Hate Speech Detection via Machine Learning

Submitted by:

B00105213 Julwin Solis

Submission date: 14/05/21

Project

Submitted in part fulfilment for the degree of

B.Sc. in Digital Forensics and Cyber Security

School of Informatics and Engineering,

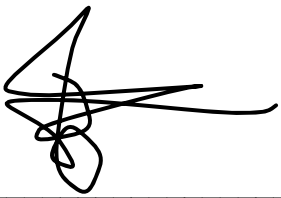
Technological University Dublin – Blanchardstown Campus,

Dublin, Ireland

Declaration

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Degree of **B.Sc. in Digital Forensics and Cyber Security** in Technological University Dublin – Blanchardstown Campus, is entirely my own work except where otherwise stated, and has not been

submitted for assessment for an academic purpose at this or any other academic institution other than in partial fulfilment of the requirements of that stated above.

Author:  Dated: 14/05/2021

Abstract

"Aggressive, hateful and offensive language, aiming at a particular group of people with a common characteristic, namely their sex, race, religion, sex or disability" called Hate speech.
Sadly, hate speech is very prevalent in the world today. The enormous number of messages contaminated with hate speech is an unfortunate reality of social media. It is a real challenge to detect such data on the website.
In this research we focus on hate speech in English language. We intend to automatically detect hateful speech based on the data set.
By training a simple bi - classifier, we tackled the challenge of hate speech detection. Data set was highly imbalanced. We compared the accuracy of the classifiers after train and tested. Multinomial Algorithm give us an accuracy of 0.95 on 4 iteration

Table of Contents

Chapter 1. Introduction

1.1 Background

In today's digital era the main usages of social media are to express emotions, to have common interactions and to communication and share thoughts. These interactions becoming a part of daily life. While we use social media for communication, the use of inappropriate and harmful language cannot be neglected and if left unchecked it can be widely harmful to the users of the platform. The background of this project is to detect hate speech using machine learning techniques. There are various techniques using machine learning algorithms which closely inspect individual comments and its patterns to

recognise and label as normal or negative language. There are various datasets are available on Kaggle and GitHub and are already labelled.

Considering the background of the project hate speech can come in many forms whether it is political, racial, bullying, or abusive language may cause distress to the platform's readers. Furthermore, we will investigate datasets and tweets to testify neutral or negative words on social media platforms such as twitter, Facebook, Instagram.

The idea behind our project is to detect such activities which will need sufficient time to research the topic to get an understanding of how a tweet can be fit into the hate speech category, deciphering whether a tweet is hate speech or just negative speech. Therefore, it is important to have a deeper understanding of the definitions of hate speech.

1.2 Problem Statement/Motivation

There are some challenges we have discovered while researching the project further that may cause issues in the enumeration of our project. The idea behind the project is to implement an algorithm to investigate bulk data/tweets and detect neutral or negative hate speech. For testing running model we must examine the step-by-step process and rely on the hierarchical breakdown of this project.

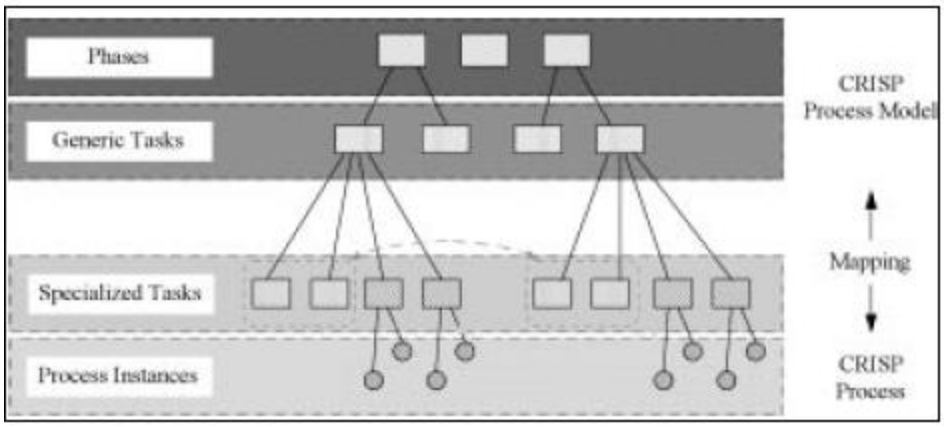
To execute the model, we will adopt the CRISP DM Data mining methodology which consists of multiple tasks commonly described in four sets:

i. Phase

ii. Generic Task

iii. Specialized Task

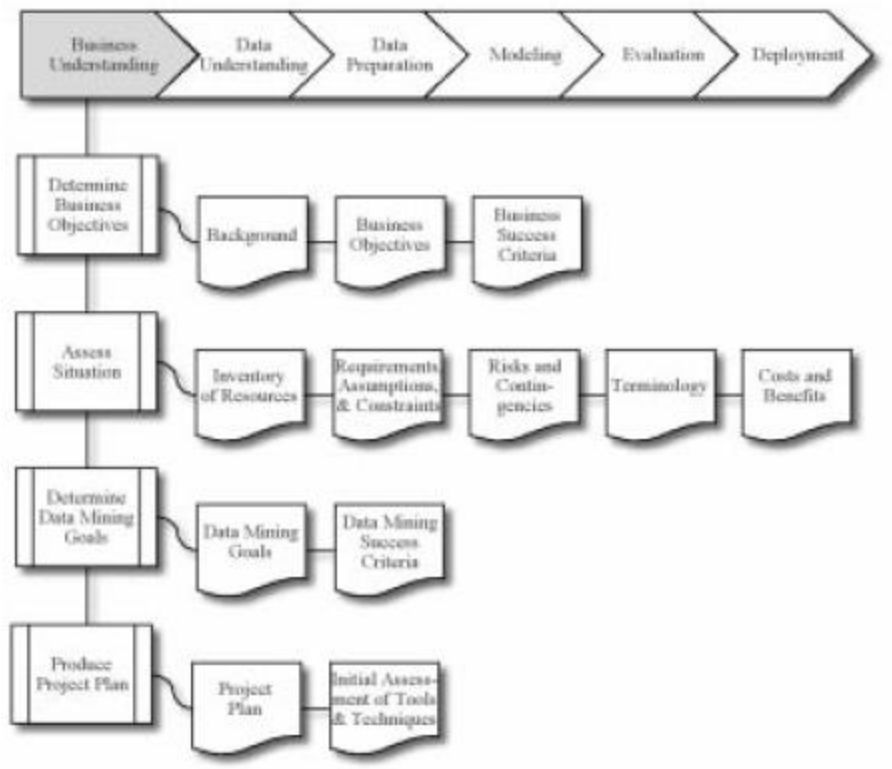
iv. Process instance



Furthermore, the project will aim to match and rank tweets between normal and negative. For data collection we are aiming to trial different data sets for example some datasets from Kaggle website.

Data mapping is one of the challenges we will need to address when analyzing the content and looking into context of the data. We will possibly rename some content for clarity reasons and to get accurate figures on our visualization and word frequencies. To carry out the modelling of our data there are some factors that we must be address for data preparation, modeling, evaluation and deployment.

Here is the understanding model for Crisp Deployment, which further define in detail in Project Aims topic



The above model is based on flow of task to determine business objectives.

1.3 Research Aim(s)

Our Core Project deployment and model flow is based on CRISP-DM Method where the focus will be on research and knowledge to outline positive project results and aim is for to analyse the data, apply tools and techniques with supporting documentation. Let's discuss CRISP-DM Methodology.

In terms of motivation for hate speech detection, there are still plenty of grey areas that need to be covered. A good model can correlate between hate or non-hate language. For example, users may use slangs for humor rather than subjecting someone to hate speech with conventional language. For example, **Fulper et al (2014)** [4], described the co-relation between misogynistic tweets, in each states of untied states of America. Where he tried to explain that social media may be used as a possible carrier of hate speech and disturbance. There is ample amount

of evidence that hate speech may cause distress to their victims which could lead to mental health issues. For example, suicide case of American teenager came to light in 2012 which was in result of hate speech on social media.

(https://www.theguardian.com/commentisfree/2012/oct/26/all)

1.4 Research Objectives

We are going to research the meaning behind hate speech for example hate speech is speech that is intended to insult, offend, or intimidate a person because of some trait as for example race, religion, sexual orientation, national origin, or disability, hate speech is spoken words that are offensive, insulting or threatening to an individual or group based on a particular attribute of that person or persons being targeted. Targeted attributes include such traits as ethnic background, sexual orientation, race, or disability, though there are other target attributes.**Invalid source specified.**

To gain a better understanding of machine learning techniques we are aiming for a program which can detect hate words in conversations.

1.5 Research Methodology Overview

In the past 10 years, we have seen an exponential growth in the number of people using online forums and social networks. Every 60 seconds, there are 510,000 comments generated on Facebook and around 350,000 tweets generated on Twitter. The people interacting on these forums or social networks come from different cultures and educational backgrounds. At times, difference in opinions lead to verbal assaults. Moreover, unchecked freedom of speech over the web and the mask of anonymity that the internet provides insights people to use racists slurs or derogatory terms. This can lower the self-esteem of people, leading to mental illness and a negative impact on society as a whole.

Furthermore, toxic language can take various forms such as cyberbullying which is one of the major factors in suicides in modern times. This issue has shown to be increasingly important in the last decade and detecting or removing such content manually from the web is a tedious task. There is a need of devising an automated model that has the capability to detect such toxic content on the web. In order for us to tackle this issue, firstly we must be able to define toxic language (Hate Speech & Toxic Language). We broadly divide toxic language into two categories: hate speech and offensive language.

1.6 Thesis Overview

The makings of the thesis will be as follows, we will be using labelled data to understand and highlight the concepts of hate speech. We are going to label the data in correlation to binary numbers 0 or 1 which meaning 0 = non-hate speech – language, 1 = hate speech – language. We will be using tweets from Twitter to indicate hate speech. We will be using tweets from any age group, gender, or religion. We will be using tweets from Twitter to indicate hate speech. We will be using tweets from any age group, gender, or religion.**Invalid source specified.**Using hate speech from Twitter we will identify the true nature of hate speech on the labelled data sets. Furthermore, program execute with loading of datasets, with two columns represent first is ‘text’ and other is ‘text type’.

After data loading program focuses on data visualization showing distributions, text information’s, word clouds and bar charts. The next phase is data processing using NLP models, stemming, lemmatizing the words replacing special characters with text. We also training and compare models with four algorithms MultinomialNB, ComplimentNB, LR (Logistic Regression), and Decision Tree for matrix accuracies. Finally, the model is fine tuning and saving the model optimal results using pickle sklearn, for better and accurate future results.

1.7 Summary

Looking at provided literature regarding hate speech includes various topics such as aggressiveness, sarcasm, slangs and misogyny. Most of models are based on two tiers

a. Neutral speech

b. Negative Speech

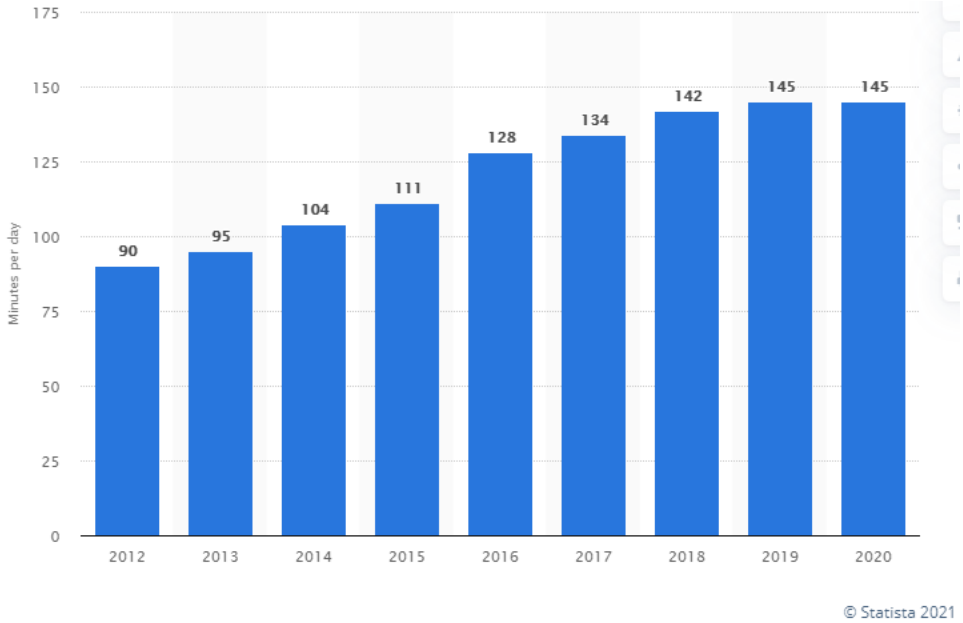
But looking at points (a) and (b) is not suffices in general hate speech detection methods used in many algorithms, also it is getting more difficult to detect hate speech when taking sarcasm in to account, few writers and authors such as Justo et al. (2014) [5] and [6] Nobata etal. (2016), both are aimed to devise a method to interpret sarcasm. Furthermore, the flow of the project is widely discussed with examples on Crisp Method.

Over-all the research will be acquired the data as follows with the data set with the data analysis and labelling of the data then feature the construction and weighting. For the construction model we would implement the projection of the data. Using a classifier, we will use the inputs of the labelled data. The solution will be evaluated whether it is hate-speech or non-hate speech.**Invalid source specified.**

Chapter 2. Literature review

2.1 Introduction

The average internet user spends 145 minutes in 2021 (10.06% of the day) on social media, which was 90 minutes in 2012 (6.25% of the day). In 8 years, social media usage time increased by 55 minutes (Statista, 2021). People share their views, accomplishments, opinions, their lives with family and friends on social media. According to Statista 2021, in Ireland average daily use of the internet is 348 minutes (24.16 % of the day) 109 minutes (7.56 % of the day) dedicated to social media.



Sadly, hate crime is very prevalent in the world today. The popularity of social media is on the rise and it is becoming the heartbeat of today's society. The enormous number of messages contaminated with hate speech is an unfortunate reality of social media. In our Literature review we are aiming to define what hate speech is and review definitions from certain adversary groups such as the ILGA. We will include major social media platforms definitions to review also. With this information we will as a group define our definition of Hate Speech for this project. We will review Hate Speech on Social media further for example the 2019 terror attack in Christchurch, New Zealand, which was streamed and shared on Facebook.

2.2 What is Hate Speech?

1. Hate speech is to incite violence or hate
2. Hate speech is to attack or diminish
3. Hate speech has specific targets

International Lesbian, Gay, Bisexual, Trans and Intersex Association (ILGA) Definition of Hate Speech:

"Hate speech is public expressions which spread, incite, promote or justify hatred, discrimination or hostility toward a specific group. They contribute to a general climate of intolerance which in turn makes attacks more probable against those given groups."

"Language which attacks or demeans a group based on race, ethnic origin, religion, disability, gender, age, disability, or sexual orientation/gender identity."

YouTube's Definition of Hate Speech:

"Hate speech refers to content that promotes violence or hatred against individuals or groups based on certain attributes, such as race or ethnic origin, religion, disability, gender, age, veteran status and sexual orientation/gender identity. There is a fine line between what is and what is not considered to be hate speech. For instance, it is generally okay to criticize a nation-state, but not okay to post malicious hateful comments about a group of people solely based on their ethnicity."

Twitter's Definition of Hate Speech:

"Hateful conduct: You may not promote violence against or directly attack or threaten other people on the basis of race, ethnicity, national origin, sexual orientation, gender, gender identity, religious affiliation, age, disability, or disease."

United Nations Definition of Hate Speech:

"Any kind of communication in speech, writing or behaviour, that attacks or uses pejorative or discriminatory language with reference to a person or a group on the basis of who they are, in other words, based on their religion, ethnicity, nationality, race, colour, descent, gender or other identity factor."

Definition of Hate Speech for Our Project:

Using the combined knowledge, we have gained from researching through multiple policies, definitions and literature we as a group going forward define Hate Speech within the parameters of our project as follows:

“Hate Speech on Social Media is any kind of direct or indirect communication/post that attacks or uses debasing or discriminatory language with reference to a person or a group based on who they are, and or on their religion, ethnicity, nationality, race, colour, descent, gender or other identity factor.”

2.3 Hate Speech and Social Media Platforms

Focus on describing social media platforms and how they can intensify the effects of hate speech.

Hate Speech on social media has come to the forefront of mainstream media over the last few years. There have been groups advocating towards the social media conglomerates to put safeguards in place on their platforms to prevent the use of Hate Speech and the general spread of hate whether that is imagery or text. Companies such as Facebook have been accused of allowing hate on their platforms following the adage of “Any Publicity is Good Publicity” as they benefited from increased usage.

