Accuracy: 0.8145

Precision: 0.8040302267002519

Recall: 0.818881477680862

F1 Score: 0.8113879003558718

Confusion Matrix:

[[1662 389]

[ 353 1596]]

Gradient Boosting Classifier

Accuracy: 0.89325

Precision: 0.8862944162436548

Recall: 0.8958440225756799

F1 Score: 0.8910436335799948

Confusion Matrix:

[[1827 224]

[ 203 1746]]

Gaussian Naive Bayes

Accuracy: 0.8375

Precision: 0.7961696306429549

Recall: 0.8958440225756799

F1 Score: 0.8430709802028006

Confusion Matrix:

[[1604 447]

[ 203 1746]]

Out of all models, support vector classification scores the highest. I want to improve my score. So I decided to put each model in a gridsearch.

Decision Tree Classifier

Accuracy: 0.849

Precision: 0.8337468982630273

Recall: 0.8619805028219599

F1 Score: 0.8476286579212916

Confusion Matrix:

[[1716 335]

[ 269 1680]]

Logistic Regression

Accuracy: 0.883

Precision: 0.8689586447433981

Recall: 0.8948178553104156

F1 Score: 0.8816986855409504

Confusion Matrix:

[[1788 263]

[ 205 1744]]

Support Vector Classification

Accuracy: 0.90575

Precision: 0.898984771573604

Recall: 0.9086711133914828

F1 Score: 0.9038019903036488

Confusion Matrix:

[[1852 199]

[ 178 1771]]

Gradient Boosting Classifier

Accuracy: 0.888

Precision: 0.8765679879578525

Recall: 0.896357106208312

F1 Score: 0.8863521055301877

Confusion Matrix:

[[1805 246]

[ 202 1747]]

Gaussian Naive Bayes

Accuracy: 0.8375

Precision: 0.7961696306429549

Recall: 0.8958440225756799

F1 Score: 0.8430709802028006

Confusion Matrix:

[[1604 447]

[ 203 1746]]

