# Machine Learning

## Depression Detection in Tweets

**Author**

**Aris Tsilifonis**

**Supervisor**

**Theodoros Giannakopoulos**

**March 4,2024**

**Contents**

# 1.Abstract

Sentiment Analysis is a machine learning technique that determines the emotional tone of a message. The aim of this project is to detect a specific kind of emotion, namely depression. In today's era, people express their opinions in social platforms, like Twitter at a very high rate. A specific tweet can reveal many aspects of the psychology of the user. Hence, engineers can utilize Machine Learning Techniques to improve the experience of the users. On this occasion, if depression is detected in a tweet, safety measures might be taken thereafter. For instance, the website may adjust the content that is shown to the user. In more severe cases, psychological help may be provided author of the problematic tweet. This work can have very important impact, such as reducing the suicide rate and limiting crime commitment.

The detection of emotion can be very challenging task. Large datasets must be processed extensively to constitute the ML models effective at predicting sentiment. Almost twenty thousand tweets were processed for this purpose. Emotions, emojis and other text features significantly improved the accuracy of the predictions. The data was used to train a collection of classical machine learning techniques, such as Multinominal Naive Bayes (MNB), K-Nearest Neighbors (KNN), Decision Tree (DT), Support Vector Machine (SVM), Randon Forest (RF) and Logistic Regression (LR). The models were evaluated according to performance metrics (accuracy, f1-score, AUC, K-Fold cross-validation, learning curve) to define the best parameters.

The outcome of the research is very positive since almost 90% accuracy was attained. The variety of the evaluation methods that were applied solidify the good results. Multinominal Naive Bayes, Logistic Regression and SVM predicted depression better than the other models.

# 1.Theory

# Multinomial Naïve Bayes

Multinomial Naïve bayes employs the frequencies of words in a document as its features or predictors for classification. It is predominantly applied for classifying text.

It assesses the probability of each category for a given text and selects the category with the highest probability as output. The presence or absence of a feature is considered independent of any other's feature absence or presence. This "naïve" assumption of feature independence simplifies probability calculations, enhancing the algorithm's computation efficiency.

### Multinomial Naïve Bayes Algorithm

Let **S** be an input string, **P**- a corpus of **d** documents, **C** a set of **m** classes:

Compute the class $C_s$ of input S as follows:

1. Split string **S** into a set of **n** terms
2. For each j-th class $C_j$ j=1… m do:
    a. Compute the vector **W** of **n** features where $\forall \mathbf{w_{ji}} \in \mathbf{W}$ where $\mathbf{w_{ji}}$ is the frequency of which the i-th term occurred in the documents of class $C_j$
    b. Evaluate the prior $\mathbf{p(C_j)}$ probability as the full probability that the document d occurred in the documents of class $C_j$.
    c. Compute the posterior $\mathbf{Pr(C_j|W)}$ by adding prior to the sum of each term's $\mathbf{w_i}$ given $\mathbf{C_j}$, probabilities $\mathbf{p(w_i|C_j)}$:

$$\boldsymbol{Pr(C_j|W) = log\ p(C_j) + \sum_{i=1}^{n} (w_i * log\ p(w_i|C_j)}$$

*Prior* *Likelihood*

    d. Determine a class $\mathbf{C_s}$ for which $\mathbf{Pr(C_j|W)}$ is maximum:

$$\mathbf{C_s = argmax_{j\in\{1,..,m\}} |Pr(C_j|W)|}$$

    e. The algorithms complexity σ is evaluated by:

$$\boldsymbol{\sigma = O(nm * (d + 2))}$$

Where d is the number of documents in P, n is the number of terms in sample S, m is the number of classes.

Bayes Theorem: $\boldsymbol{p(c|D) = \frac{P(c|D)P(c)}{P(D)}}$ , class c document D, $P(D)$ is constant for all the classes.

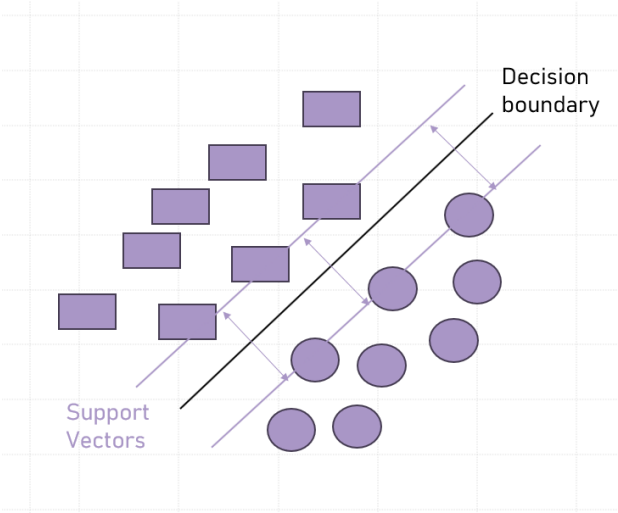
Multinomial naïve Bayes can be improved with semi-supervised learning. Some of the samples that are classified by the model are inserted to the corpus of the training dataset to improve performance in the re-evaluation of the model.

## Supported Vector Machines (SVM)

Support Vector Machines (SVM) are a set of supervised learning methods used for classification, regression, and outliers' detection. The core principle of SVM is to find the hyperplane that best divides a dataset into classes.

The strength of SVM lies in its use of kernels, which allow it to efficiently perform a non-linear classification, thereby transforming the input space into a higher dimensional space.

SVM is particularly well-suited for complex but small- or medium-sized datasets, offering high accuracy and robustness against overfitting, especially in cases where the dimensionality of the data exceeds the number of samples. For the purposes of this project, only linear SVM was used.




The objective is to find a plane that has the maximum margin, the maximum distance between data points of both classes. Maximizing the margin distance provides some assurance so that new data points can be labeled confidently. The hyperplane (decision boundary) is a line in this case since we have only two classes.

In SVM, we take the output of the linear function and if that output is greater than 1, we identify it with one class and if the output is -1, we identify is with another class. The loss function that helps maximize the margin is **hinge loss**.

$$c(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{f}(\boldsymbol{x})) = \begin{cases} \mathbf{0}, & \boldsymbol{if}\ \boldsymbol{y} * \boldsymbol{f}(\boldsymbol{x}) \geq \mathbf{1} \\ \mathbf{1} - \boldsymbol{y} * \boldsymbol{f}(\boldsymbol{x}), & \boldsymbol{if}\ \ \boldsymbol{y} * \boldsymbol{f}(\boldsymbol{x}) < \mathbf{1} \end{cases}$$

The cost is zero if the predicted and actual value are of the same sign. If they are not, then we calculate the loss value. To balance the margin between the margin maximization and loss, we add a regularization term.

$$\min_{\boldsymbol{w}} \boldsymbol{\lambda}\|\boldsymbol{W}\|^{\mathbf{2}} + \boldsymbol{C}\sum_{\boldsymbol{i}=\mathbf{1}}^{\boldsymbol{n}} \boldsymbol{max}(\mathbf{0}, \mathbf{1} - \boldsymbol{y_i}(\boldsymbol{w^T x} - \boldsymbol{b})\ \boldsymbol{)}$$

{ $w^T x - b$ is the black line in the figure above}

To find the gradients, we take the partial derivatives with respect to the weights.