Again, taking Facebook’s structure for example on Facebook users can:

Post a “Status Update” which can be shared ‘Publicly’ which means any logged-on user can view the post. A Post shared ‘Privately’ this means that the user posting can choose the audience for their post this could be their added ‘Friends’ on their profile, ‘Friends of Friends’ or just a select few of chosen users.

Any post can be ‘Reported’ by a user which would send the post in question to be reviewed by a member of staff at the Social Media Company in question.

The aim for our project is to automate this process in the detection of hate speech.

The form of spreading hate speech on Social Media we will focus on for our project will be tweets from Twitter. As explained above most mainstream social media platforms have report and review functions but these are processed over time by staff members.

The Community Empowerment for Progress Organisation (CEPO), Juba & r0g_agency for open culture and critical transformation / Berlin conducted within their studies categorised the drivers of Online Hate Speech as follows:

- Children affected by or involved in violence at increasingly earlier ages - proximity to conflict, learning to hate, breakdown of positive norms/values.
- Youth feel frustration, social media offers an open platform, which can further entrench hate.
- Lack of accountability - no reconciliation, power of anonymity / distance.
- Political / tribal alignments - misconception on cultural diversity
- Lack of policies to guide on online freedom of expression, ethics and privacy

Hate Speech is abusive or stereotyping speech towards a collection of people, primarily based totally on characteristics that include race, religion, sexual orientation and gender. It is illegal based on the present-day regulation with inside America and Europe, but the Internet and social media made it feasible to spread hatred easily, rapidly and anonymously. The massive scale of information produced via social media platforms calls for the improvement of a powerful automated model to discover such content.

2.4 Automatic Approaches to Hate Speech Detection

In this project, we are trying to build a tool that can pick messages for an automatic inspection (if we found any traces of Hate speech). Due to the increase of hate speech on social media, most social media platforms have made changes in their user policy and the methods of implementing these rules. However, they need to increase their labour force to manually review at each content. This processes productivity can be increased using automatic hate speech text detection tools.

Shervin Malmasi Harvard Medical School Boston, and Marcos Zampieri University of Wolverhampton conducted research of Hate speech on social media. In this paper they attempted to detect hate speech to try differentiating it from general offensive language.

Shervin and Marcos thought that the presence of bad and offensive content (bad language) does not qualify for hate speech. The target of bad language is not intended towards individuals but hate speech is targeted towards individuals and or group. Their main aim in this paper is to establish a lexical baseline for discriminating between hate speech and profanity on this standard data set.

They created a data set which contains English tweets with three labels.

1. Hate Speech (Hate Speech)
2. Offensive Language but not Hate speech (Offensive)
3. No offensive content (No offensive content at all)

They showed us the Hate Speech Detection dataset used in their experiment. *They implement a linear Support Vector Machine (SVM) classifier and used three groups of features extracted for these experiments:*

1. Surface n-grams
2. Word skip-grams
3. Brown cluster

They crawled twitter and collected 14509 English tweets which will propagate the dataset. They will then categorize the dataset in 3 different classes mentioned above.

Class	*Texts*
HATE	2399
OFFENSIVE	4836
(contains offensive language but no hate speech)	
NOT OFFENSIVE	7274
(no offensive content at all)	
Total	*14509*

They used a linear Support Vector Machine (SVM) detection data set in their experiment to perform multi-class classification in their experiment. They used LIBLIN EAR classification package which was very efficient for a similar text classification task. They used 10-fold cross validation. They report their results in terms of accuracy. The results are compared against majority class base line and oracle classifier.

Feature	Accuracy (%)
Majority Class Baseline	50.1
Oracle	91.6
Character bigrams	73.6
Character trigrams	77.2
Character 4-grams	**78.0**
Character 5-grams	77.9
Character 6-grams	77.2
Character 7-grams	76.5
Character 8-grams	75.8
Word unigrams	77.5
Word bigrams	73.8
Word trigrams	67.4
1-skip Word bigrams	74.0
2-skip Word bigrams	73.8
3-skip Word bigrams	73.9
All features combined	77.5

They applied a text classification method to differentiate between hate speech and other text. They applied a standard lexical feature and a linear SVM classifier to establish a baseline for this project. The best results were obtained by a character 4-gram model achieving 78% accuracy.

They then used a keyword-based approach to the detect hate speech. A keyword-based approach is relatively easy to grasp, but it has a major disadvantage to its approach also. Their system would not detect certain hate speech which were not using hateful words. For example:

BROWNIE IS FOR MY BROWNIE

This is a racial slur but not using abusive words, brownie is not an abusive word. On the other side including hateful words sometimes are not always hateful for e.g., “trash”, “swine” would create too many false alarms.

In “Detecting Hate Speech on the World Wide Web William Warner and Julia Hirschberg from Columbia University”, they discussed hate speech detection methods which gave us a clear view of potential hate speech detection approaches we could incorporate. Data was collected from the American Jewish Congress (AJC) and Yahoo. They crawled the platform data of texts which was marked as offensive by those sites readers and comes under the hate speech according to their definition. After analysing the data, they divided the information by their stereotype, the Annotators could label a paragraph as Anti-Semitic, anti-black, anti-Asian, Anti-Woman, Anti-Muslim, anti-immigrant, or other hate. They build classifier for every stereotype. For every classifier they used SVM classifier with linear kernel and perform 10-fold cross. In two different tables their accuracy was 96 %.

2.5 Summary

We also reviewed what is hate speech researching past research for definitions and also looked at Social Media Companies like twitters definitions of hateful content within their platform to gain a better knowledge to conduct our research in an informed manner. We as a group devised our definition of Hate Speech for this project in which we will call too for the enumeration of our project.

We reviewed Hate Speech and Social Media as a whole reviewing research papers and previous hypothesis available to us. We reviewed the main causing factors leading to hate speech on social media. We also discussed the flawed system of many Social Media companies which can amplify hateful content. How social media’s anonymity allows people to spread hate with virtually no repercussions.

Chapter 3. Methodology

Our approach for the methodology consists of processing and comparing machine learning classifiers to identify hate speech from textual input i.e. the data set from Kaggle. The dataset will then be presented in a visualized format for further deep analysis using features such as word clouds, pie charts and word frequency. Then we will move on to preprocessing the text from the dataset to bring it along to next phase of training model along with data tuning and comparing data between different models. We will then formulate our data that will be saved into a file for making future predictions accurately. We will take into consideration the appropriate results from the program that is relying on the TF-IDF vectorising data. This will make good use of the neural networks as these are better performing methods than other methods commonly used.

3.1 Keyword-based approaches

A keyword-based approached is considered as the natural approach by applying text using ontology or dictionary where potential hate words are identified such as Hatebase. In general, keyword-based approaches are not considered as an incorporable approach towards detecting hate speech, although in many instances this approach is considered as a much more straightforward in terms of implementation, detection and additionally using terms to detect hate speech such as (pet, rubbish, hairy, brown skin) not always considered as hate words. Furthermore, if you consider slangs using keyword-based approach is bit harder to distinguish, for example “Build the barrier” considered as simple slang for building a wall, but in USA this is considered as condemn towards the immigrants.

3.2 Metadata Characteristics and Identification

Using Social Media as a guide will aid in our combined effort to understand the characterization of feeds will be leading towards the advancement and accurate

identification approach to detect hate speech. But there are few obstacle's with this such as supply and understanding of the data on a granular level. This a difficult process as social media platforms are very reluctant to provide that information with a very high-level detail.

3.3 TF-IDF Algorithm

TF-IDF algorithm is commonly used for its feature The Generation model and is also classified as "Custom Word Embedding model" or "Bag-of-Words" model, mentioned below are the acute steps to describe TF-IDF:

$$tf_{t,d} = \begin{cases} 1 + \log_{10} \text{count}(t, d), & \text{if count}(t, d) > 0. \\ 0, & \text{otherwise.} \end{cases}$$

Reference to above "*tft,d"* is considered as computing term frequency and "*t"* considered as the text document

Compute the inverse document frequency idf_t:

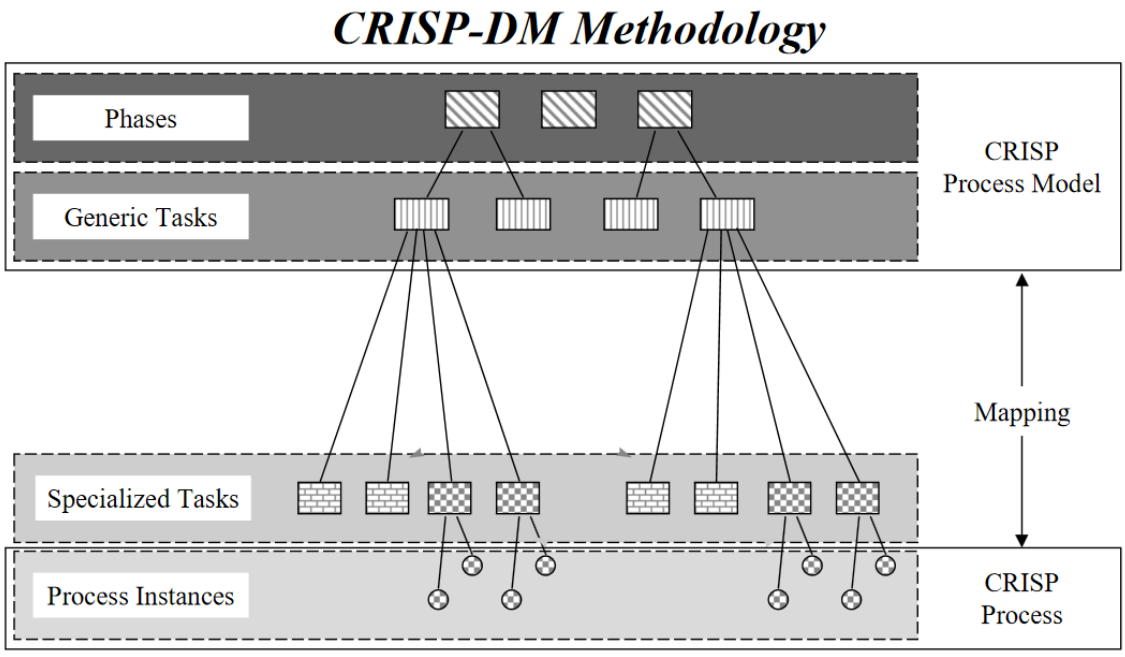
$$idf_t = \log_{10}\left(\frac{N}{df_t}\right),$$

Referenced to above *"dft"* is estimating sum of documents considered as *"t"* (text Document) and *"N"* is referred as sum of total documents in corpus.

$$tf\text{-}idf_{t,d} = tf_{t,d} \times idf_t.$$

Finally, the frequency is multiplied by inverse document frequency

3.4 CRISP-DM Methodology



The CRISP Model consists of four consecutive sets of tasks which contain phases, generic tasks, specialized task process instances.

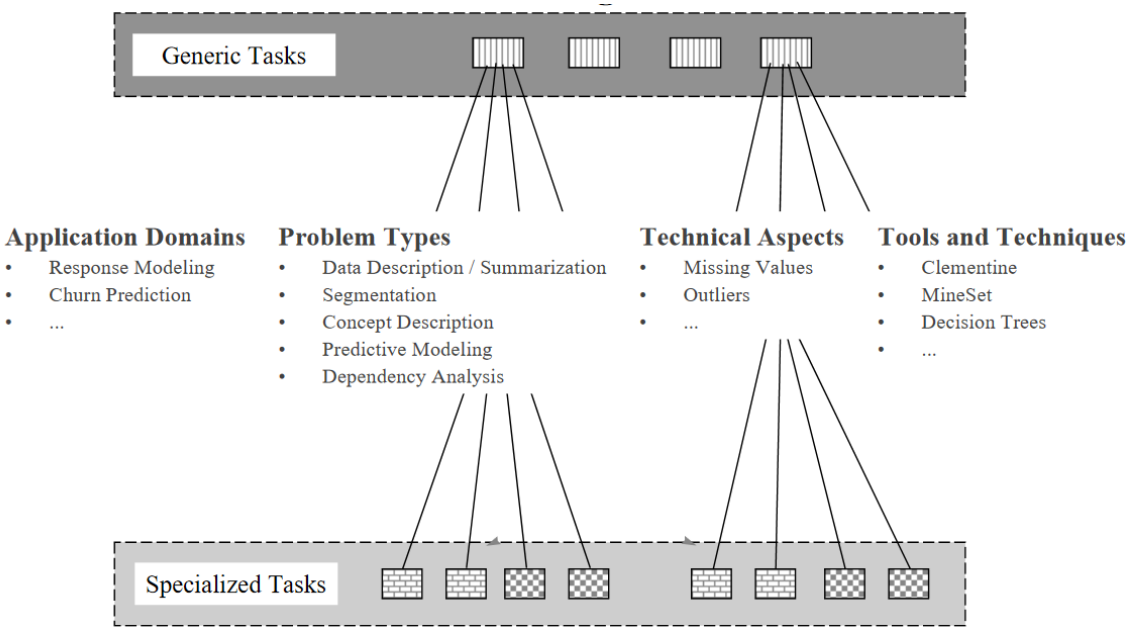
In the above model, the top level is the organization level that will address the data mining process. Each phase is relying on generic multiple second level tasks as mentioned named as generic that is pointing toward the tasks that must be covered in general to address data mining possibilities. The idea behind the generic task is to execute tasks as sturdy as possible.

The third Level Specialized Tasks is aiming for tasks which are executing on Generic Tasks to be determined in particular circumstances, for example in the second task code is aims for normalizing data to proceed.

The final and fourth level's aim is to record the actions on instances considering results extract from data mining. This level also represents what actually happened to the task when executed.

In undermentioned model data mining contexts are defined in a flow.

Data Mining Contexts



- In Application Domains, the domain is specific to data mining, response modeling and churn predictions.

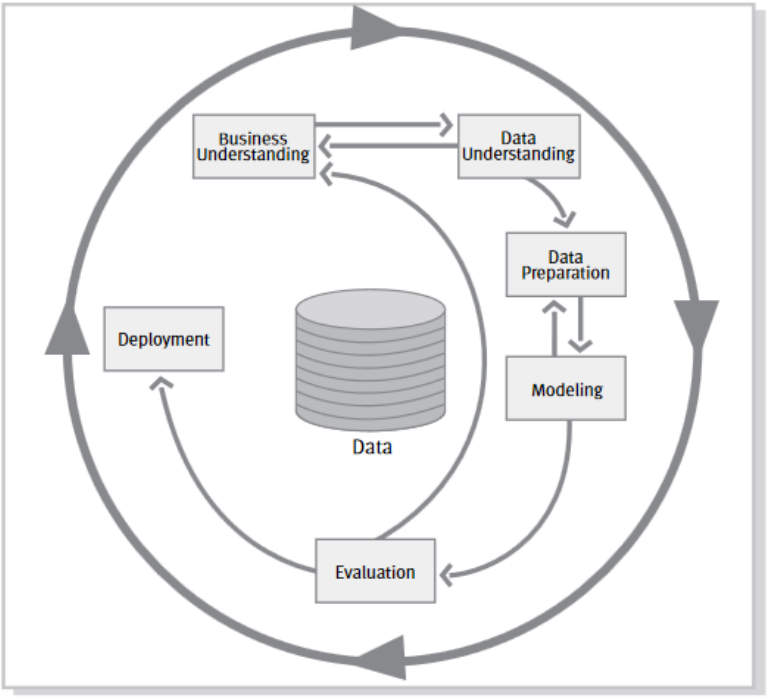
- The Problem Types section has been outlined for specific classes, objectives of data description and its summarization. However, the data will be segmented and its concept of predictive modelling and dependency analysis.

- The Technical Aspects section is covering the technical challenges which mostly happened on data mining instances where there where commonly overlooked missing values and outliers.

- The Tools and Techniques sectioned is aimed for data mining tools during project execution and looked at mining set and decision trees

Referenced Model (CRISP-DM)

In The below Referenced Model which is based on the overview of the project life cycle the model is focused on phases of its outlines the tasks and in-between its relationship.



Business Understanding:

In this phase of the model, the focus is on the model objective and the necessary requirements needed to accomplish the projects aims and its solutions for data mining issues.

Data Understanding

This phase starts with the initial data collection. For example here datasets from Kaggle are referenced as data collection and further investigates the quality of data and their subsets forming.

Data Preparation

This phase mostly focuses on data construction and finalizing data to be present in the data preparation task model. This model aims for table, records and attribute selection as well as cleansing of data

Evaluation

In this phase the project data gathered is considered as high value analytical data well before initiating into final deployment model.