Random Forest Classifier

Accuracy: 0.8915

Precision: 0.894325871941697

Recall: 0.8814776808619805

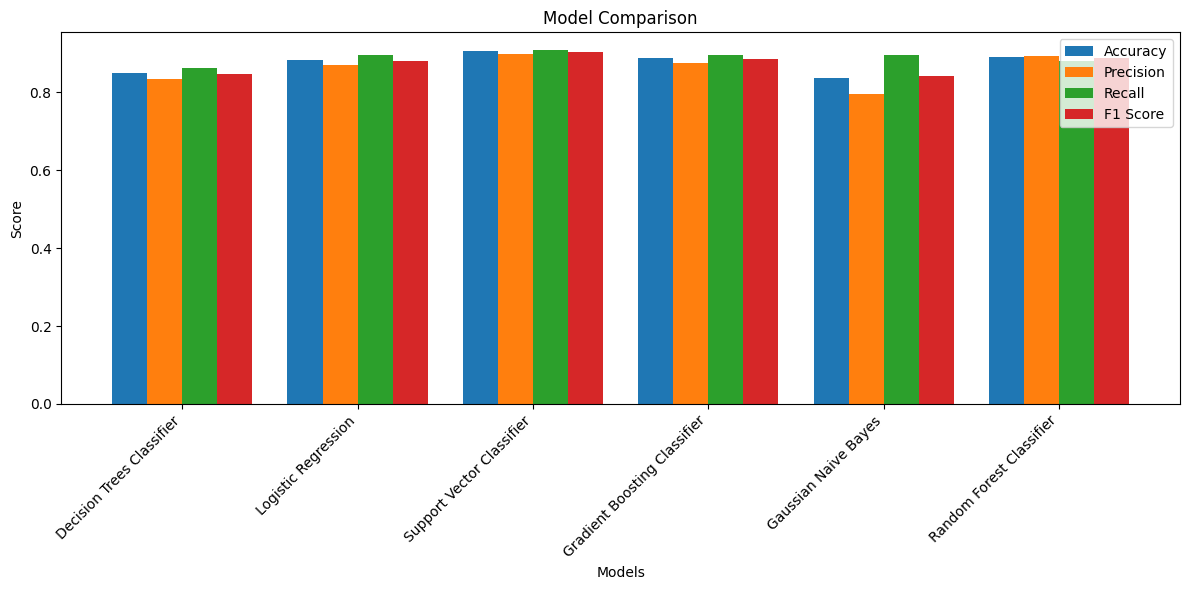
F1 Score: 0.8878552971576227

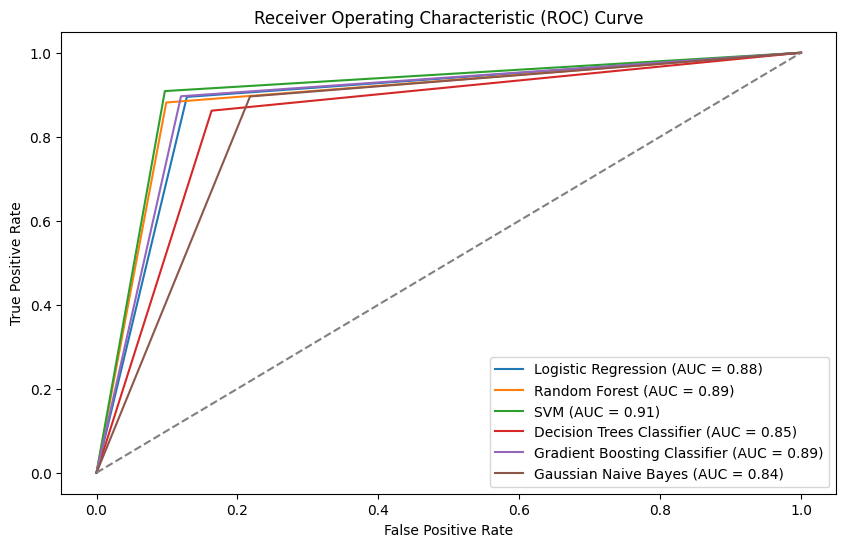
Confusion Matrix:

[[1848 203]

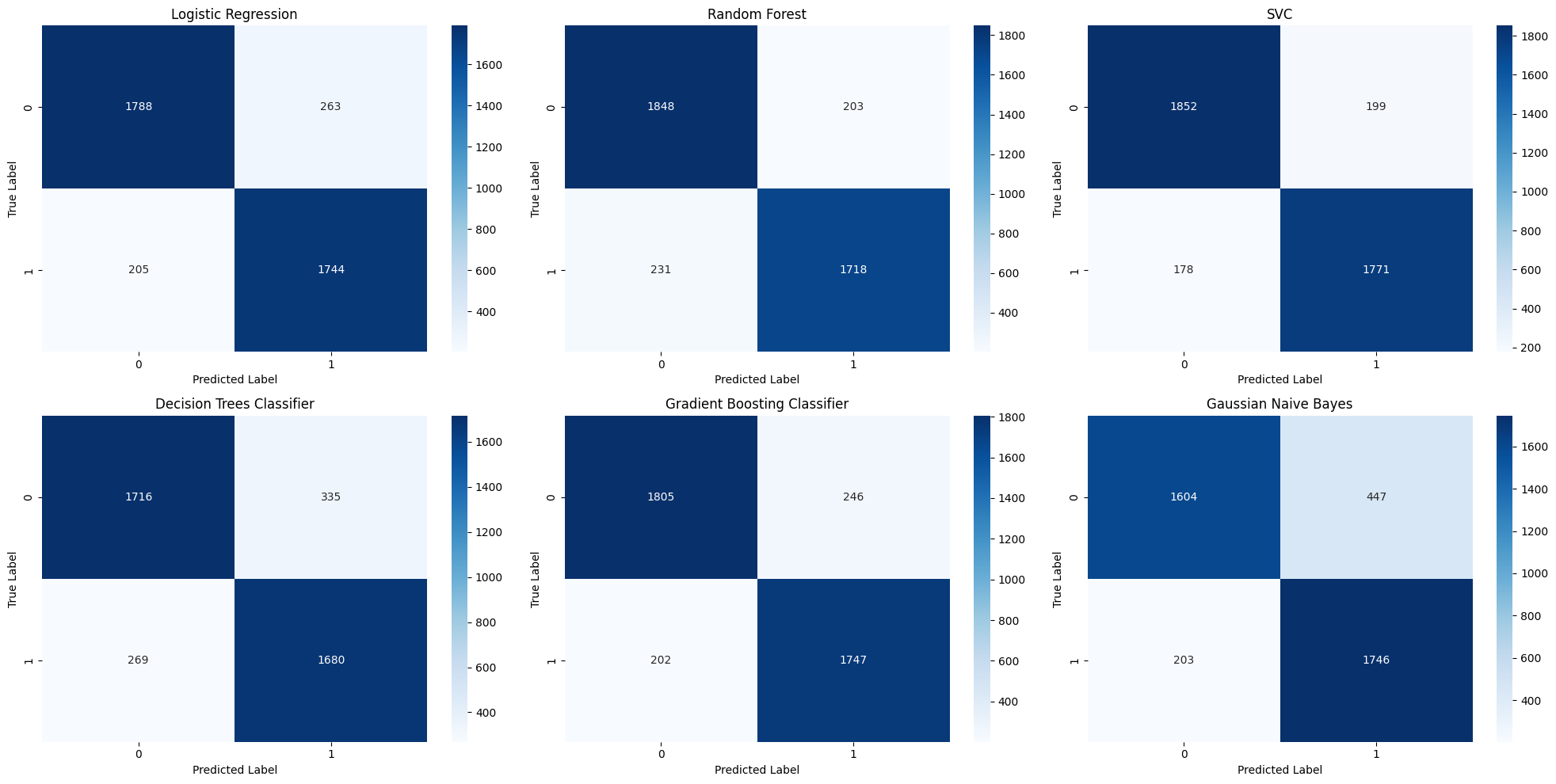
[ 231 1718]]

The support vector classification still ends up better than the other model, but it wasn’t much of improvement from its original model.



THe S

Support vector classification also scores the highest on ROC AUC score.



This is a graphical representation of the confusion matrix. Support vector classification scores the highest in true positive and true negative. Random forest classifier is right behind it when it comes to analyzing its true positives.