C parameter controls the trade-off between maximizing margin and minimizing loss.

$$\frac{\delta}{\delta w_k}\lambda\|W\|^2 = 2\lambda w_k$$

$$\frac{\delta}{\delta w_k}(1 - y_i\langle x_i, w\rangle) = \begin{cases} 0, & if\ y_i\langle x_i, w\rangle \geq 1 \\ -y_i x_{ik}, & if\ \ y_i\langle x_i, w\rangle < 1 \end{cases}$$

When there is no misclassification gradient is calculated only by regularization term:

$$w = w - \alpha * (2\lambda w)$$

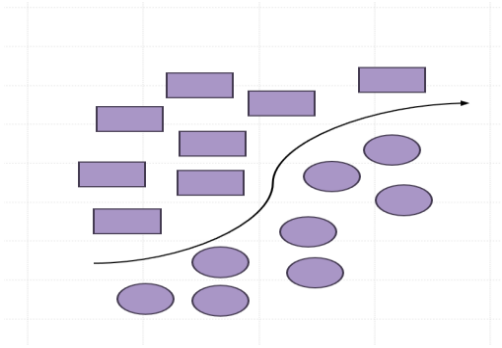
When there is a misclassification, we include the loss:

$$w = w + \alpha * (y_i x_i - 2\lambda w)$$

Decision Boundary $= \frac{\|w\|}{2}$

# Logistic Regression

Logistic Regression is a statistical method used for binary classification. It models the probability that a given input belongs to a particular category. Logistic Regression works by fitting a logistic curve to the data and using the sigmoid function to estimate probabilities, which are then mapped to the closest class. This technique is widely used for predictive analysis to determine outcomes that have two possible states like yes/no, win/lose, alive/dead.



The logistic regression classifier can be derived by analogy to the linear regression hypothesis:

$$h_\theta(x) = \theta^T x$$

**For logistic regression hypothesis** we have:

$$h_\theta(x) = g(\theta^T x)$$

$$g(z) = \frac{1}{1 + e^{-z}}$$

**Sigmoid function:** $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$ ***or σ function***

***Logistic regression decision boundary***

$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \cdots + \theta_n x_n)$$

***The logistic regression will classify depression if:***

$\theta_0 + \theta_1 x_1 + \cdots + \theta_n x_n \geq 0$ (0: decision boundary)

**Maximum Likelihood estimator:**

$L(\theta) = \prod_1^n \sigma\left(\theta^T x^i\right)^y \left(1 - \sigma\left(\theta^T x^i\right)\right)^{1-y}$, $y \sim bernoulli\ \{0, 1\}$

***Cost function:***

$$J(\theta) = -\left[y \log\left[\sigma\left(\theta^T x^i\right)\right] + (1-y)\, log\left(1 - \sigma\left(\theta^T x^i\right)\right)\right]$$

$$\min_\theta J(\theta)$$

***Gradient Descent:***

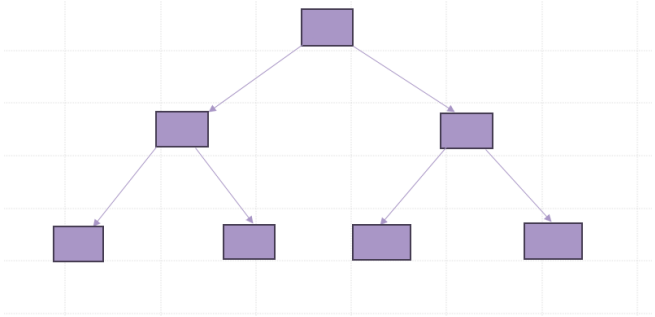
***Iterate until convergence {***

$$\vartheta_j = \theta_j - \alpha\,(h_\theta(x) - y)x$$

***} (iterate along the negative direction of theta / minimizing sequence)***

# Decision Trees

The decision tree stands as a widely recognized and highly effective method for both classification and prediction tasks. It is depicted as a tree-like model in flowchart form. In this structure, each internal node signifies a test on a feature, every branch indicates the possible result of this test, and the leaf nodes, or terminal nodes, contain the classification labels. Decision trees offer the advantage of executing classifications with minimal computational demand. Furthermore, they are versatile, capable of processing variables that are either continuous or categorical in nature.



**Initiating at the Root**: The process begins at the root node, which encapsulates the full dataset.

**Identifying Optimal Splits**: The algorithm searches for the most informative feature that can divide the dataset into clear subsets. It achieves that by choosing a path at a tree's junction.

**Creating Branches**: Depending on the feature's value, the dataset is split into smaller groups, which form the branches of the tree. Each branch symbolizes a decision pathway.

**Iterative Splitting**: This procedure of splitting by asking questions continues down each branch until it arrives at the leaf nodes, which denote the final predictions or classes.

Decision trees fix disorderness in the tree, which happens when we can't really decide the label of the class based on the split, by using entropy.

$$\boldsymbol{Entropy}(\boldsymbol{sp}) = -\boldsymbol{pr}_{(+)}\boldsymbol{log}\boldsymbol{pr}_{(+)} - \boldsymbol{pr}_{(-)}\boldsymbol{log}\boldsymbol{pr}_{(-)}$$

$pr_{(+)}$ is the probability of positive class.

$pr_{(-)}$ is the probability of negative class.

Sp is the subset of the training set.

Entropy basically measures the impurity of a node. The level of impurity reflects how mixed the data is. A completely pure sub-split indicates that the data falls into a single category, yielding exclusively "yes" or "no" outcomes.