Deployment

In this part of the project model, it will aim towards presenting of data in a way that the algorithm can use that data to judge the objectives such as hate speech wording. This also is where the project model decides between neutral and negative language approach.

3.5 Classifiers:

We used number of commonly used classifiers such as, “MNB” (Multinomial Naïve Bayes), “CNB” ComplimentNB, “LR” (Logistic Regression), “DTC” (Decision Tree Classifier) AdaBoost, all of these classifiers are used to trained machine learning model, furthermore “sklearn” is used to evaluate classifiers mentioned as follows:

3.5.1 “MNB” (Multinomial Naïve Bayes)

“MNB” (Multinomial Naïve Bayes) is considered as a probabilistic model heavily considered as successor version of Naïve Bayes models for best optimal results and text classification. MNB interestingly works on ‘Bayes’ theorem to predict text from individual tags and the words occurrences in dataset, for example detecting hate words and their frequency of occurrences and discrete count of words from Social media such as twitter and Facebook , and Instagram. Bayes theorem is introduced by Thomas Bayes, and algorithm works on probability calculations of occurring instances and execute based on undermentioned formula:

P(AIB)=P(A)*P(BIA)/P(B)

In above formula we are making calculation of A and B where data to predict is already provided in ‘B’, looking at data set we are multiplying likely hood of data sets such as:

P (hate speech) # prior

*P (disgusting | hate speech)

*P (redirected | hate speech)

*P (allowed | hate speech)

Also we take consideration of words have ‘0’ probability * 0 = 0 such as word ‘city’ may consider as 0 probability, where the model focusing only on the presence and the absence of the words, the reason to choose the “MNB” against the NB model is the advancement where model is focusing on word count and the words occurrence calculations. The good advantage of using MNB Model is to train machine learning model fast and effectively. Also MNB is trained on same TF-IDF offered features, furthermore values are tested on hyper-parameter as well as combining word n-gram and character n-gram for the best results and accuracy in our dataset.

3.5.2 ComlimentNB Algorithm

ComplimentNB works on CNB algorithm, and is considered as a simple and standard version of ‘MNB’ as discussed in 3.51 section, where this algorithm is best worked on imbalanced datasets, but unlikely ‘CNB’ handles statistic of each class compliment and measuring the models weightage. And also calculating certain class probability below is example of calculating the model weights

$$\hat{\theta}_{ci} = \frac{\alpha_i + \sum_{j:y_j \neq c} d_{ij}}{\alpha + \sum_{j:y_j \neq c} \sum_k d_{kj}}$$

$$w_{ci} = \log \hat{\theta}_{ci}$$

$$w_{ci} = \frac{w_{ci}}{\sum_j |w_{cj}|}$$

On above weightage example ‘j’ is considered as not in class, but ‘c,d,I,j’ may responsible for summing/count and ‘tf-idf’ is refered to value ‘i’ in the document where ‘j’ and α_i is related to stable hyperparameter similar to ‘MNB’ model. Also on other side $\sum_i \alpha_i$ works for normalization and for the tendency of large documents

3.5.3 “LR” (Logistic Regression)

Logistic Regression is considered as a statistical model which transforms its output into probability where it’s values are mapped between two or more classes. “LR” (Logistic Regression), is also analysing regressively on dependent variables in binary. Our model is trained by the LR on a same trained approached used in SVC for the higher accuracy. In general Logistic regression model is based on equation where input values (x_lables) are merged for weight and for coefficient values to accurately predict the value of (y_labels) and the output value is delivered binary as ‘0 or 1’ based on sigmoid equation mentioned below:

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

Sigmoid function equation is defined as ‘1’ divided ‘1’ plus ‘e’ power ‘-z’ and ‘e’ is refereed to Euler’s numbers, in above equation ‘1’ is divided by value which is slightly greater than 1 and when applying this outcome will be less than ‘1’ as referenced above sigmoid function coverts input into binary range 0 to 1 for best accuracies and results.

3.5.4 DT Decision Tree Model

We use the Decision model to check accuracies and comparison between the Logistic regression model which mainly split dataset by just putting a single line, due to data set imbalance we used Decision tree model to split dataset again and

again to come up with decision boundaries, generally decision tree consider as ‘tree’ which starts from the root and branches out number of possible solutions and root is keep growing with decisions and conditions. In this model dataset is divided into two subsets ‘true’ or ‘false’ consider as hate or non-hate, which added into two child node to produce purest subsets on each nodes

3.6 Overview of Present Approach

In this model contains the whole process of creating and comparing machine learning classifiers to identify the hate speech from textual input. We will be starting off with the data visualization, then we will pre-process the text. From here we will move on for model training and tuning. After training we will then devise a comparison between the different models. Then finally we will save our model in the disk for making predictions in future. Here are the steps that will be following to achieve the work that needs to be done:

3.7 Dataset loading:

We started off with reading data set. We will make a single data frame of data set. The data set will have two columns, one column will represent text and the other will represent the text type. Code of loading data set is explained below:

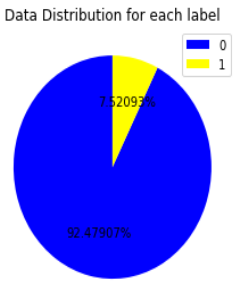
```
#importing warning to igonore warning message
import warnings
warnings.filterwarnings("ignore")

#importing pandas to load dataset
import pandas as pd
df= pd.read_csv("train.csv",encoding='latin-1')

#Labeling Tweets, text
df=df[['label','tweet']]
df.columns=["Target","text"]
df.head()


#defining hate and non hate speech
no_hate=df[df['Target']==0]
hate=df[df['Target']==1]
```

In above code ‘Pandas’ library is used to read and labelled data sets where text, and target columns are defined, regarding target dataset with labelled ‘0’ considered as no hate, and datasets labelled with 1 considered as hate

3.8 Data visualization: (Pie Charts)

2) We then did some data visualization that includes showing the data distribution, text information, word clouds, pie charts, and bar charts.

```
### Pie chart showing the class distribution in data

import matplotlib.pyplot as plt

class_names = (df['Target'].unique()) ## This line will store the name of the labels, since in our case
## labels are stored in the form of numbers i.e 0 for non hate , 1 for hate speech
## So this variable will store only 0 and 1


## This variable will store the occurance of 0 and 1 in our data set
values_of_classes = list(df['Target'].value_counts())


colors = ['blue', 'yellow', 'brown', 'black']
patches, texts,per = plt.pie(values_of_classes,autopct='%0.5f%%', colors=colors, startangle=90)
plt.legend(patches, class_names, loc="best")


plt.title('Data Distribution for each label')
plt.show()
```



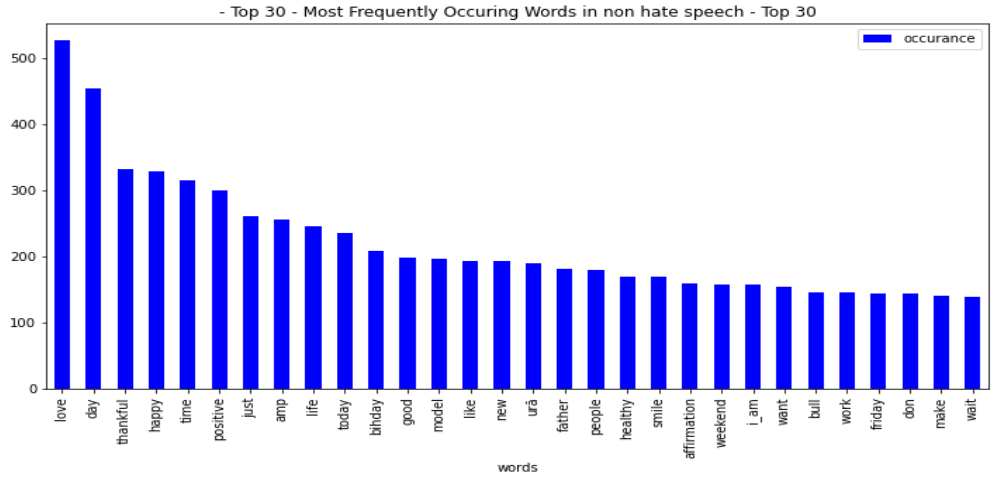
3.9 Data visualization: (Bar Chart frequent non hate words)

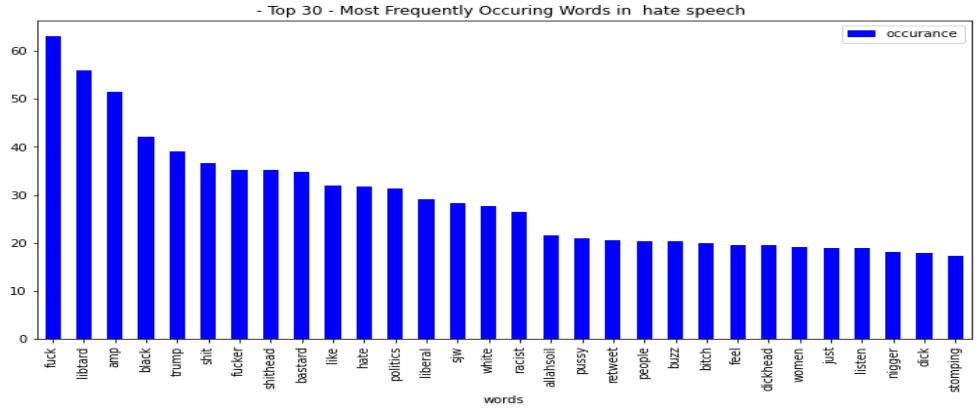
In below code we used TF-IDF vectorizer from sklearn library in order to prepare data for bar chart, where we use bag of words approach and convert text into individual words, approached is defined screenshot below:

```
#Taken from
#https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html#

from sklearn.feature_extraction.text import TfidfVectorizer

#setting stop words
vectorizer = TfidfVectorizer(stop_words = 'english')

from wordcloud import WordCloud
no_hate['text'] = no_hate['text'].str.replace('@user',' ')

#Getting vectors for each word of text
words = vectorizer.fit_transform(no_hate.text)

#Summing the vectors across the rows
sum_words = words.sum(axis=0)

#Getting the frequency of each word
words_frequency = [(word, sum_words[0, i]) for word, i in vectorizer.vocabulary_.items()]

#Sorting the words with in ascending order
words_frequency = sorted(words_frequency, key = lambda x: x[1], reverse = True)

#Making a dataframe of the occurances
occurance = pd.DataFrame(words_frequency, columns=['words', 'occurance'])

occurance.head(30).plot(x='words', y='occurance', kind='bar', figsize=(12, 6), color = 'blue')
plt.title("- Top 30 - Most Frequently Occuring Words in non hate speech - Top 30")
```

Frequency of hate words are defined as below:

```
## Replacing the word @user
## Refernece : https://stackoverflow.com/questions/28986489/how-to-replace-text-in-a-column-of-a-pandas-dataframe

hate['text'] = hate['text'].str.replace('@user',' ')


## Getting vectors for each word of text
words = vectorizer.fit_transform(hate.text)
## Summing the vectors across the rows
sum_words = words.sum(axis=0)

## Getting the frequency of each word
words_frequency = [(word, sum_words[0, i]) for word, i in vectorizer.vocabulary_.items()]
## Sorting the words with in ascending order
words_frequency = sorted(words_frequency, key = lambda x: x[1], reverse = True)
## Making a dataframe of the occurances
occurance = pd.DataFrame(words_frequency, columns=['words', 'occurance'])

occurance.head(30).plot(x='words', y='occurance', kind='bar', figsize=(12, 6), color = 'blue')
plt.title("- Top 30 - Most Frequently Occuring Words in  hate speech ")
```



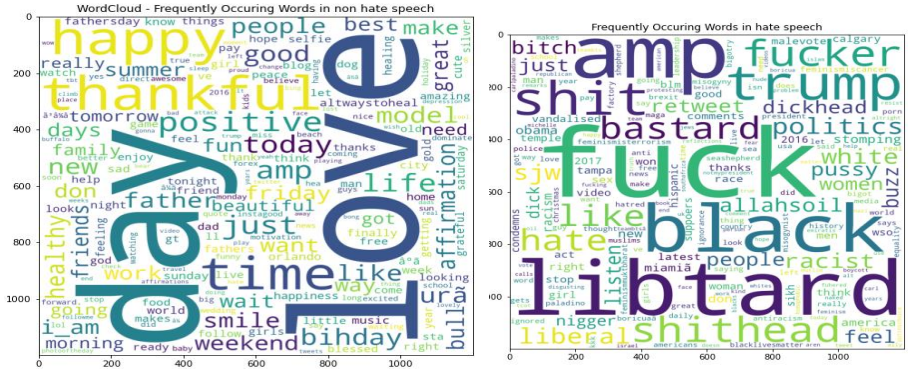


Frequency of hate words as defined in wordcloud (non hate /hate):

```
# non hate word cloud
wordcloud = WordCloud( width = 1200, height = 1200,background_color = 'white').generate_from_frequencies(dict(words_frequency))

plt.figure(figsize=(10,8))
plt.imshow(wordcloud)
plt.title("WordCloud - Frequently Occuring Words in non hate speech", fontsize = 12)
```

```
# hate words frequencies defined in wordcloud
wordcloud = WordCloud(background_color = 'white', width = 1200, height = 1200).generate_from_frequencies(dict(words_frequency)

plt.figure(figsize=(10,8))
plt.imshow(wordcloud)
plt.title("Frequently Occuring Words in hate speech", fontsize = 12)
```

On above screen shot frequencies of non-hate words are defined in word cloud using ‘dict’ feature results of wordcloud are displayed:



3.10 Data pre-processing:

Data pre-processing plays an important role in NLP models. Since there are some characters/words that can hinders the model performance. So, our data processing includes removing special characters, lemmatizing the words, replacing emojis with text, also punctuations are removed for better accuracies, converting two letter repetition e.g. ‘funnnnny’ to ‘funny’, and validation of tweet to check if it is beginning withalphabet as mentioned below in detail:

```
def preprocess_tweet_word(tweet_word):
    """
    This function will remove the repitition of words and ',- symbols from the text of tweets
    """
    # Remove punctuation
    tweet_word = tweet_word.strip('\'"?!,.():;')
    # Converting more than two letter repetitions to two letter
    # funnnnny --> funny
    tweet_word = reg.sub(r'(.)\1+', r'\1\1', tweet_word)
    # Remove - & '
    tweet_word = reg.sub(r'(-|\')', '', tweet_word)
    return tweet_word


def is_valid_tweet_word(tweet_word):
    ## Reference :https://stackoverflow.com/questions/27198984/regex-match-if-strings-starts-with-a-letter-or-number-and-then
    # Check if tweet_word begins with an alphabet
    return (reg.search(r'^[a-zA-Z][a-z0-9A-Z\._]*$', tweet_word) is not None)
```

```
def preprocess_text_of_tweet(text_of_tweet,use_stemmer=False):
    """
    This function will remove the URLS , mentioins , hashtags in the text
    """
    processed_text_of_tweet = []
    # Convert to lower case
    text_of_tweet = text_of_tweet.lower()
    # Replaces URLs with the tweet_word URL
    text_of_tweet = reg.sub(r'((www\.[\S]+)|(https?://[\S]+))', ' URL ', text_of_tweet)
    # Replace two+ dots with space
    text_of_tweet = reg.sub(r'\.{2,}', ' ', text_of_tweet)
    # Strip space, " and ' from text_of_tweet
    text_of_tweet = text_of_tweet.strip(' "\'')
    # Replace emojis with either EMO_POS or EMO_NEG
    text_of_tweet = replacing_emoji(text_of_tweet)
    # Replace @handle with the tweet_word USER_MENTION
    text_of_tweet = reg.sub(r'@[\S]+', 'USER_MENTION', text_of_tweet)
    # Replaces #hashtag with hashtag
    text_of_tweet = reg.sub(r'#(\S+)', r' \1 ', text_of_tweet)
    # Remove RT (retext_of_tweet)
    text_of_tweet = reg.sub(r'\brt\b', '', text_of_tweet)
    # Replace multiple spaces with a single space
    text_of_tweet = reg.sub(r'\s+', ' ', text_of_tweet)
    tweet_words = text_of_tweet.split()

    for tweet_word in tweet_words:
        tweet_word = preprocess_tweet_word(tweet_word)
        if is_valid_tweet_word(tweet_word):
            if use_stemmer:
                tweet_word = str(porter_stemmer.stem(tweet_word))
            processed_text_of_tweet.append(tweet_word)

    return ' '.join(processed_text_of_tweet)
```

```
def preprocess_csv(lines,use_stemmer=False):
    """
    This function will make a data frame for the processed data
    """
    total = len(lines)
    pre_processed=[]
    for i in range(total):
        pre_processed.append(preprocess_text_of_tweet(lines[i]))

    return pre_processed
```

3.11 Training and comparing the model's performance:

We then converted the text into numerical data using tf-idf vectorizer and trained our machine learning models on the vectorized data.

For the evaluation of the classifier built we used the metric accuracy that was tested on different folds using k-fold cross validation. And then we plot the confusion matrix and an accuracy plot.

```
## Guide for performing cross validation was taken from this reference :
## https://my.oschina.net/Bettyty/blog/751627

def k_fold_cross_validation(training_columns, labels, vectorizer, classifier, name_of_model):

    kf = KFold( shuffle=True,n_splits=10) # Initializing KFold object with 10 as k-value
    accuracies=[]
    training_columns_train_array = []
    labels_train_array = []
    training_columns_test_array = []
    labels_test_array = []
    iter = 0
    print("Applying K fold cv algorithm for :", (name_of_model))
    for training_indices, test_indices in kf.split(training_columns):
        iter += 1
        print("iter ", iter)
        training_columns_train_cv, labels_train_cv = training_columns.iloc[training_indices], labels.iloc[training_indices]
        training_columns_test_cv, labels_test_cv = training_columns.iloc[test_indices], labels.iloc[test_indices]

        # adding training score for  the given iteration
        training_columns_train_array.append(training_columns_train_cv)

        # adding test score for iteration
        training_columns_test_array.append(training_columns_test_cv)

         # adding training labels for  the given iteration
        labels_train_array.append(labels_train_cv)

        # adding test labels for  the given iteration
        labels_test_array.append(labels_test_cv)

        # learning vocabulary of training set
        vectorizer.fit(training_columns_train_cv)
        training_columns_train_vectorizer_cv = vectorizer.transform(training_columns_train_cv)
        print("Shape of training training_columns: ", training_columns_train_vectorizer_cv.shape)
        training_columns_test_vectorizer_cv = vectorizer.transform(training_columns_test_cv)
        print("Shape of test training_columns: ", training_columns_test_vectorizer_cv.shape)
        classifier.fit(training_columns_train_vectorizer_cv.toarray(), labels_train_cv)

        # Calculating accuracy
        score = classifier.score(training_columns_test_vectorizer_cv.toarray(), labels_test_cv)
```

```
        # adding k-fold  accuracy for each iter
        accuracies.append(score)
    print("List of k-fold  accuracies for {}: ".format(name_of_model), accuracies)
    average_accuracy = np.mean(accuracies)
    print("Average k-fold  accuracy for {}: ".format(name_of_model), average_accuracy)
    print("Best k-fold  accuracy for {}: ".format(name_of_model), max(accuracies))

    # best k-fold  accuracy
    maximum_accuracy_index = accuracies.index(max(accuracies))

    # training training_columns corresponding to best k-fold  accuracy
    maximum_accuracy_training_columns_train = training_columns_train_array[maximum_accuracy_index]

    # test training_columns corresponding to best k-fold  accuracy
    maximum_accuracy_training_columns_test = training_columns_test_array[maximum_accuracy_index]

    # training labels corresponding to best k-fold  accuracy
    maximum_accuracy_labels_train = labels_train_array[maximum_accuracy_index]

    # test labels corresponding to best k-fold  accuracy
    maximum_accuracy_labels_test = labels_test_array[maximum_accuracy_index]
```

```
# Prediction on test training_columns
labels_pred = clf.predict(maximum_accuracy_training_columns_test_vectorizer)

## Plotting confusion matrix is taken from :
## https://medium.com/@dtuk81/confusion-matrix-visualization-fc31e3f30fea
cf_matrix = confusion_matrix(maximum_accuracy_labels_test, labels_pred)
d={ 0: 'Non Hate', 1: 'Hate'}
sentiment_df = labels.drop_duplicates().sort_values()

group_names=["Correct Non Hate","Incorrect Hate","Incorrect Non Hate","Correct Hate"]

group_counts = ["{0:0.0f}".format(value) for value in
            cf_matrix.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in
                     cf_matrix.flatten()/np.sum(cf_matrix)]
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)


sns.heatmap(cf_matrix, annot=labels, fmt='', xticklabels=sentiment_df.values, yticklabels=sentiment_df.values)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title("Confusion Matrix (Best Accuracy) - {}".format(name_of_model))
plt.show()
return
```

3.12 Fine tuning and saving the model:

We will then fine tune our one of the models and saved it in the disk using pickle so in future we can make predictions using the saved models.

```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import reciprocal, uniform
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(tfidf.fit_transform(data), targets, test_size=0.1, random_state=101)

## Hyper-parametertuning
param_distributions = { "alpha": uniform(1, 10), "fit_prior":[True,False]}
rnd_search_cv = RandomizedSearchCV(NBC_clf, param_distributions, n_iter=10, verbose=2, cv=3)
rnd_search_cv.fit(X_train, y_train)
```

3.13 Dataset Description (David)

The chosen dataset has been chosen using Kaggle for this project. Our chosen Dataset is divided into three columns ID, Label and tweet. The ID column attaches a numeric value to the row for each tweet which increments with every new line. The label column identifies if the given tweet was deemed to be hate speech or not hate speech with 0 representing non hate speech and 1 representing tweets that were deemed to be hate speech. Using python, we have easily counted how many of each tweet were deemed to be hate speech and how many are defined as non-hate speech.

```
[4]: data['label'].unique()

[4]: array([0, 1])

[5]: data['label'].value_counts()

[5]: 0    29720
     1     2242
     Name: label, dtype: int64
```

Shown in the image above we imported the dataset and then using the 'label' column we filtered between hate and non-hate speech using .unique(). After this and confirming there was only 2 variables 0 & 1 we used .value_counts() to count the amount of tweets which had been labeled as hate speech and non-hate speech respectively. From this we learned there is 2242 thousand instances of hate speech within the dataset of 31,962 tweets.

3.14 Initial Cleaning Visualisation and Data Understanding (David)

The text normalization of the dataset we have used is as follows using Python3:

Using pandas we used the pd.read the csv file into python for us to clean our dataset. For our preprocess method of cleaning we used imported preprocessor and used the function p.clean and specified the tweet row to do a preclean of the data removing @'s etc shown below.

```
def preprocess_tweet(row):
    txt = row['tweet']
    txt = p.clean(txt)
    return txt
```

```
data['tweet'] = data.apply(preprocess_tweet, axis=1)
```

We first using the label column separated the hate speech and non-hate speech rows into two separate variables shown below.

```
[5]: data['label'].value_counts()

[5]: 0    29720
     1     2242
     Name: label, dtype: int64

[6]: hate_rows = data[data['label'] ==1]
     hate_rows.shape

[6]: (2242, 3)

[7]: non_hate_rows = data[data['label'] == 0]
     non_hate_rows.shape

[7]: (29720, 3)
```

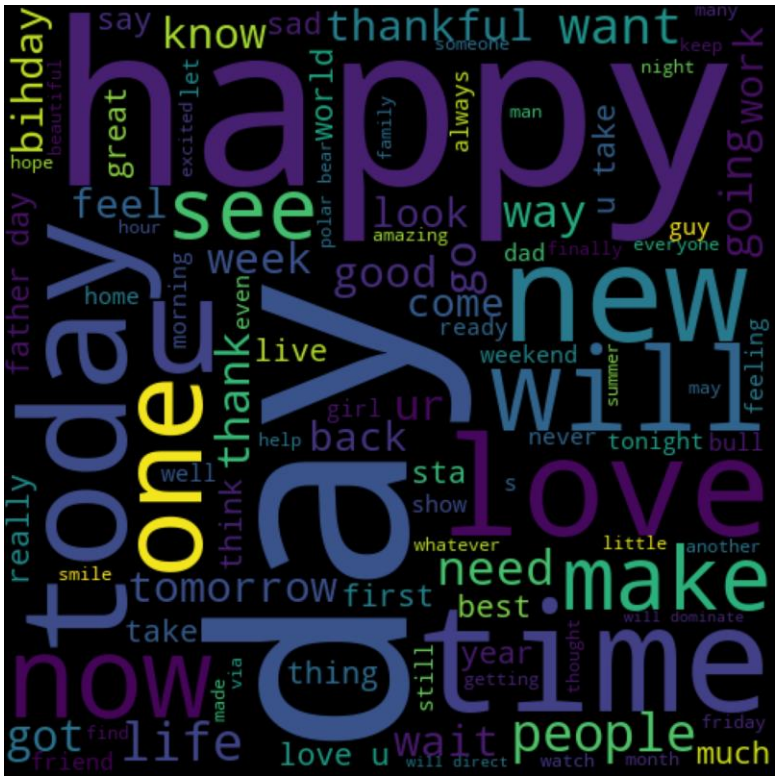
This then allowed us to extract the tweets as a string listed into the hate_text and non-hate_text respectively to further clean up the datasets. We then removed all extra spaces between each tweet to remove the spaces between each tweet shown below.

```
[11]: hate_text = ' '.join(hate_text)
      hate_text
```

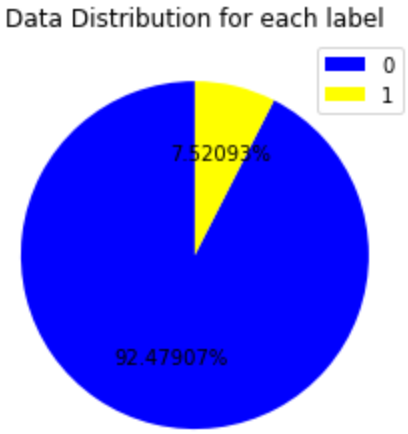
We then created a word cloud to compare and contrast the frequency of words used in hate speech compared to non-hate speech. Shown below.

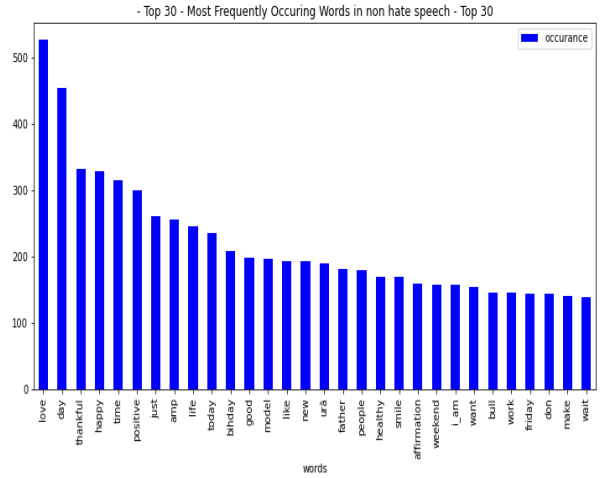
We where forced to manually remove words such as @user and amp from the following word clouds as part of our preprocessing of the tweets.

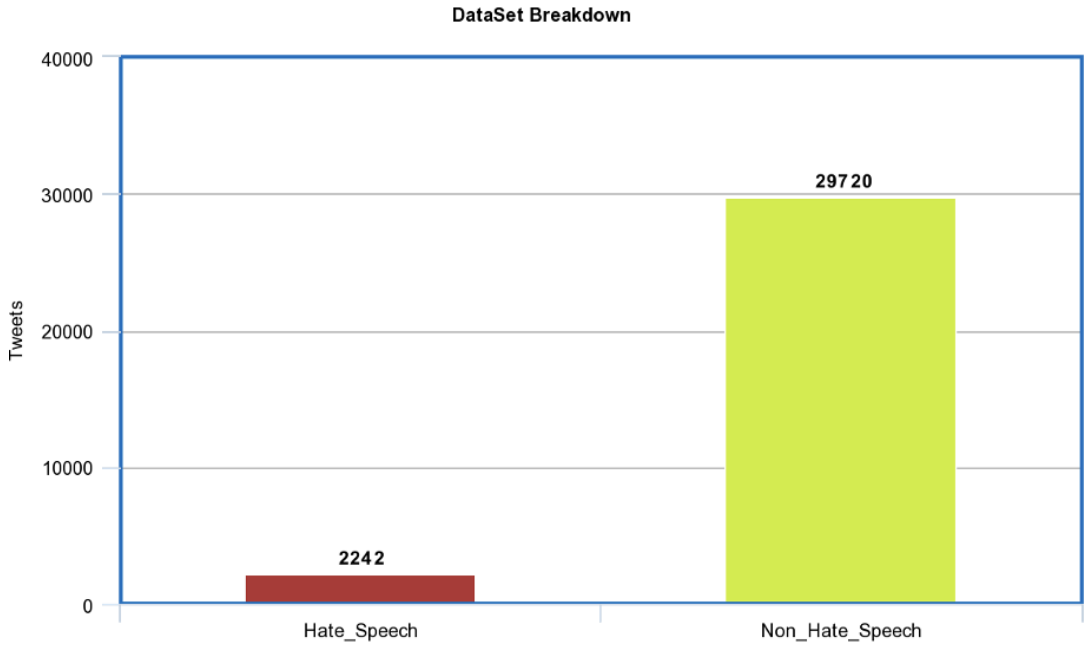
Hate Speech WordCloud



Non-Hate Speech WordCloud

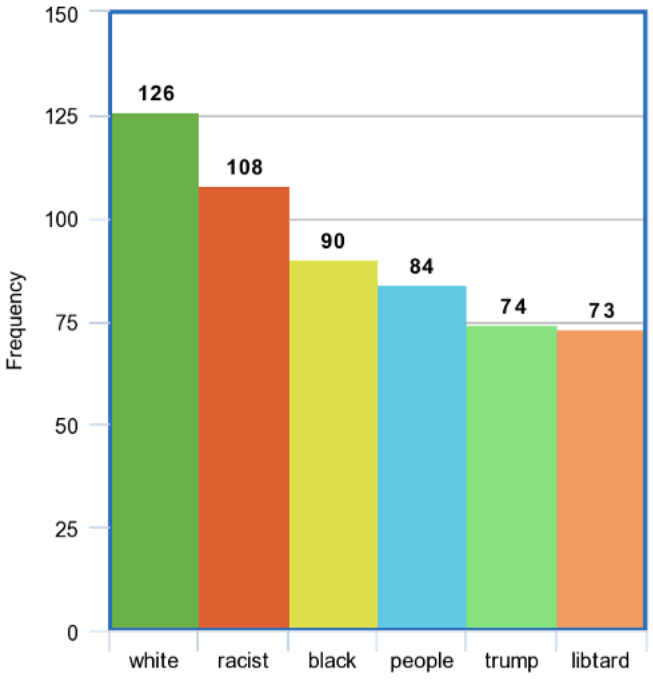






Hate Speech Keyword Frequency's

Primary Keywords	Frequency
white	126
racist	108
black	90
people	84
trump	74
libtard	73
women	67
racism	66



3.15 Baseline Design

In this section we will discuss the algorithms which we will use in this paper for classifying the dataset. All classifiers go through the two phases, which are pre-testing and testing. Here, we are going to use three classification algorithms which are as follows:

- Logistic Regression
- Support Vector Machine (SVM)
- Naïve Bayes

The main goal here is to compare between classification results and see if there are major performance differences.

Logistic Regression

Logistic regression is simple and commonly used for text classification. It is described as a statistical model. Initially it was designed to solve binary classification. Because, LR has a lower model complexity, it can get good results. To separate the class function it uses logistic function. The output of LR act as a probability that the given sample belongs to positive class.

$$p = \sigma (\theta T (x))$$

Naïve Bayes (NB) Classifier

The Naive Bayes classifier is a probabilistic machine learning model. It is a simple and fast algorithm that, despite its simplicity, often performs admirably.