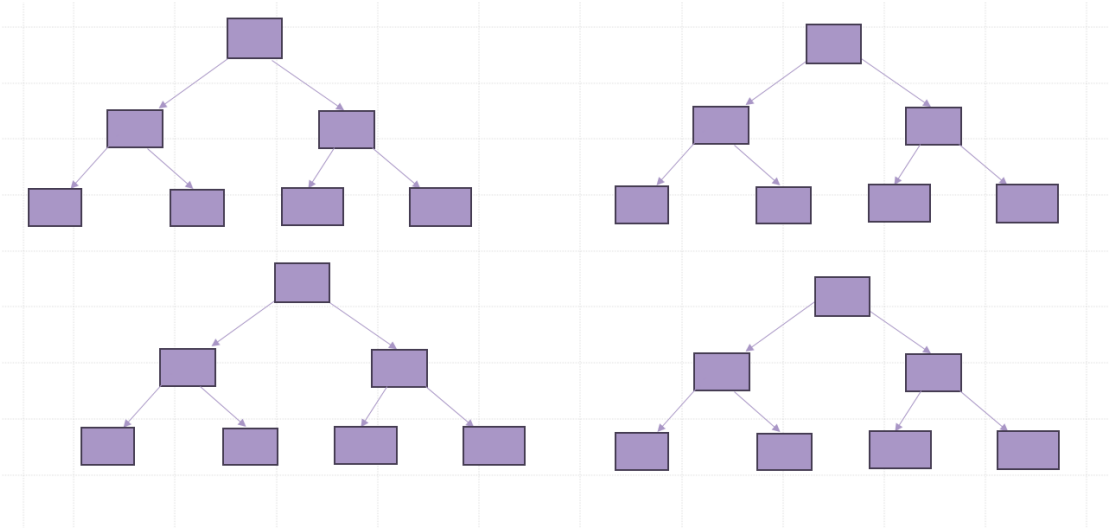
The root node of the tree, which is essentially the feature that the split is based on, is decided by information gain. It is calculated by:

$$\boldsymbol{Information\ gain} = \boldsymbol{Entropy}(\boldsymbol{Y}) - \boldsymbol{Entropy}(\boldsymbol{Y}|\boldsymbol{X})$$

$\boldsymbol{Entropy}(\boldsymbol{Y})$ measures the entropy of the root node. It contains instances that have labels of both classes (depression and fine). The root node is divided into two nodes. The one node will contain more instances of tweets with depression. The other will contain instances that are not considered problematic. $\boldsymbol{Entropy}(\boldsymbol{Y}|\boldsymbol{X})$ expresses the weighted average of entropy of each node. The same procedure is performed for the other features as well. The split of the root node is decided by the feature that produces the highest information gain.

Other parameters of the model can help us alleviate many problems that may occur, such as overfitting. For instance, one may decide to stop splitting after a specific depth. Another way is to set the minimum number of samples for each spilt (It is denoted by min_samples_split). When looking for best split, someone can explicitly define the maximum number of the features to consider.

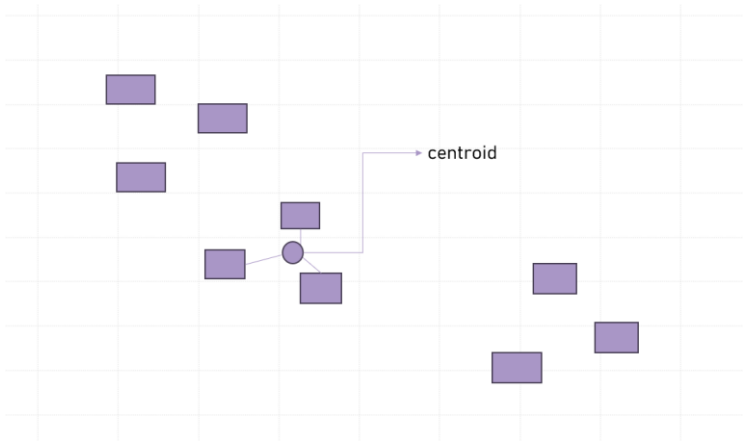
## Random Forest



Random forest is an ensemble method that combines a multitude of decision trees to make predictions. In this approach, each tree within the random forest contributes its prediction, with the most frequently predicted class becoming the final output of the model.

The underlying principle of the random forest is straightforward yet effective: By pooling the predictions of several independent models (trees), the collective decision is typically more accurate than that of any single tree within the ensemble. In the Random Forest algorithm, each decision tree is built using a random selection of data points and features. This means that for a dataset containing 'k' entries, 'n' random instances and 'm' features are chosen to form the foundation of each individual tree. Individual decision trees are constructed for each sample and the final output is considered based on Majority Voting for classification.

# K-nearest neighbors




The KNN model is a supervised learning technique which assumes that similar things exist in close proximity. In other words, similar things are near to each other. The steps of K-NN algorithm are the following:

**Set 'K' Value**: Choose a specific number of neighbors, denoted as 'K'.

**Distance Calculation**: For every point in the dataset:

3.1 Measure the distance from the point to the query instance.

3.2 Record the distance along with the point's index in a list.

**Organize Distances**: Sort the list of distances and corresponding indices from the lowest distance to the highest.

**Select Neighbors**: Choose the top 'K' points from the sorted list.

**Extract Labels**: Retrieve the labels associated with these 'K' points.

**Prediction**: For classification tasks, identify and return the most frequent label among the K labels.

For distance calculation, we can use Euclidean distance:

$$Euclidean\ Distance(p,q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$$

For data points p,q

# Evaluation methods

A **learning curve** illustrates the relationship between a classifiers' training performance and its test performance as the volume of training data changes.

It assesses the degree to which additional data improves the classifiers' effectiveness, helping to determine whether the current dataset is sufficient or if the classifiers' performance could be enhanced with more data.

**Cross-validation** is a method used to assess how well a machine learning model will perform on unknown data. It involves partitioning the data into several "folds," then using each fold in turn as a validation set while the model is trained on the remaining folds. This procedure is carried out multiple times, with each fold being used as the validation set once. The performance metrics from each validation phase are then averaged, providing a comprehensive evaluation of the model's effectiveness. Cross-validation ensures that the chosen model is reliable and generalizes effectively to new data. **Stratified method** was used to enhance cross-validation accuracy since it maintains the same ratio in the sample as in the original data. It helped the models generalize well. The performance of stratified k-fold cross-validation was compared with **leave-one-out** cross validation method. In leave-one-out cross-validation, the machine learning model undergoes training n times, corresponding to the total number of samples in our dataset. During each training iteration, a single data point is held out as the test set, and the remaining data points are utilized to train the model.

**Confusion matrix** allows you to see the frequency of correct and incorrect predictions made by the model, compared against the actual classifications in the test data. It involves four types of predictions. The **True Positives** (TP), **True Negatives** (TN), **False Positives** (FP) **and False Negatives** (FN). TP are the cases in which the model correctly predicts the positive class. TN are the cases in which the model correctly predicts the negative class. FP are the cases in which the model incorrectly predicts the positive class when the actual class is negative. FN are the cases in which the model incorrectly predicts the negative class when the actual class is positive. Several scores can be computed based on this matrix:

$$\boldsymbol{Accuracy} = \frac{\boldsymbol{TP + TN}}{\boldsymbol{TP + FP + TN + FN}}$$

$$\boldsymbol{Precision} = \frac{\boldsymbol{TP}}{\boldsymbol{TP + FN}}$$

$$\boldsymbol{Recall} = \frac{\boldsymbol{TP}}{\boldsymbol{TP + FN}}$$

$$\boldsymbol{f1} = \boldsymbol{2} * \frac{\boldsymbol{Precision} * \boldsymbol{Recall}}{\boldsymbol{Precision + Recall}}$$