The Naive Bayes classifier is based on Bayes' Theorem. The theorem can be used to estimate the probability that a sample belongs to a certain class in the context of a classification problem**.** The theorem can be used to estimate the probability that a sample belongs to a certain class given the sample's collection of feature values in the context of a classification problem.

$$P(y|x_1, x_2, ..., x_n) = \frac{P(y)P(x_1, x_2, ..., x_n|y)}{P(x_1, x_2, ..., x_n)}$$

Support Vector Machines

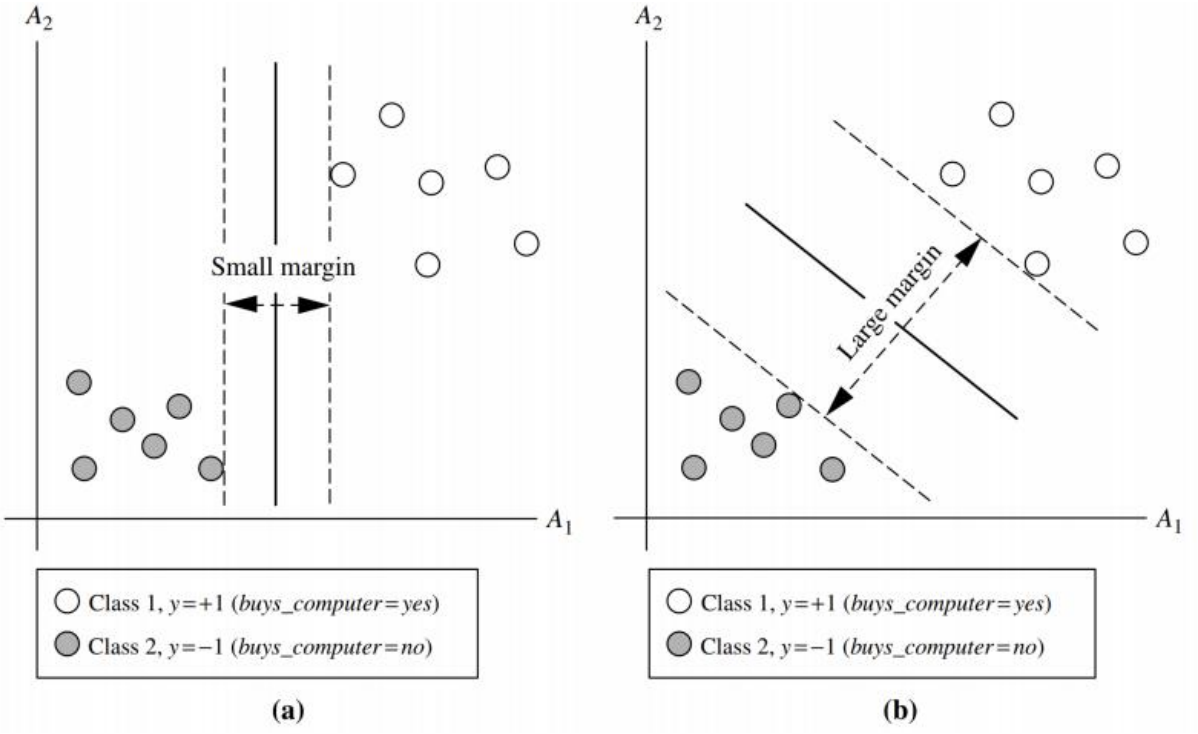
Support Vector Machines are Machine Learning Models that applied for text classification. It uses hyperplanes to find that maximum margin between the classes. The classes defining the hyperplane are called support vectors.

$$\vec{x} : f(x) = \vec{w} \cdot \vec{x} + b = 0$$

The data points with the shortest distance to the hyperplane on both sides are the real support vectors. The Figure depicts the problem geometrically. (b) depicts a hyperplane with a greater margin, resulting in higher classification accuracy than

the hyperplane in (a). The data points with the shortest distance to the hyperplane on both sides are the real support vectors.

Hyperplanes, in general, are a good solution for linearly separable data. This is an unusual occurrence in non-artificial datasets. To improve classification accuracy, the data is transformed into a higher dimension using a kernel function, which allows it to be linearly separated later.



3.16 Data Pre-processing and Feature Engineering

Pre-processing plays a necessary role in disambiguating the short-texts, not solely in applications that classify short-texts but conjointly for cluster and anomaly detection. (Razzak, 2020)

Pre-processing will have a considerable impact on the overall system performance. However, it is less explored in most literature compared to feature extraction and classification. For simplicity, we will keep the pre-processing minimal for optimal hate speech detection. For pre-processing, we tend to apply case-folding, tokenization, and punctuation removal. For features, we tend to merely extract word TF-ID. (Naseem, 2020)

3.17 Modelling

The knowledge set we are going to use for the hate speech detection model consists of the train set. The coaching package includes a list of 31,962 tweets, a corresponding ID and a tag zero or one for every tweet. The particular sentiment we'd like to notice during this dataset is whether or not or not the tweet relies on hate speech.

So, let’s start with the task of building a hate speech detection model. We will begin with reading the datasets by exploiting the panda’s package in python:

Data cleaning is the method of prepping incorrectly formatted knowledge for analysis by deleting or modifying the incorrectly formatted knowledge that is mostly not necessary or helpful for knowledge analysis because it will hinder the method or offer inaccurate results. We performed the method of information clean up by using the re library in Python:

To establish or classify user-generated content, text options indicating hate should be extracted. Obvious options are individual words or phrases (n-grams, i.e., sequence of *n* consecutive words). To boost the matching of features, words is stemmed to get only the foundation removing morphological differences. process likewise will extract features.

TF-IDF is AN abbreviation for Term Frequency Inverse Document Frequency. this can be terribly common algorithmic program to remodel text into a meaning illustration of numbers which is employed to suit machine algorithmic program for prediction. (ONE, 2019)

The term frequency — inverse document frequency (tf-idf) weight

We saw above how to calculate term frequency. Now let us come to the idf (inverse document frequency) topic how it is calculated and the multiplication with tf (term frequency). The idf is defined as*:*

$$\operatorname{idf}(t) = \log \frac{|D|}{1 + |\{d : t \in d\}|}$$

where $|\{d : t \in d\}|$ is the **number of documents** where the term t appears, when the term-frequency function satisfies $\operatorname{tf}(t, d) \neq 0$, we're only adding 1 into the formula to avoid zero-division.

$$\text{tf-idf}(t) = \operatorname{tf}(t, d) \times \operatorname{idf}(t)$$

This formula has an important consequence because if a high tf-idf calculation weight is achieved we will have a high term frequency (tf) in the given document (local parameter) and a low document frequency of the term in the entire collection (parameter global). We have calculated the above test data matrix, we need to calculate idf (t). (Chaudhary, 2020)

3.18 Evaluation

Split validation

Split validation is a means of predicting how the model fits a hypothetical test when there is no explicit test set. The Split Validation operator also offers training on an explicitly tested data set.

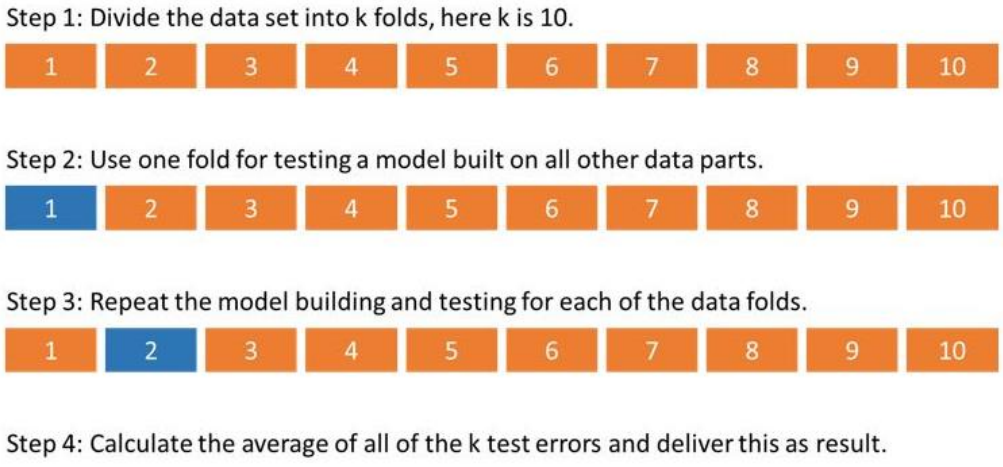
The operator for Split Validation is a nestled operator. It has two subprocesses:

- A subprocess for training
- A subprocess for testing.

The education subprocess is used to study or create a model. The trained model is then used in the subprocess of the test. During the test stage, the performance of the model is also measured.

Cross Validation

Cross Validation is a statistical method; the technique assess the potential of machine learning models. Cross validation is used for comparisons in machine learning because of its simplicity, ease of use, and its results are less than other methods. Cross validation divides the data in number of small groups called folds, the number of groups is called a parameter k. For example, if you divide the data into 10 equal sections the value of k will be 10 and is called 10-fold cross validation.



Mean Squared Error (MSE)

A regression line's mean squared error (MSE) indicates how close it is to a set of points. It accomplishes this by squaring the distances between the points and the regression line (these distances are the "errors").

$$MSE=(1/n)*\Sigma(y_i - f(x_i))^2$$

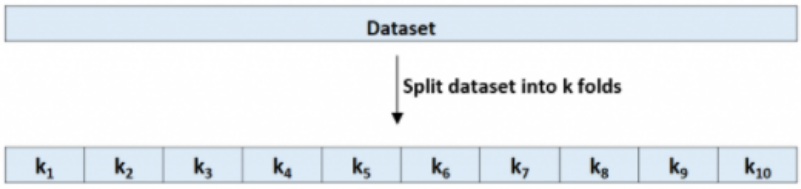
n is total number of observations

y_i is the return value of i^{th} observation

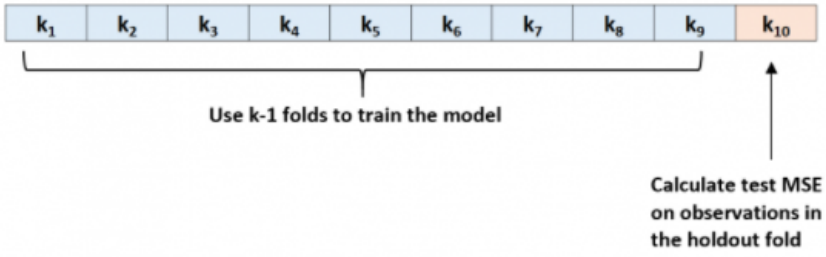
$f(x_i)$ is the value of predicated return

The MSE will be small if the predication and observation close to each other.

Step 1: Randomly divide a dataset into *k* groups, or "folds", of roughly equal size.



Step 2: Choose one of the folds to be the holdout set. Fit the model on the remaining k-1 folds. Calculate the test MSE on the observations in the fold that was held out.



Step 3: Repeat this process *k* times, using a different set each time as the holdout set.

Step 4: Calculate the overall test MSE to be the average of the k test MSE's.

$$\text{Test MSE} = (1/k)*\Sigma MSE_i$$

(MSE_i is the test on last iteration)

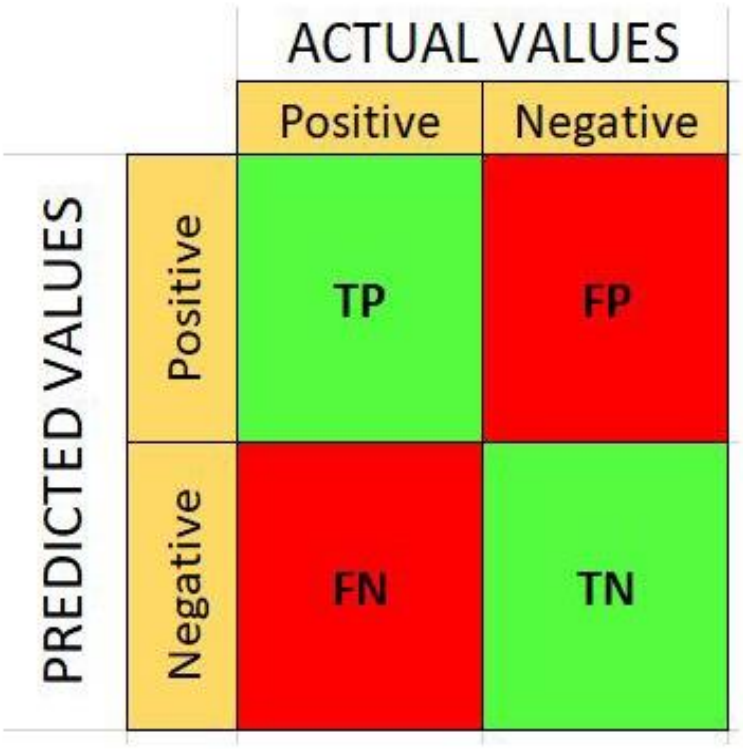
Model assessment is a key component of an effective machine education model. There are different assessment metrics, which are used for different kind of problems. We used confusion metrics in our project.

Confusion Metrics

Confusion Metrics are a synopsis of true and false predication made by a classifier in a tabular form. It is used for measuring the efficiency of a classification model through calculation of performance metrics like accuracy, precision, recall, and F1- score. Confusion Metrics are used because they give a better understanding of model's performance than classification accuracy.

Model efficiency can be measured by two methods.

1. Confusion matrix
2. Classification Measure



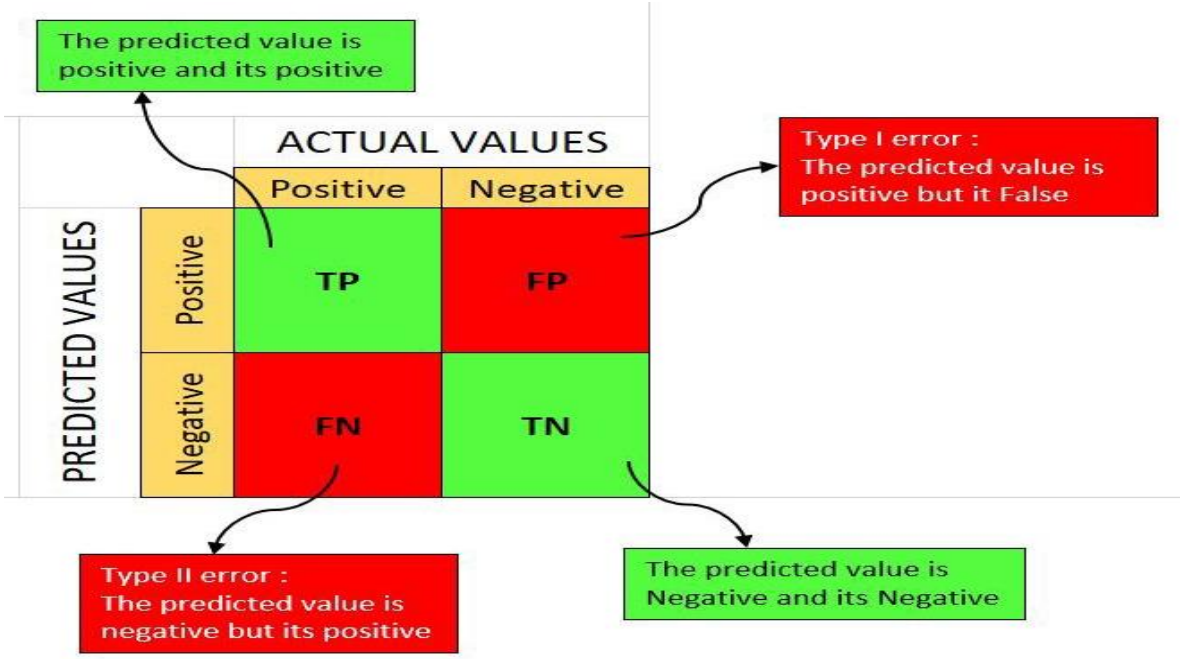
1. Confusion Matrix

A confusion matrix is a method of summarizing a classification algorithm's results. As we discussed previously if the number of observations in each class is unequal, classification accuracy alone can be misleading. Calculating a confusion matrix can help you understand what your classification model is getting right and where it is going wrong.

The below Four terminologies will help us to understand confusion Matrix.

- **True Positives (TP):** when both values (Actual & Predicted) are positive
- **True Negatives (TN):** when both values (Actual& Predicted) are negative
- **False Positives (FP):** When actual value is negative and predication is positive, also called **Type 1 error**.

- **False Negatives:** When actual is positive, but predication is Negative, also called **Type 2 error.**



2. Classification Measure

Other than confusion matrix, few measurement techniques can help us with understanding and the analysis of data. Basically, it is an extended version of confusion matrix.

a. Accuracy:

   Accuracy measures how many times a classifier makes the correct predication. It is in proportion between the correct predications and the total number of predications. Accuracy metrics are not suitable for unbalanced classes.

b. Precision

   The proportion of the total number of correctly classified positive classes divided by the total number of predicted positive classes.

   It is a useful metric where False Positive is higher concern than False Negative.

c. Recall

   The proportion of the total number of correctly classified positive classes dived by the total number of positive classes. Alternatively, in simple words in all positive classes and how many we predicted correctly.

   It is a useful metrics in cases where False Negative is higher than false positive.

Chapter 4. Implementation

4.1 Overview

Overview of the implemented steps. The title and order of the next sections should be customised, depending on what you did.

In this section we will discuss the implementation phase of the project using the NLP Model. First, we began by reading the dataset to then create a data frame. The dataset will have two columns' separating the tweets and their category. We then will set out to visualize the data using word clouds, pie charts, data distribution and text information.