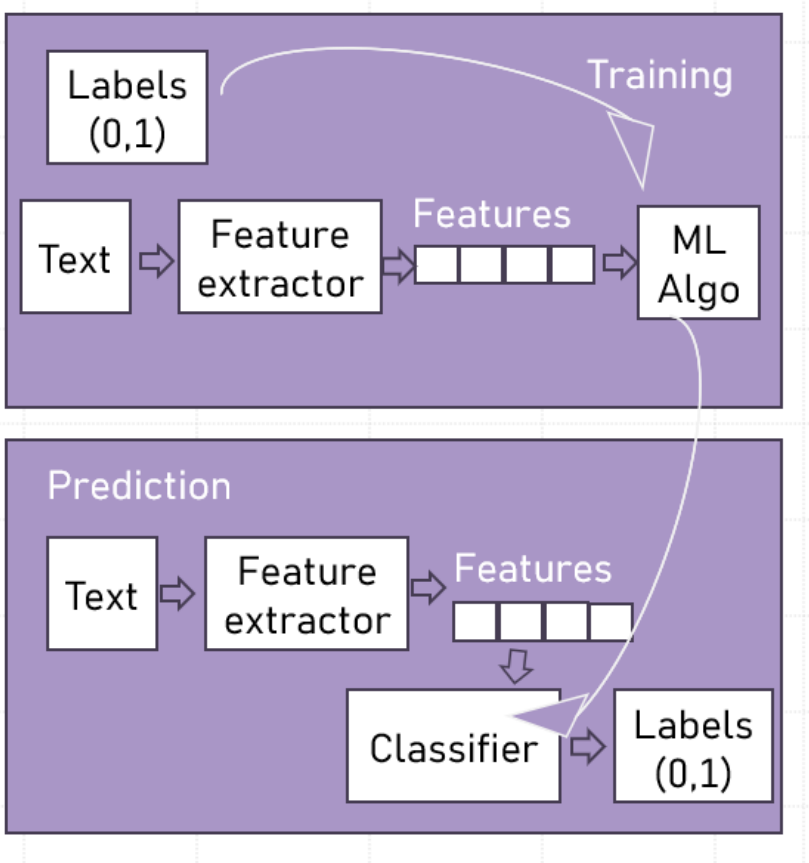
We use precision when we want to minimize false positives and recall reducing false negatives. To compute **macro f1**, one must find average recall then average precision. In this implementation, **weighted f1** was mainly used. It is the same as macro but weighted by the true number of samples in class (namely **support**).

Receiver operating characteristic (ROC) curve is a graphical representation of true positive rate and false positive rate at various threshold levels. A threshold determines the probability score at which each sample is classified as positive or negative outcome by the model. True positive rate is the recall score.

$$\boldsymbol{False\ Positive\ Rate} = \frac{\boldsymbol{FP}}{\boldsymbol{FP + TN}}$$

The Area Under the Curve score (AUC) measures the ability of the model to discriminate over positive and negative class. A score 1.0 suggests a perfect model with no error. A 0.5 score represents a model that can't discriminate effectively, which is considered as random guessing. Generally, higher AUC score indicates better performance of the model. The goal is to increase true positive rate and avoid false positives by selecting the best threshold for the model. This means that the ROC curve should approach top left corner.

## 2.Methodology



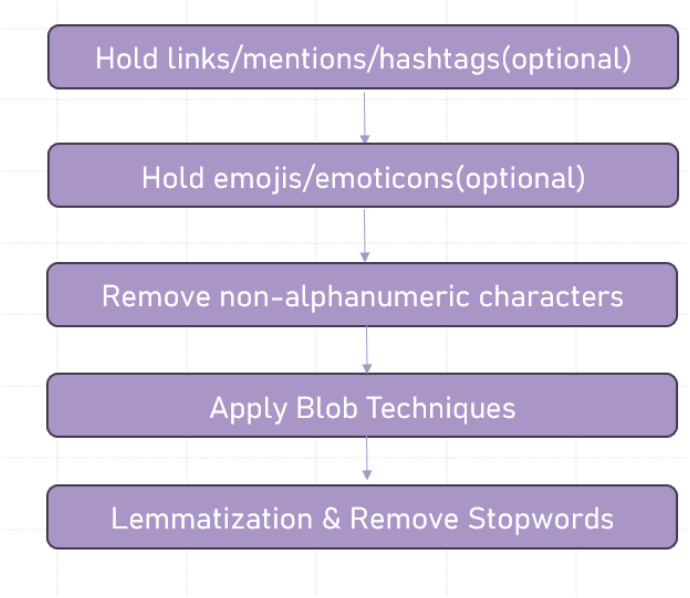

During the training phase, the model is taught to link text with the appropriate outcome using the training samples provided. The feature extractor converts the text

input into a feature vector, which is then inputted into the machine learning algorithm to create a model.

In the prediction phase, the same feature extractor converts new, unseen text inputs into feature vectors. These vectors are inputted into the trained model, which then produces the predicted labels.

**Data preprocessing**

The data preprocessing is crucial step before the training of the machine learning models. Holding Links/mentions/hashtags step is conditional and suggests that you may choose to retain elements such as URLs, social media user mentions, or hashtags. Maintaining those elements in the dataset improved the performance of the models. Also, emojis and emoticons express the emotional tone of the text, and they remained in text after preprocessing. It should be noted that in feature extraction, a specific token pattern was used so that these pictographs could influence feature vectors. Removing non-alphanumeric characters step let us preserve only the important information. Lemmatization is the process of converting words to their base form (“run” from “running”). Stopwords were removed since they occur frequently in a language and usually don't carry important meaning for analysis (like “the”,”and”). Text Blob library offers many built-in methods for sentiment analysis.




**Feature extraction**

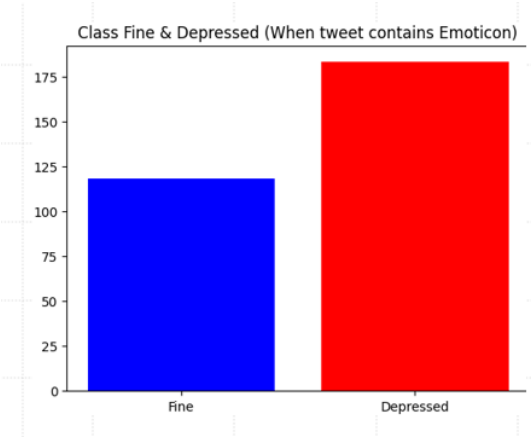
Count Vectorizer: It works by breaking down text into words (or tokens) and counting how many times each word occurs.

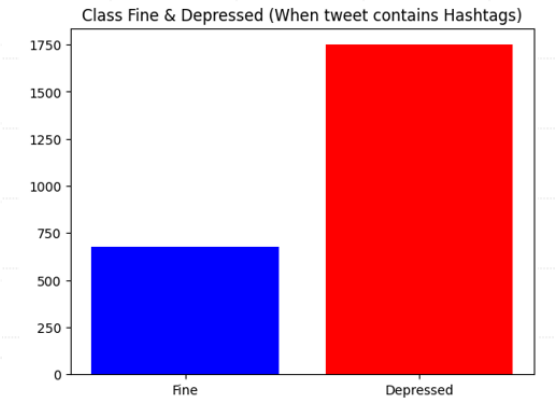
TF-IDF vectorizer: Stands for Term Frequency-Inverse Document Frequency, is a numerical statistic used to indicate how important a word is to a document in a collection or corpus. Term Frequency (TF): These measures how frequently a term occurs in a document. Inverse Document Frequency (IDF): This measures the importance of the term across a set of documents. It's calculated by taking the logarithm of the number of documents in the corpus divided by the number of documents where the specific term appears.

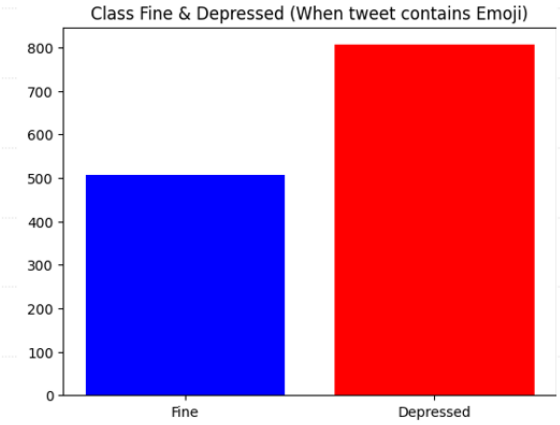
# 3.Results

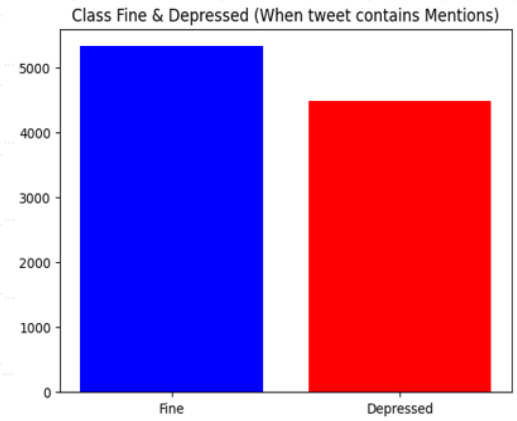
In this project, results are mainly quantitative.

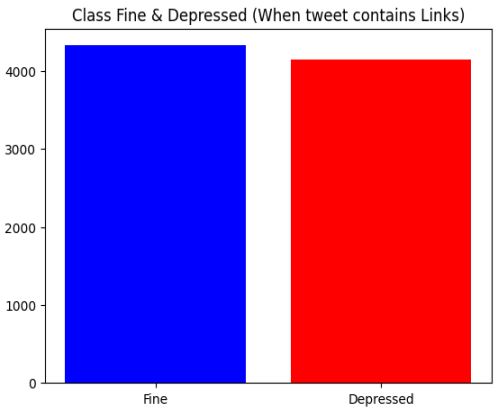
### Data Analysis



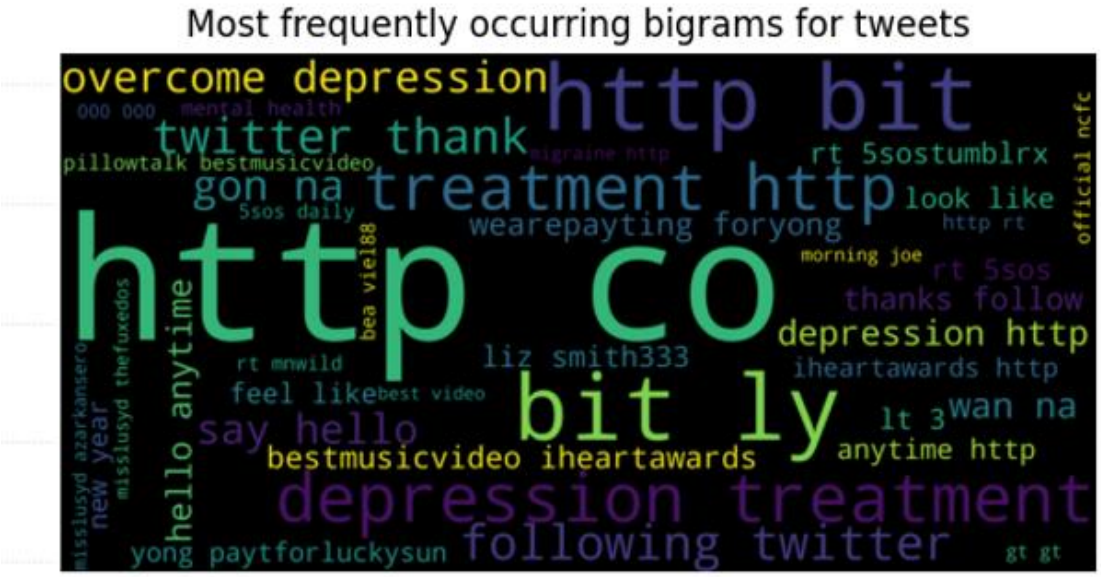




```
Number of total rows
20000
Percentage of rows containing Emoticons
1.505 %
Percentage of rows containing Emojis
6.859999999999999 %
Percentage of rows containing Links
42.35 %
Percentage of rows containing Hashtags
12.120000000000001 %
Percentage of rows containing Mentions
49.03 %
Percentage of rows containing Depression
50.0 %
Percentage of rows not containing Depression
50.0 %
```