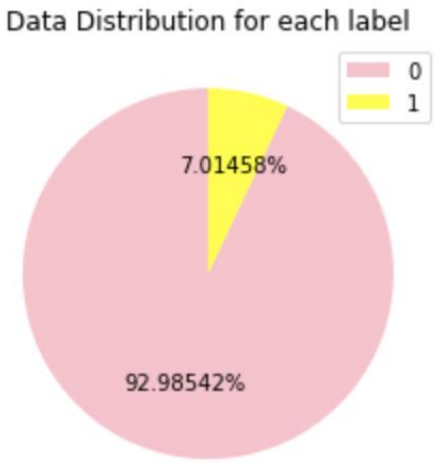
We will then evaluate our findings from the visualisation phase to aid with our text pre-processing and removing certain stop words, replacing emoji with text and other special characters that are not pertinent to the visualisation of the data and for the NLP. Another pre-processing method we will enact is stemming which is a process of producing morphological variants of a root/base word. But being aware of the possible errors i.e., Over stemming and Understemming.

After the Data pre-processing, we move on to the training of the model using cross validation. We will convert the data sets text into numerical data using tf-idf vectorizer to then train our machine learning models with the vectorised data. To evaluate the classifier build we then used metric accuracy which was then tested on different folds using k-fold cross validation. Then using this data, we will plot the confusion matrix along with an accuracy plot for evaluation. Lastly when our model has been finely tuned we will save it in the disk using the pickle function meaning in the future we can make predictions using the saved models.

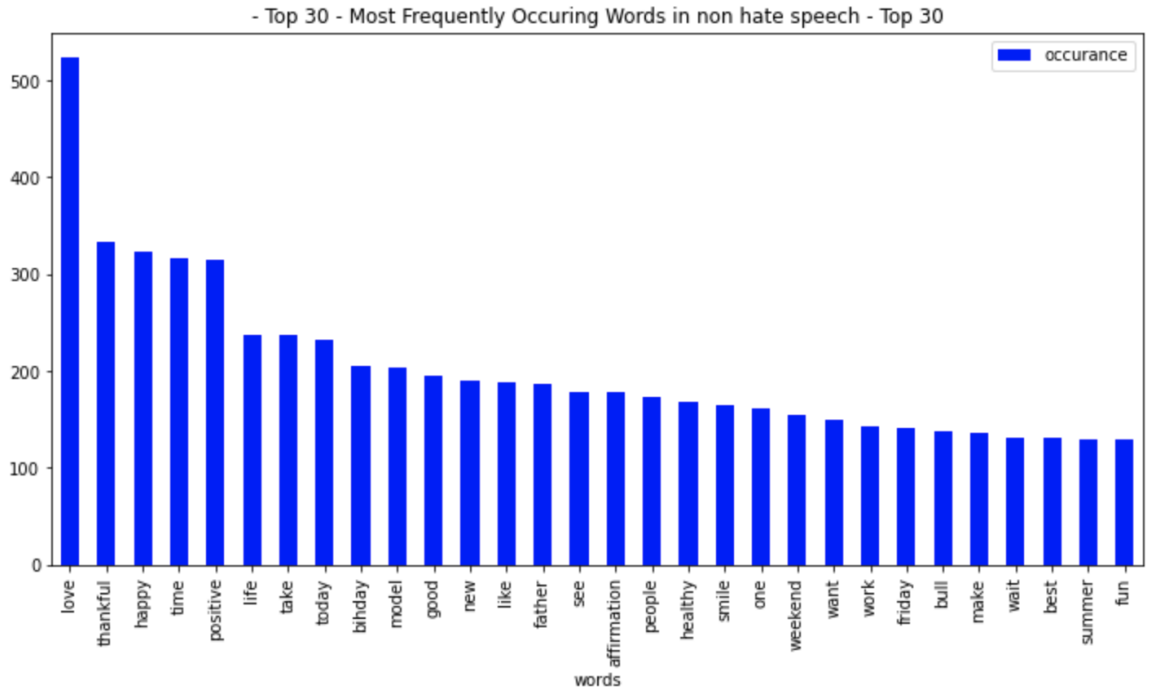
4.2 Data Understanding

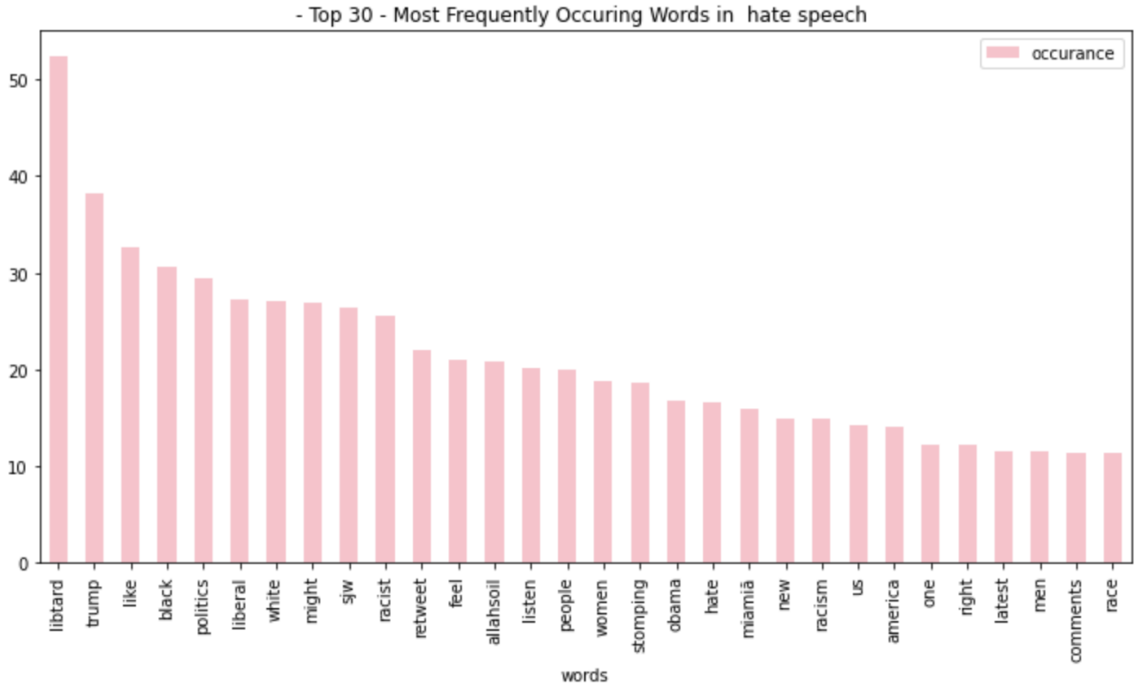
For our data understanding phase we first began by visualising the raw the data itself to show the data distribution for each label using matplotlib to create a pie chart i.e. hate speech and non-hate speech represented as 0 being non hate speech and 1 being hate speech as we previously explained in the methodology.

```
### Pie chart showing the class distribution in data

import matplotlib.pyplot as plt

class_names = (df['Target'].unique()) ## This line will store the name of the labels, since in our case
## labels are stored in the form of numbers i.e 0 for non hate , 1 for hate speech
## So this variable will store only 0 and 1


## This variable will store the occurance of 0 and 1 in our data set
values_of_classes = list(df['Target'].value_counts())


colors = ['pink', 'yellow', 'brown', 'black']
patches, texts,per = plt.pie(values_of_classes,autopct='%0.5f%%', colors=colors, startangle=90)
plt.legend(patches, class_names, loc="best")


plt.title('Data Distribution for each label')
plt.show()
```



We then set out to vectorise the data using TfidfVectorizer which will get the vector of each word of text in the dataset. From here we then summed up all the vectors across the rows using the .sum function then using the sum of words we used Scikit-Learn .vocabulary_.items() function to get the frequency of each word. Now that we have the frequency of the words, we can visualise the information using a Data Frame of the Top 30 Most Frequent Occurring Words in both Hate Speech and Non Hate Speech respectively.


- Top 30 - Most Frequently Occuring Words in non hate speech - Top 30
occurance
500
400
300
200
100
0
love
thankful
happy
time
positive
life
take
today
bihday
model
good
new
like
father
see
affirmation
people
healthy
smile
one
weekend
want
work
friday
bull
make
wait
best
summer
fun
words


- Top 30 - Most Frequently Occuring Words in hate speech
occurance
50
40
30
20
10
0
libtard
trump
like
black
politics
liberal
white
might
sjw
racist
retweet
feel
allahsoil
listen
people
women
stomping
obama
hate
miamiä
new
racism
us
america
one
right
latest
men
comments
race
words

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(stop_words = ["via", "re", "go", "get", "day", "i", "i_am","urã", "

from wordcloud import WordCloud
no_hate['text'] = no_hate['text'].str.replace('@user',' ')
no_hate['text'] = no_hate['text'].str.replace('amp',' ')


## Getting vectors for each word of text
words = vectorizer.fit_transform(no_hate.text)
## Summing the vectors across the rows
sum_words = words.sum(axis=0)

## Getting the frequency of each word
words_frequency = [(word, sum_words[0, i]) for word, i in vectorizer.vocabulary_.items()]
## Sorting the words with in ascending order
words_frequency = sorted(words_frequency, key = lambda x: x[1], reverse = True)
## Making a dataframe of the occurances
occurance = pd.DataFrame(words_frequency, columns=['words', 'occurance'])

occurance.head(30).plot(x='words', y='occurance', kind='bar', figsize=(12, 6), color = 'blue')
plt.title("- Top 30 - Most Frequently Occuring Words in non hate speech - Top 30")
```

```
hate['text'] = hate['text'].str.replace('@user',' ')
hate['text'] = hate['text'].str.replace('amp',' ')


## Getting vectors for each word of text
words = vectorizer.fit_transform(hate.text)
## Summing the vectors across the rows
sum_words = words.sum(axis=0)

## Getting the frequency of each word
words_frequency = [(word, sum_words[0, i]) for word, i in vectorizer.vocabulary_.items()]
## Sorting the words with in ascending order
words_frequency = sorted(words_frequency, key = lambda x: x[1], reverse = True)
## Making a dataframe of the occurances
occurance = pd.DataFrame(words_frequency, columns=['words', 'occurance'])

occurance.head(30).plot(x='words', y='occurance', kind='bar', figsize=(12, 6), color = 'pink')
plt.title("- Top 30 - Most Frequently Occuring Words in  hate speech ")
```

As seen above in both data charts we can see the juxtaposition between occurring words from hate speech and non-hate speech tweets with the most frequent words of each category rarely overlapping with exception of relevant stop words that are pertinent to the context of hate speech category and non-hate speech category. This will be expanded on in the below section.

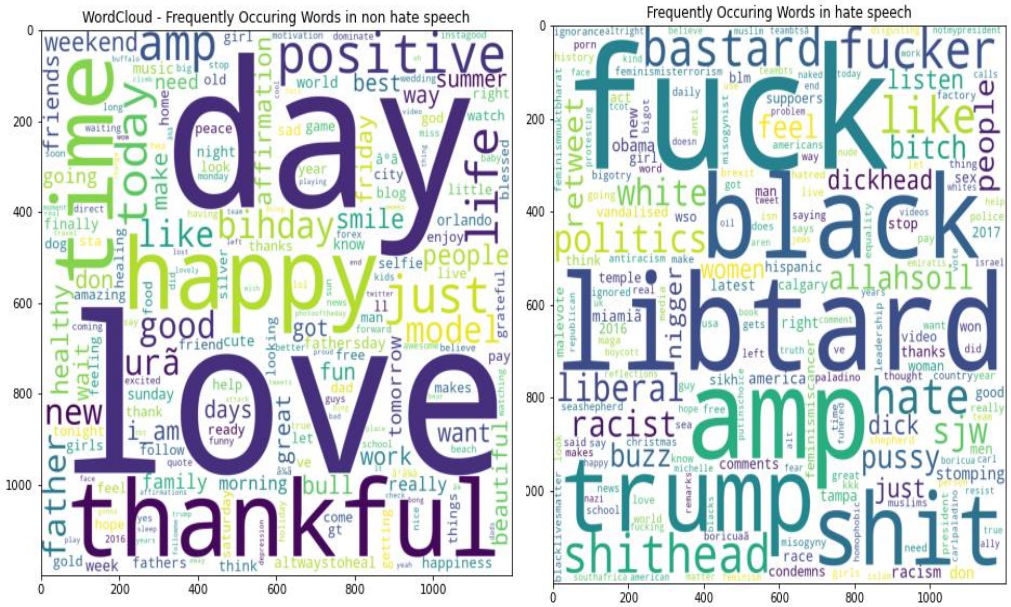
4.3 Vectorisation

4.3.1 TF-IDF Vectorisation

After successfully loading the dataset, we try to extract vectors for each word from text, these are words are related to the non-hate speech context. After gathering relevant data words are summed across the rows and extract the frequency of each word occurrence in the rows. Furthermore, words are sorted ascending order. Also in the data set word user referred as “individual users” are also taken out in terms of proper graph alignments.

4.3.2 Frequency Vectors

Furthermore, we will apply the same rules as previously mentioned to sort the words using the same method to present words frequencies in a word cloud. This project’s model is heavily focusing on the words frequency and its occurrence to find the balance between hate and non-hate speech words as they appear in the dataset. Finally, the words are normalizing by the occurrence of each words based on their weightage and the words populated in the document. The model tried to present the words using frequency vectors in the word cloud.



4.4 Baseline Implementation

4.4.1 NLTK Removing Stop Words

In terms of classification and data folds, our model's baseline implementation focuses on machine learning algorithms. In order to run the model accurately it is very important to covert data in pre-processing phase so the machine can understand data precisely. In the NLTK model, the pre-processed data is filters out useless data, which also referred as stop words.

4.4.2 NLTK Tokenization

In data processing NLTK tokenization is referred as taking large chunks of texts to process and divide into smaller part known as tokens. Tokens are very useful to locate patterns and also it is considered as the early step towards stemming and lemmatization. Tokens are also help to alternate between sensitive and non-sensitive data elements. In our model tokenization method splitting sentences into words to convert data frames for better understanding.

4.4.3 NLTK Lemmatization

In our model NLTK Lemmatization is used to grouping various forms of inflected words so they can be analysed as a single item. Also, Lemmatization has similarities to stemming where it can bring word contexts. Interestingly Lemmatization not analysing words in morphological order.