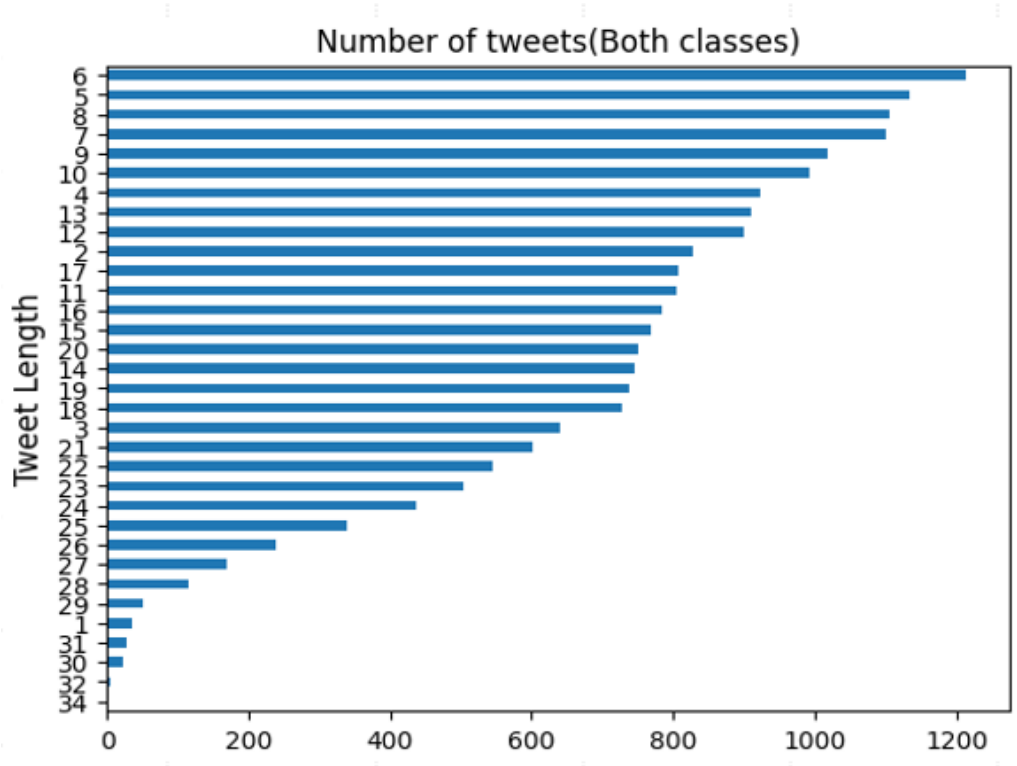
The dataset consists of 20.000 tweets and each of the two classes is represented equally. Except for links and mentions, the other textual elements do not exist often in the data. The tweets were divided into two dataframes, based on the class that they were labeled. It was noticed that tweets with depression were more likely to include

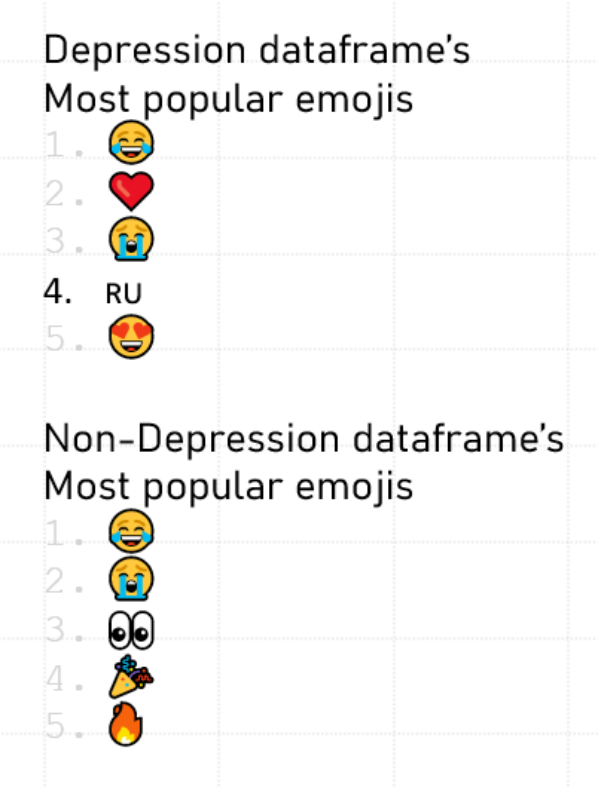
emoticons, emojis and hashtags. On the other hand, tweets that were not labeled with depression, contained mentions and links at a higher rate.




To further examine the dataset, a wordcloud of the most frequently occurring bigrams was created. Apart from links and mentions, phrases namely “overcome depression” and “depression treatment”, express the essence of the dataset.

From the diagram below, it can be understood that tweets usually have short length. Most of them are under ten characters. Greater than twenty characters length occur less often.






Laughing and crying emoji existed in both classes. Emojis with heart occurred more often in the depressed tweets. Fine tweets usually had fire or eyes emoji.

## Performance Overview

The purpose of this assignment was not only to predict if a tweet expresses depression but also to realize which type of dataset performs best.

| | (without emojis, links, mentions) preprocessed text | unchanged | emoji textual replacement | (with emojis, links, mentions) preprocessed text |
|---|---|---|---|---|
| MNB | 0.868 | 0.8615 | 0.82 | 0.8765 |
| KNN | 0.762 | 0.786 | 0.746 | 0.7725 |
| DT | 0.7885 | 0.7135 | 0.731 | 0.8165 |
| RF | 0.7955 | 0.7525 | 0.7395 | 0.827 |
| SVM | Not Tested | Not Tested | Not Tested | 0.877 |

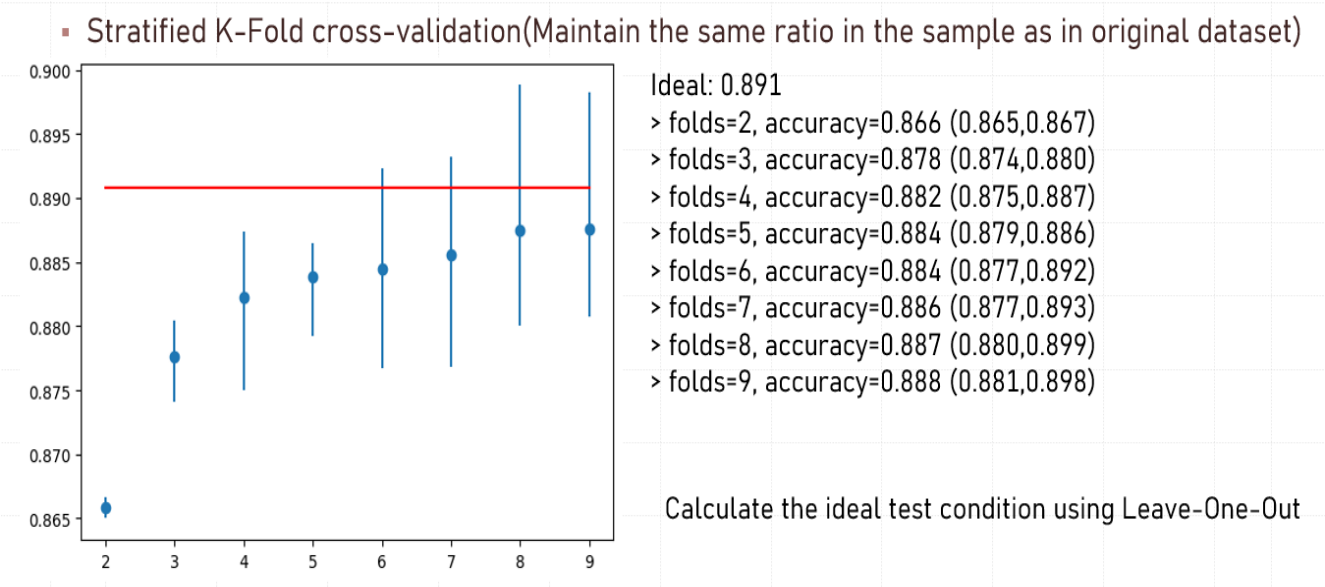
Each of the machine learning models was examined based on four tests. Initially, no preprocessing was performed on the tweets. Then, preprocessed text without emojis, links and mentions were introduced to the models. Preprocessed tweets, with their emojis converted to text, were also fed into the models. Finally, preprocessed text with emojis, links and mentions were tested thoroughly. The floating-point number in each cell indicates the accuracy of the models.

Multinomial Naïve Bayes had robust performance across all the methods. However, holding emojis, links and mentions proved the most successful implementation. More specifically, it was around 1% better than the second-best method.

Regarding KNN, when the dataset remained unchanged, the accuracy peaked. The model had the worst results in emoji textual replacement method. It was approximately 4% worse than the unchanged method. Decision Tree, Random Forest and SVM predicted sentiment more accurately in the fourth method, when emojis, links and mentions where maintained in the preprocessed dataset. Due to long running times, SVM was only tested in the fourth method. Emoji textual replacement method produced unsatisfying results and it was removed from the notebook for clarification purposes. The technique that replaces emoji with text remained in the code to provide insight about the method. It is worth mentioning that the results were produced based on feature vectors from tf-idf. Count vectorizer was tested too but results were not included in the report since they were worse most of the time.

# Performance Overview

### Evaluation of Multinomial Naïve Bayes




Since Multinomial Naïve Bayes produced the best outcome, we needed to verify that these results can be maintained in more demanding tests. In the diagram above, we see that the mean accuracy of the model improves when the number of the folds increases. Peak performance is reached in the ninth fold. Each bar represents the range of accuracy across each fold. The dots inside the bars depict the mean accuracy in each fold. It is evident that the eight and ninth fold achieve maximum accuracy of almost 90%. Since the dots are below the red line, it means that the evaluation method underestimates the accuracy. The stratified k-fold cross validation was compared to the leave one out (red line). After the fifth fold, stratified k-fold can produce better accuracy than leave one out. This graph allowed us to have a better understanding of the best model's accuracy.

**Confusion matrix**

Multinomial NB

| | Fine (predicted) | Depression (Predicted) | Sum |
|---|---|---|---|
| Fine (true) | 819 | 155 | 974 |
| Depression (true) | 91 | 935 | 1026 |
| Sum | 910 | 1090 | 2000 |

Logistic Regression

| | Fine (predicted) | Depression (Predicted) | Sum |
|---|---|---|---|
| Fine (true) | 829 | 145 | 974 |
| Depression (true) | 155 | 871 | 1026 |
| Sum | 984 | 1016 | 2000 |

Confusion matrices of the two most accurate models is showed. Multinomial Naïve Bayes has better precision and f1-score than logistic regression model for both classes. Macro average and weighted average scores show similar pattern. Recall score for “fine” class is 1% better in logistic regression. MNB has 6% better recall score for “depression” class. Average scores are a lot higher for MNB in comparison with LR. In this project, depression class is more important than fine class. As a result, we care more about the recall in the depression class. It is more important to find tweets which truly have depression, even if it means that some are predicted falsely “fine”.

## Classification Report

**Multinomial Naïve Bayes vs Logistic Regression**

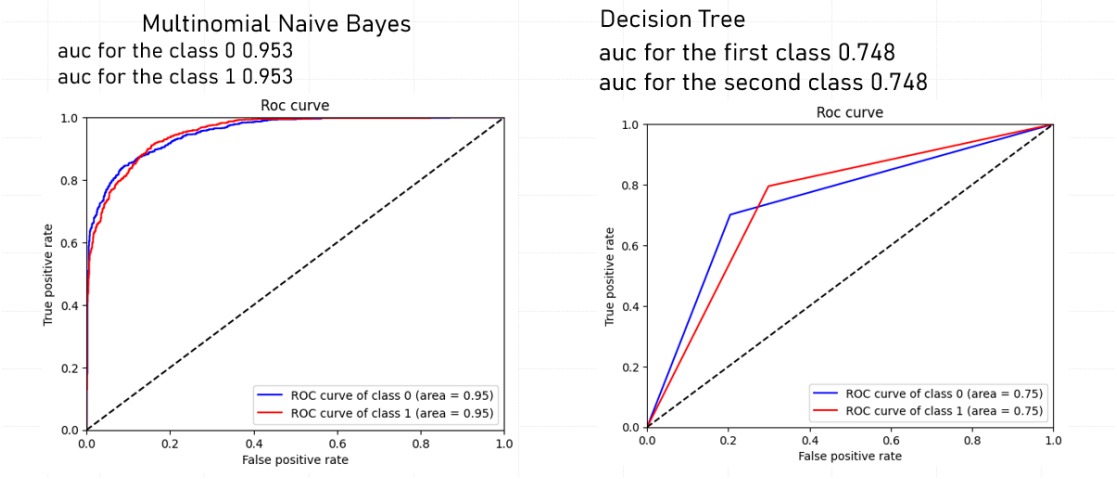
- Accuracy Score: 0.877

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.90 | 0.84 | 0.87 | 974 |
| 1 | 0.86 | 0.91 | 0.88 | 1026 |
| accuracy | | | 0.88 | 2000 |
| macro avg | 0.88 | 0.88 | 0.88 | 2000 |
| weighted avg | 0.88 | 0.88 | 0.88 | 2000 |

- Accuracy Score: 0.85

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.84 | 0.85 | 0.85 | 974 |
| 1 | 0.86 | 0.85 | 0.85 | 1026 |
| accuracy | | | 0.85 | 2000 |
| macro avg | 0.85 | 0.85 | 0.85 | 2000 |
| weighted avg | 0.85 | 0.85 | 0.85 | 2000 |

## ROC Curves




The Area under the Curve score is equal for each of the classes in MNB's and DT's diagrams. The goal is to achieve a score close to 1.0. It is evident that in MNB model the auc is nearly perfect for both of the classes. Also, the curves are close to the top left corner of the graph, indicating a high level of performance. On the other hand, decision tree model is less strong. Even though, DT's curves are above the diagonal line (equals random guessing), they cover less area than MNB's. As a result, MNB has better predictive abilities.