```
Examples of lemmatization:

-> rocks : rock
-> corpora : corpus
-> better : good
```

4.4.4 NLTK Stemming

We used NLTK stemming model to produce morphological variations of root and base words as stemmers for example importing modules PorterStemmer from NLTK and load words from the file for implementation mentioned in output mentioned below:

```
program  :  program
programs  :  program
programer  :  program
programing  :  program
programers  :  program
```

4.4.5 Training and Comparing the Models using K-Fold

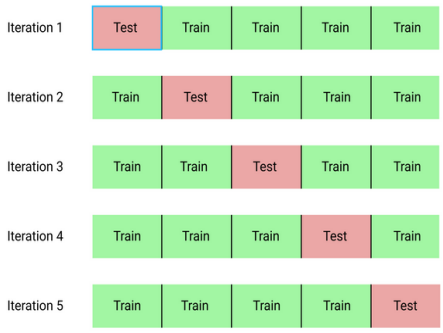
In our model we used the holdout validation technique where we split our datasets into two separate portions, “training Sets” and “Test Sets” where model was trained on training set to predict labels on test sets and computing metrics for accuracy.

To execute the K-Fold model first we split our data sets in to equal partitions where data iterated ten times for accuracy and results.

Understanding K-Fold:

We split data into “k” referred as equal partition of data, and selected k-1 as the main partition to training set of data. We then selected the remaining partitions as

the “test set” and train our model on training set to predict labels on “test set” mentioned in example below:

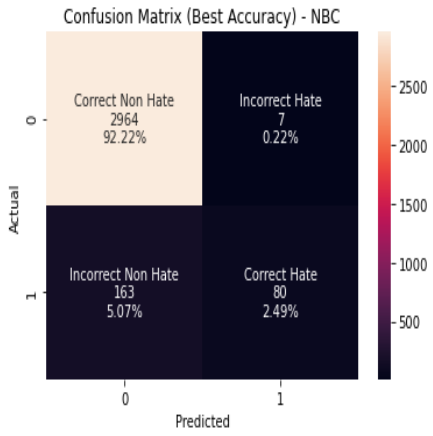


We use 10 folds for cross validation and for accuracy, here are the results of K-Fold after iterations using MultinomialNB model for scores across each fold

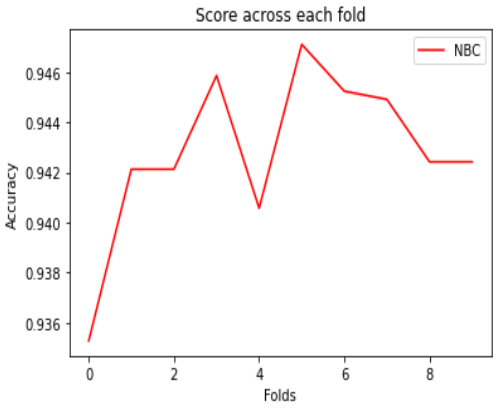
*4.4.6 MultimomialNB **Matrix**:*

In the this step we use MultinimailNB model to test data based on data occurrence count the only difference and to use this model for the best results is that Multinomial purely focuses on count occurrence

```
from sklearn.naive_bayes import MultinomialNB

data = df.normalized
targets = df.Target

# min_df=30 is a clever way of feature engineering
tfidf = TfidfVectorizer(sublinear_tf=True, min_df=30, norm='l2', ngram_range=(1,3))

# NBC Model
NBC_clf = MultinomialNB()

# NBC cross-validation
NBC_average_acc, training_data_with_max_acc, test_data_with_max_acc, targets_with_max_acc_train, targets_with_max_acc_test

# NBC confusion matrix
evaluate(training_data_with_max_acc, test_data_with_max_acc, targets_with_max_acc_train, targets_with_max_acc_test, tfidf,

plt.plot(nbc_accuracies,color='r',label="NBC")
plt.legend(loc="best")

## Setting x-label of fig
plt.xlabel("Folds")

## Setting y-label of fig
plt.ylabel("Accuracy");

## Setting subtitle of fig
plt.title("Score across each fold")
```




```
Out[12]: Text(0.5, 1.0, 'Score across each fold')
```

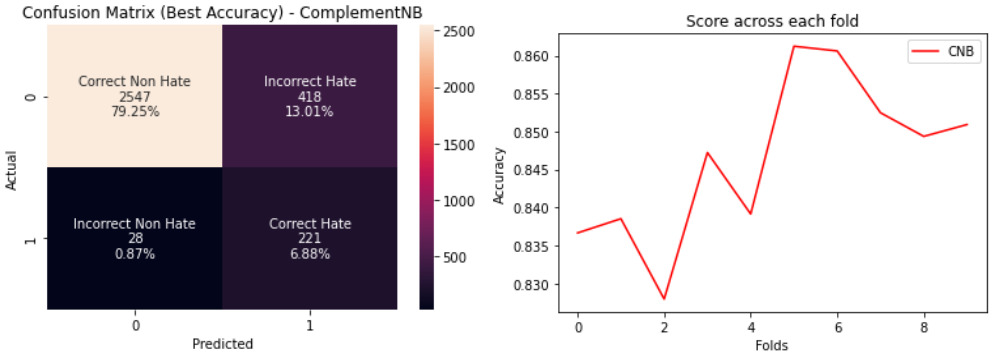



```
Average k-fold  accuracy for NBC:  0.9428073974525164
Best k-fold  accuracy for NBC:  0.9471064094586186
```

4.4.7 "ComplimentNB" Matrix

Compliment NB model is the good model to correct the serious assumptions and best worked with imbalanced datasets. We use this model to calculate compliment from each class to sum out the weight of the model for the text classification purpose specially for large documents

```
#ref: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.ComplementNB.html#sklearn.naive_bayes.Compleme


import numpy as np
from sklearn.naive_bayes import ComplementNB


data = df.normalized
targets = df.Target

# min_df=30 is a clever way of feature engineering
tfidf = TfidfVectorizer(sublinear_tf=True, min_df=30, norm='l2', ngram_range=(1,3))

# CNB Model
B_clf = ComplementNB()
B_clf.fit(xlabel, ylabel, sample_weight=None)
B_clf.fit(xlabel, ylabel)

# NBC cross-validation
CNB_average_acc, training_data_with_max_acc, test_data_with_max_acc, targets_with_max_acc_train, targets_with_max_acc_test

# NBC confusion matrix
evaluate(training_data_with_max_acc, test_data_with_max_acc, targets_with_max_acc_train, targets_with_max_acc_test, tfidf,


plt.plot(bnc_accuracies,color='r',label="CNB")
plt.legend(loc="best")

plt.title("Score across each fold") ## Setting subtitle of fig 1
plt.xlabel("Folds")
plt.ylabel("Accuracy"); ## Setting y-label of fig 1
```

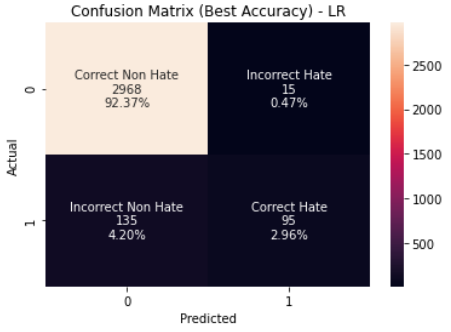


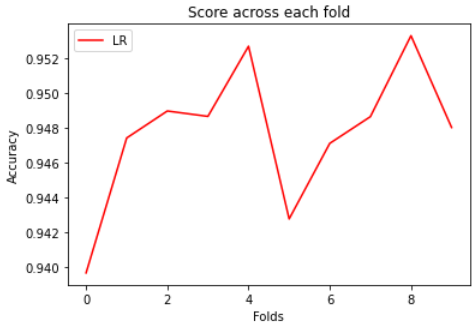

```
List of k-fold  accuracies for ComplementNB:  [0.8366521468574984, 0.8385189794648413, 0.8279402613565651, 0.84723086496577
47, 0.8391412570006223, 0.8612321095208463, 0.8606098319850654, 0.8524743230625583, 0.8493619670090259, 0.8509181450357921]
Average k-fold  accuracy for ComplementNB:  0.846407988625859
Best k-fold  accuracy for ComplementNB:  0.8612321095208463
```

4.4.8 “LR” (Logistic Regression) Matrix

In this fold model uses “LR” regression model for the best accuracy and results with transforming its output into probability and using TF-IDF offers high accuracy results as mentioned below:

```
#ref: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix ##

#xlabel, ylabel = load_iris(return_xlabel_ylabel=True)
data = df.normalized
targets = df.Target

#LR_clf = LogisticRegression() # Logistic Regression Model
LR_clf = LogisticRegression(random_state=0).fit(xlabel, ylabel)
LR_clf.predict(xlabel[:2, :])
LR_clf.predict_proba(xlabel[:2, :])



LR_average_acc, training_data_with_max_acc, test_data_with_max_acc, targets_with_max_acc_train, targets_with_max_acc_test,

# LR confusion matrix
evaluate(training_data_with_max_acc, test_data_with_max_acc, targets_with_max_acc_train, targets_with_max_acc_test, tfidf,

## Comparision through graphs

plt.plot(lr_accuracies,color='r',label="LR")
plt.legend(loc="best")

plt.title("Score across each fold") ## Setting subtitle of fig 1
plt.xlabel("Folds")
plt.ylabel("Accuracy"); ## Setting y-label of fig 1


print("Coeficient results")
print(LR_clf.coef_)

print("Intercepts results")
print(LR_clf.intercept_)
```



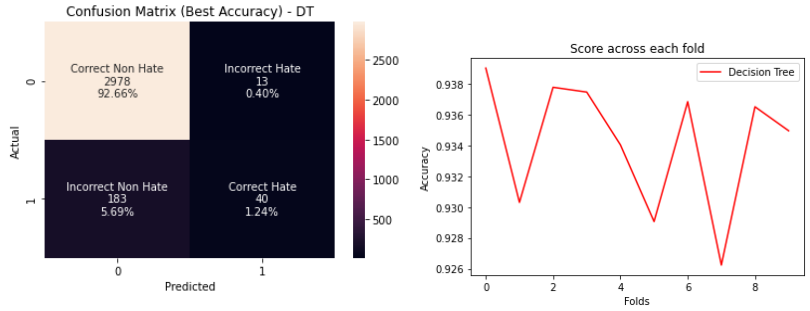



```
List of k-fold  accuracies for LR:  [0.939639079029247, 0.947417548226509, 0.9489732420659615, 0.948662103298071, 0.9527069
072806472, 0.9427504667081519, 0.9471064094586186, 0.9486461251167133, 0.9533146591970122, 0.9480236539060068]
Average k-fold  accuracy for LR:  0.9477240194286939
Best k-fold  accuracy for LR:  0.9533146591970122
```

4.4.9 “DT” Decision Tree Matrix

In addition, our model uses Decision tree algorithm to predict correct accuracy between k-folds

```
#Ref: https://scikit-learn.org/stable/modules/tree.html
#ref: https://scikit-learn.org/stable/auto_examples/datasets/plot_iris_dataset.html

from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import export_text
from sklearn.datasets import load_iris
#from sklearn import tree

xlabel = [[0, 0], [1, 1]]
ylabel = [0, 1]
DT_iris = load_iris()
xlabel, ylabel = DT_iris.data, DT_iris.target

DT_clf = DecisionTreeClassifier(random_state=0, max_depth=4)
DT = DT_clf.fit(xlabel, ylabel) #add 1
DT_clf = DT_clf.fit(DT_iris.data,DT_iris.target)
DT_average_acc, training_data_with_max_acc, test_data_with_max_acc, targets_with_max_acc_train, targets_with_max_acc_test,
evaluate(training_data_with_max_acc, test_data_with_max_acc, targets_with_max_acc_train, targets_with_max_acc_test, tfidf,

## Comparision through graphs



plt.plot(dt_accuracies,color='r',label="Decision Tree")
plt.legend(loc="best")

plt.title("Score across each fold") ## Setting subtitle of fig 1
plt.xlabel("Folds")
plt.ylabel("Accuracy"); ## Setting y-label of fig 1