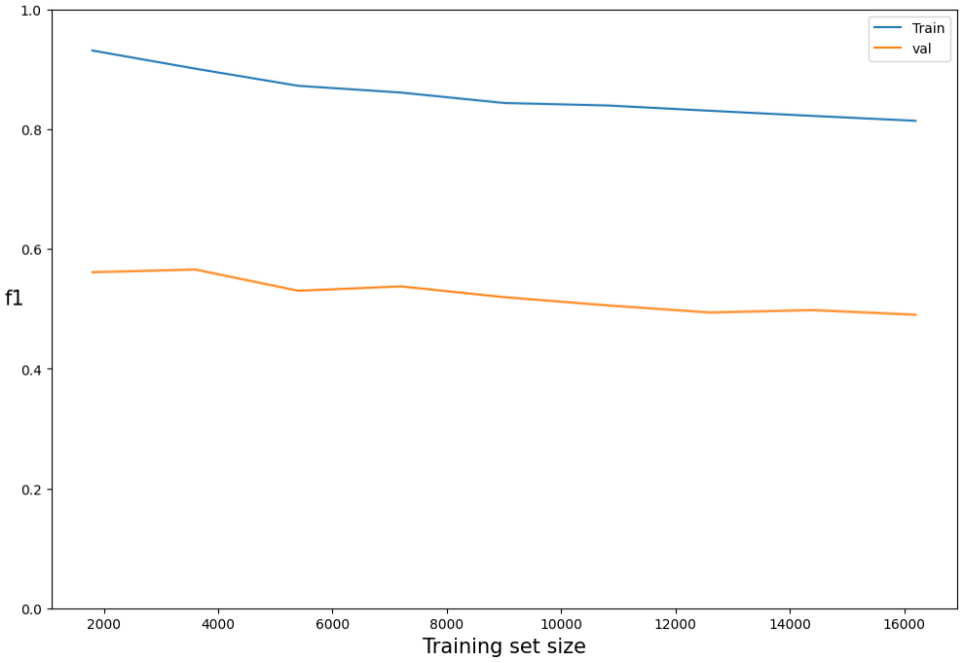
## Learning Curves


The line graphs plot the F1 score as a function of training set size for two machine learning models. The graph includes two lines: one for the training set (Train) and one for the validation set (val). In the graph, both lines start at a training set size of 2000 and extend to 16000. Throughout the range, the F1 scores for both training and validation sets appear to remain relatively stable. The training set F1 score starts slightly higher but seems to converge with the validation score as the training set size increases. This indicates that the addition of more training examples doesn't improve the model performance on training data. The lines relatively converge even for a

small training size (2000 samples), which shows that the model learns early at the training. As the training size increases, the lines converge even more. This means that the models learn effectively. The training error slightly reduces during the iterations, which means that there is no underfit. The fact that the lines converge proves that there is no overfit.

If we tweak parameters, the lines do not converge. Consequently, overfit occurs. This can be seen in the diagram below.

**Multinomial Naïve Bayes**

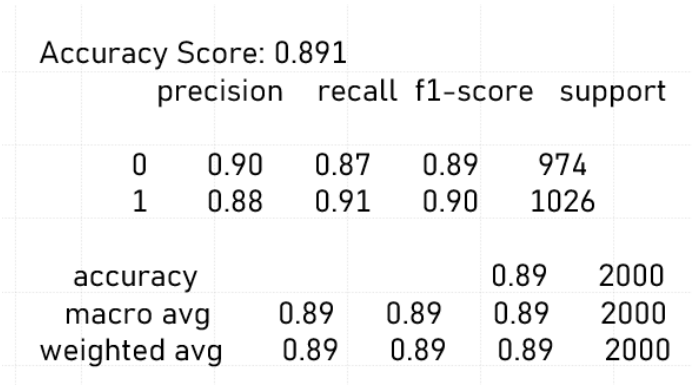



**Parameter hypertuning**

To optimize the greatest performing model, we hypertuned parameters of Multinomial Naïve Bayes. Combining stratified 10-Fold with GridSearch, we found that the optimal values were:

```
Multinomial best parameters:
{'alpha': 0.12, 'fit prior': False, 'force_alpha': True}
```

**Multinomial Naïve Bayes**



Accuracy Score: 0.891

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.90 | 0.87 | 0.89 | 974 |
| 1 | 0.88 | 0.91 | 0.90 | 1026 |
| accuracy | | | 0.89 | 2000 |
| macro avg | 0.89 | 0.89 | 0.89 | 2000 |
| weighted avg | 0.89 | 0.89 | 0.89 | 2000 |

There is approximately 1% improvement in each score of the classification report.

Some hardcoded tests were executed to see if the model generalizes well. As you would expect, the MNB is correct most of time.

**test tweet, label (0: fine, 1: depressed)**

This is insane @ArisTsili. I am not here to fight with you but you are wrong: 1

I am not feeling good: 1

@donaldtrump you made America terrible 😟 : 1

@donaldtrump you made America terrible :((((( : 0

@donaldtrump you made America terrible: 0

@donaldtrump you made America terrible https://t.co/Xk3UOkGYxU: 0

@donaldtrump you made America terrible #giveup: 0

I might kill someone: 1

I need helpppp: 1

YOU ARE THE WORST PRESEDENT EVER @donaldtrump: 1.

**Parameters of the models**

Multinomial naïve bayes: Alpha=1.0, fit prior=False

Supported Vector Machine: C=1.0, kernel='linear', gamma='auto', probability=True

K-Nearest Neighbors: number_of_neighbors=12, weight_type='uniform'

Decision Trees: depth_limit=None, feature_limit=None, split_criterion="gini"

Random Forest: num_trees=10, depth_limit=None, feature_selection="auto", split_criterion="gini"

Logistic Regression: solver='lbfgs',max_iter=10000

f1_score: average='weighted'

CountVectorizer(analyzer='word', ngram_range=(1, 2),token_pattern=r'[^\s]+') # To include emojis and emoticons

vectorizer = TfidfVectorizer(max_df=0.5,sublinear_tf=True,use_idf=True, ngram_range=(1,2),norm = 'l2',decode_error='ignore',analyzer='word',token_pattern=r'[^\s]+')

# 5. Conclusion

Our models are robust and have achieved a high accuracy rate of 89.1% on a dataset consisting of 20,000 labeled tweets. The data analysis was effective in providing a clear understanding of the dataset. We have successfully avoided overfitting or underfitting, as evidenced by the learning curves which show consistent training and validation loss. Cross-validation has been used to verify the performance of our models, and it suggests that the accuracy might actually be underestimated. The confusion matrix indicates a low rate of both false positives and false negatives. ROC curves have been utilized to depict the accuracy for each specific class, which is determined by the area under the curve.

The experimentation proved that pictographs (emojis, emoticons) were useful at improving sentiment analysis. We acknowledge that tagging sentiment can be highly subjective, as it can be influenced by personal experience and the use of irony in language. A proper annotation of the dataset is essential for the models to achieve high accuracy. Attempts to stack classifiers did not yield any improvements in the predictive performance. To further improve performance, we are considering feature engineering strategies, such as incorporating measures of subjectivity and polarity. It would be interesting to see if neural networks can optimize the results even further.