#tree.plot_tree(DT_clf)
```




```
List of k-fold  accuracies for DT:  [0.9390168014934661, 0.9303049159925326, 0.9377722464219042, 0.9374611076540137, 0.9340
385812072184, 0.9290603609209708, 0.9368388301182328, 0.9262371615312792, 0.9365079365079365, 0.9349517584811703]
Average k-fold  accuracy for DT:  0.9342189700328726
Best k-fold  accuracy for DT:  0.9390168014934661
```

4.5 Data Pre-processing

During data pre-processing, the raw data is prepared and made suitable for a machine-learning model. This is the critical first step in building a machine learning model. When creating a machine learning project, we don't always find

cleanse and formatted data. Whenever you perform any operation on data, you need to clean and format it. (shahbazchandio, 2020)

Data pre-processing is a process in which raw data is prepared and made suitable for a machine-learning model. This is the first and critical step in building a machine learning model. Why do we need data pre-processing? Real world data usually contains noise, missing values, and possibly an unusable format that cannot be directly used in machine learning models. Data pre-processing is a necessary task to clean up the data and make it suitable for a machine-learning model, which also increases the accuracy and efficiency of a machine-learning model. (JavaTPoint, 2018)

Functional Scale Retrieving the Dataset In order to build a machine learning model, you must first do a dataset, because a machine learning model works entirely with data. The data collected in a suitable format for a specific problem is called a data set. The dataset can have different formats for different purposes. For example, if you want to build a machine learning model for business purposes, the dataset will be different from the dataset. required for a patient with liver. Therefore, each data set is different from another data set. To use the dataset in our code, we usually include it in a CSV file. CSV stands for "Comma Separated Values" files. is a file format that allows us to save tabular data such as spreadsheets. It is useful for large amounts of data, and you can use those amounts of data in programs. (Anon., 2020)

During data pre-processing, raw data are prepared and made suitable for a machine learning model. This is the first and critical step in building a machine learning model. When creating a machine learning project, we don't always come across clean and formatted data. Whenever you perform any operation on data, you need to clean and format it. We therefore use the data pre-processing task for this. Why do we need data pre-processing? Real world data usually contains noise, missing values, and possibly an unusable format that cannot be directly used in machine learning models. This also increases the accuracy and

efficiency of a machine learning model. It consists of the following steps: Get the record for importing libraries. Import records. (khan, 2020)

4.6 Modelling

Getting the Dataset To build a machine learning model, we first need a dataset because a machine learning model works entirely with the data. Data that is captured in a format appropriate for a specific problem is called a data set. For example, if we wanted to build a machine-learning model for commercial purposes, the dataset would be different from the dataset required for a liver patient, i.e. every dataset; it would be different from any other data set. To use the dataset in our code, we usually include it in a CSV file. What is a CSV file? CSV stands for "Comma Separated Values" files. It is a file format that allows us to save table data as a spreadsheet. It is useful for large amounts of data, and you can use those amounts of data in programs. In the event of problems in practice, we can download data sets online from various sources, e.g. //www.kaggle.com/. We can also create our dataset by collecting data using various APIs with Python and pasting it into a CSV file. 2) Importing Libraries In order to carry out the data preprocessing with Python, some predefined Python files have to be imported. (McCullum, 2020)

Libraries These libraries are used to perform specific tasks. There are three specific libraries that we use for data preprocessing: Numpy: The Numpy Python library is used to include any type of math operation in the code. It is the basic package for scientific computing in Python. It also supports adding arrays and large multidimensional arrays. So in Python we can import it as: Import Numpy, which Matplotlib is a short name for numpy and is used throughout the program: the second library is matplotlib, a Python 2D graphics library, and with this library we need to import a Pyplot sub-library. This library is used to draw all kinds of diagrams in Python for your code. It is imported as follows: importmatplotlib.pyplot as plt. Here we have used the plt as a short name for this library.Pandas: The final library is the Pandas library, one of the most popular Python libraries used for importing and managing records. It is an open source library for manipulating and analyzing data. It is imported as follows: Here we have used pd as the short name of this library. (Hewitt, 2005)

Importing the Datasets Now we need to import the datasets that we have collected for our machine learning project. Before importing a dataset, however, we need to set the current directory as the working directory. The current folder is now defined as the work directory.read_csv () function: To import the data record, we now use the read_csv () function from the Pandas library. With this feature we can read a CSV file both locally and via a URL. Use the read_csv function like this: data_= pd.read_csv (train.csv) Here is data_set: A variable name for storing our data set.
We pass the name of our data set inside the function. Once we have executed the above line of code, the dataset will be successfully imported into our modeling code. The intuition of the tf-idf method is to give heavy weight to any term that occurs frequently in a particular document, but not in many corpus documents.The word appears frequently in a particular document, but not in many documents. It is probably very important to the content of this document. Sickit-Learn implements the tf -idf method in two classes: TfidfTransformer, which uses the sparse matrix output generated by CountVectorizer and transformsit, and TfidfVectorizer, which uses text data and both the extraction and transformation of word bag features tf -idf.The predictor and testing against the same data is a methodological flaw: a model that simply repeats the label. (Science, 2020)

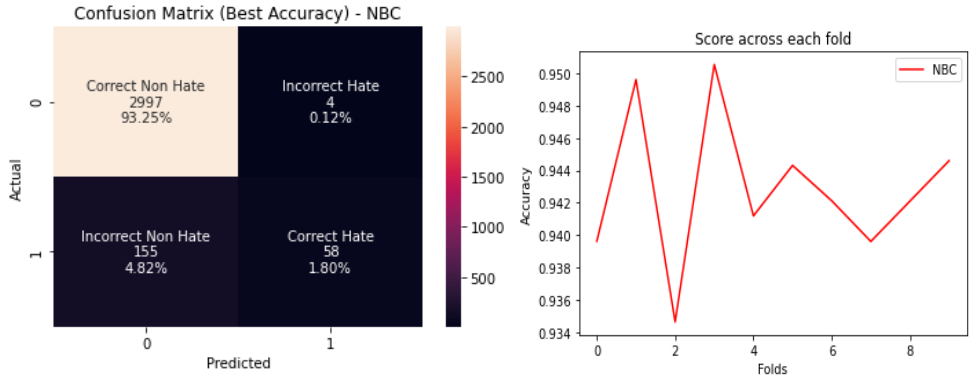
Chapter 5. Results and Analysis

In this chapter, we will discuss the results of our classifiers. We have a highly imbalanced dataset obtained from Kaggle.
Guidelines for policy makers stated (Duarte et al., 2017), “It is tough to recreate a data set for hate speech, since both the distribution and the target of hate series through people’s perception, time and place.”
The results of the models are:

Multinomial Algorithm
Naïve Bayes Classifier
Logistic Regression Model
Decision Tree Algorithm

5.1.1 Multinomial Algorithm

In this model we used sklearn naïve bayes model using multinomial algorithm to take advantages of word counts, and text classification in model for the best result accuracies and to normalize and set target data using TF-IDF vectorizer. The main model works on “MultimoialNB”, the model work on training data training and testing data with max accuracies. Using k-fold cross validation data is comparing between (data, targets, TF-IDF, and Multinomial algorithm), furthermore data is evaluated with maximum accuracies, on target, train and test train data in confusion matrix where predictions and accuracies are outlined below:



List of K-Fold Accuracies (each Iteration)

1	2	3	4	5
0.9396391	0.9495955	0.934660859	0.950528936	0.950528936
6	7	8	9	10
0.944306161	0.94212819	0.939620293	0.942110177	0.944600062
Average K-Fold Accuracy		0.942838405		
Average Best Accuracy		0.950528936		

Based on above results comparing each fold average results comes up as 0.94 and average best accuracy result comes up as 0.95, considered as Naïve Bayes Model using “Multinomial” algorithm successfully executed

5.1.2 Naïve Bayes Classifier (ComplementNB)

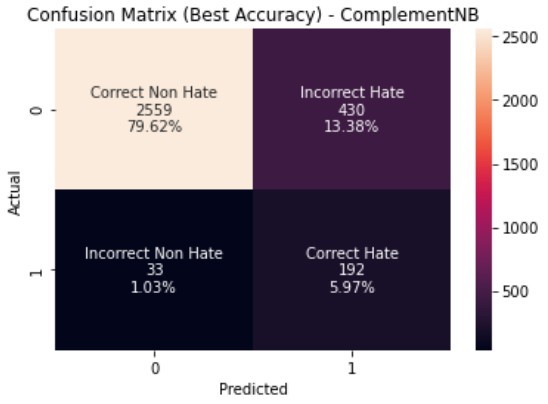
We used Naïve Bayes Classifier model as to make assumption on test and train data on standard naïve base model based on occurrences of words in class, where we take consideration of our imbalance data set. The advantage of using complementNB is for fast and easy implementation for classification as well as real time fast predictions. In this model algorithm check accuracies between test and train data. To extract best result 10 iterations on k-fold method was executed to provide best results using confusing matrix, which gave best accuracies for ComplementNB as: 0.85

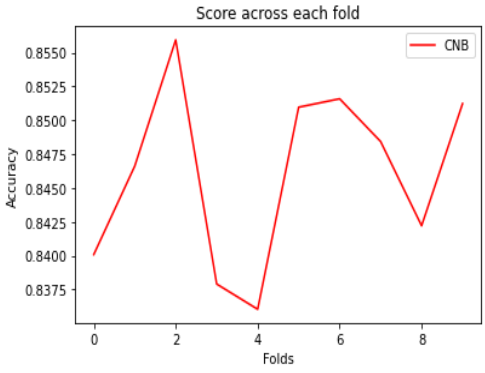
ComplementNB Results using kfold iterations

1	2	3	4	5
0.840074673	0.84660859	0.85594275	0.837896702	0.836029869

6	7	8	9	10
0.836029869	0.85096453	0.851586808	0.84842826	0.842203548

Average K-Fold Accuracy	0.846096511
Average Best Accuracy	0.85594275



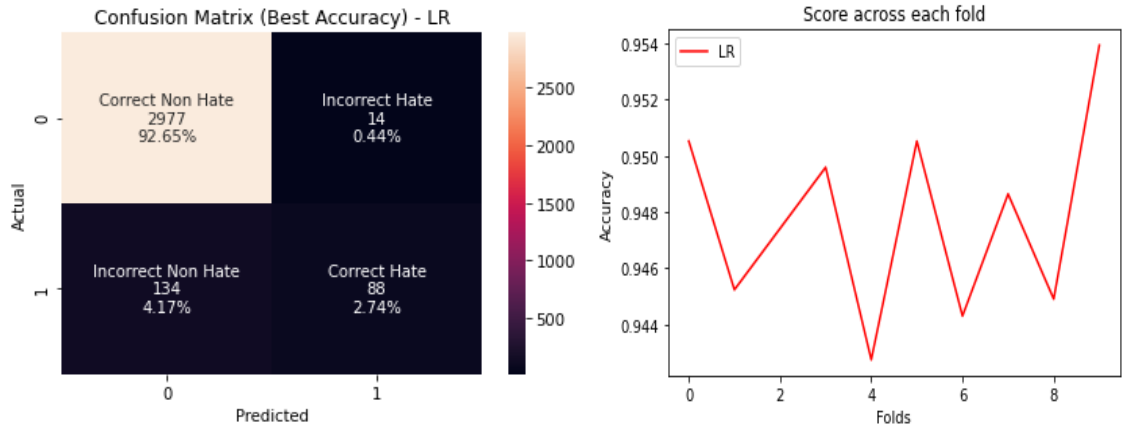


On above screen shots results are presented based on scoring 0.85 on k-fold, and confusion matrix to predict correct total words 2559 with accuracy of 79.62% and correct Hate total predict as 192 with 5.97 percentage, incorrect hate words results was 33 with 1.03 percent, as well as correct incorrect 430 words predict with percentage of 13.38%.

5.1.3 Logistic Regression Model

Logistic regression model is used in third algorithm using k-fold iterations to extract predict result, in this model we aim for average accuracies, and train our data with max accuracies as well same approach is used to test the data accordingly. Furthermore results are cross validated between data and targets to evaluate train and test data with maximum accuracies where the average results came as 0.94% and best accuracy results iterated as 0.95% considered as good results. Also coefficient results using LR model are as follows:

LR Model Results using kfold iterations				
1	2	3	4	5
0.950528936	0.94523958	0.947417548	0.94959552	0.942750467
6	7	8	9	10
0.950528936	0.94430616	0.948646125	0.944911298	0.95393713
Average K-Fold Accuracy		0.94778617		
Average Best Accuracy		0.95393713		



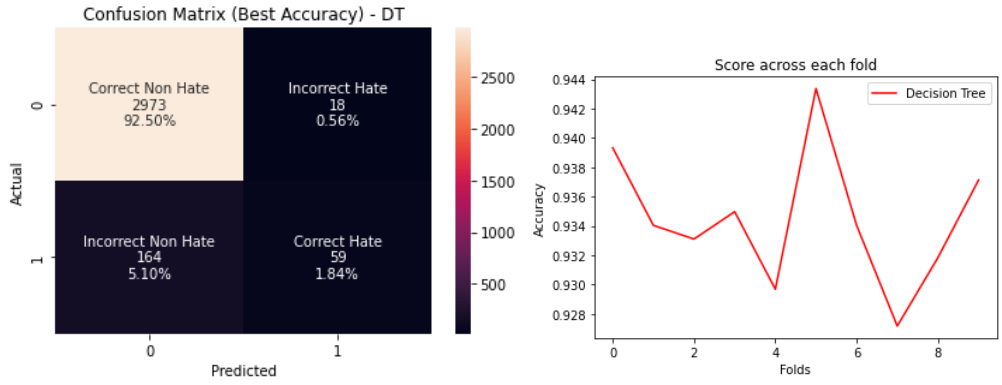
Based on above confusing matrix total ‘Correct Non Hate words’ are accounted for 2977 with 92% result and ‘Correct Hate’ accounts for 88 words with total of 2.74% as well as ‘Incorrect Hate’ 0.44% and correct hate represent 2.74% respectively

5.1.4 Decision Tree Algorithm

We used Decision Tree algorithm, for in depth insights on target and prediction of data for optimal results. In general terms decision tree considered as a binary tree structure which analysing individual nodes to check ‘leaf’ presence. In defined method set depth and ID to zero (0) to check labels beginnings with its root and make sure individual node must checked at least once. Finally the depth node is set to 4 to analyse and predict data accordingly. The whole process executed using k-fold method with set of ten (10) iterations per fold to get the best results and accuracies on test and trained data mentioned below are the results of each fold iteration using confusion matrix.

DT Model Results using kfold iterations

1	2	3	4	5
0.93932794	0.93403858	0.933105165	0.934971998	0.929682638
6	7	8	9	10
0.950528936	0.94337274	0.934038581	0.927170868	0.937130408
Average K-Fold Accuracy		0.934467833		
Average Best Accuracy		0.943372744		



Final results based on k-fold and confusion matrix where Correct non hate prediction comes as 92.50%, correct hate prediction comes as 1.84%, incorrect

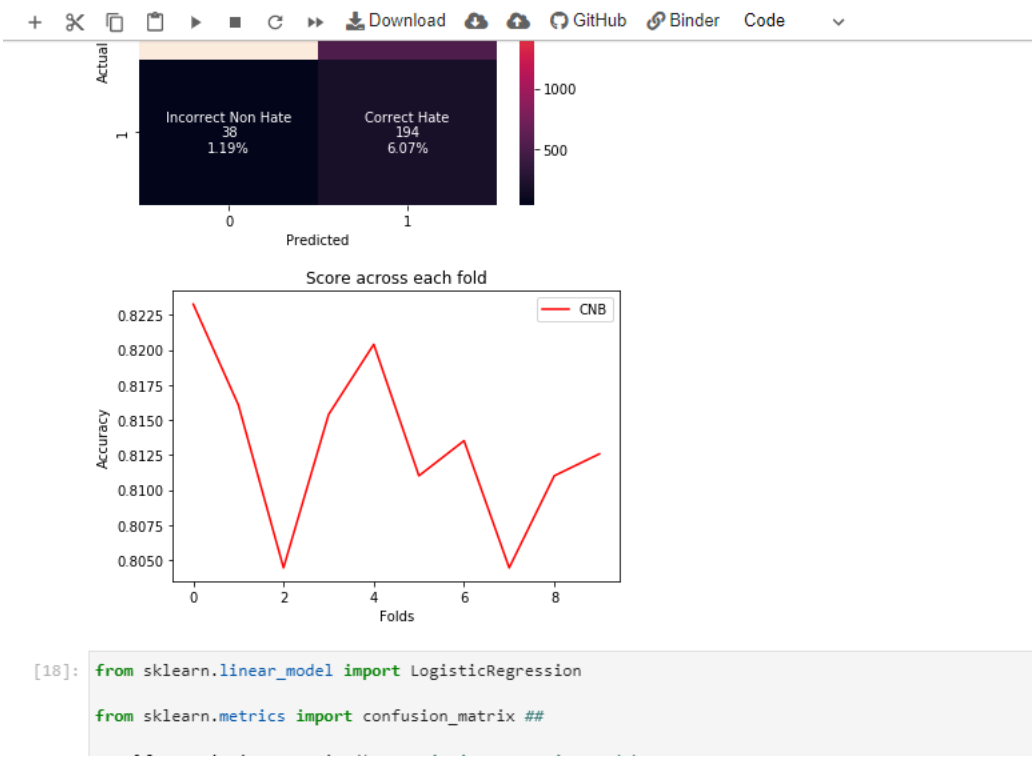
non hate results 5.10% and correct hate showed as 0.56% and results are considered as optimal but not as good as MultinomialNB which gave the best accuracy of 0.95%.

5.2 Optimal Model

The best model for k-fold validation was the LogisticRegressionNB

Logistic regression is a technique of modeling the chance of a discrete final results given an enter variable. The maximum common logistic regression models a binary final results; something which can take values which include true/false, yes/no, and so on. Multinomial logistic regression can version situations wherein there are greater than viable discrete outcomes. Logistic regression is a beneficial evaluation approach for category problems, wherein you are attempting to decide if a brand-new pattern suits quality right into a category. As elements of cyber safety are category problems, which include assault detection, logistic regression is a beneficial analytic technique. (Edgar, 2017)

The best overall model with the correct hate was the Naïve Bayes ComplementNB model which gave us a result of 6.07% correct hate compared to other models such as Logistic Regression, MultinomialNB and DecisionTreeClassifier.




```
[18]: from sklearn.linear_model import LogisticRegression

      from sklearn.metrics import confusion_matrix ##
```

Bayes' Theorem with ComplementNB included is named after Pastor Thomas Bayes. It has a conditional probability. Conditional probability is the probability that certain things will happen, assuming other things have already happened. With the conditional probability we can calculate the probability of an event based on your previous knowledge (formula for calculating the conditional probability). (Double, 2021)

Where P (H) is the probability of assuming that H is true. This is known as the previous probability. P (E) is the probability of the test (it has nothing to do with the hypothesis). P (E | H) is the given test's probability that the hypothesis is true. P (H | E) is the probability of the hypothesis if there is evidence. Let's look at an example to understand how the above Bayes' theorem formula works.Naive Bayesian Algorithm Plugin (CNB) The Naive Bayesian Algorithm is a series of very popular machine learning algorithms for classification. There are many ways to use the Naive Bayesian Algorithm. It is implemented as Gaussian Naive Bayes, Polynomial Naive Bayes, etc. Click this link to learn more about the basics of Naive Bayes. The Naive Bayesian Complement is an adaptation of the standard Naive Bayesian polynomial algorithm. The number of examples in one category is greater than the number of examples in other categories. The samples are not evenly distributed. This type of dataset can be difficult to use because the model can easily replace the dataset to provide more examples for the class. How CNB works: Supplemental Naive Bayes is particularly useful for dealing with unbalanced datasets. In the Naive Bayes plugin, we do not calculate the probability that an element belongs to a certain class, but rather the probability that an element belongs to all classes.This is the literal meaning of the

word "plugin", hence it is referred to as the "Naïve Bayes plugin". Algorithm exercise (not math): For each class, calculate the probability that the specified instance is not one of them. After all the categories have been calculated, we checked all the calculated values and selected the lowest value. The lowest value (lowest probability) was chosen because it is the lowest probability for that particular category. This means that it is likely that this really falls into that category. (Routledge, 2018)

Plugin NB implements the naiveBayes (CNB) plug-in algorithm. An adaptation of the standard algorithm Polynomial Naive Bayes (MNB), which is particularly suitable for asymmetrical data sets. This is statistical information about adding a single record. CNB inventors have shown from experience that the CNB parameter estimate is more stable than the MNB parameter estimate.In addition, CNB generally outperforms MNB on text classification tasks (usually in large situations). The weights are calculated as follows: Bayes' theorem, named after Rev. Thomas Bayes. With a conditional probability. If something else happened, this is the likelihood something will happen. (Milton, 1995)

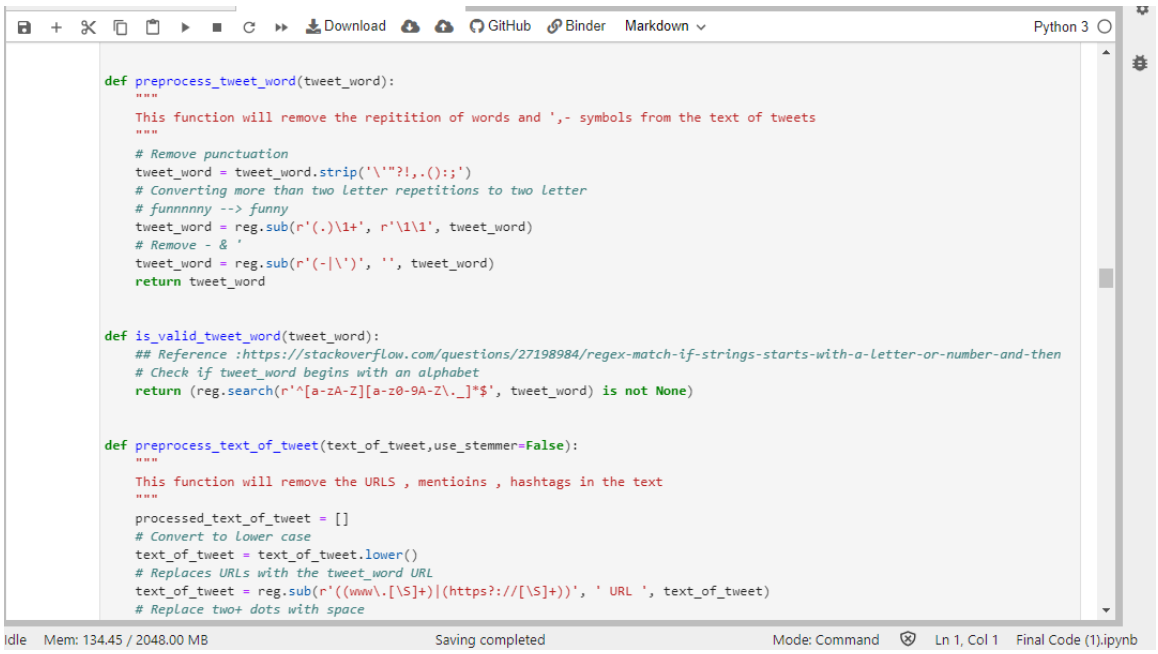
Using conditional probability, we can calculate the probability of an event based on your previous knowledge. The following is the formula for calculating the conditional probability, where P (H) is the probability of the hypothesis. H is this is called the previous probability. P (E) is the probability of the test (it has nothing to do with the hypothesis).P (E | H) is the given test's probability that the hypothesis is true. P (H | E) is the probability of the hypothesis with evidence. Let's look at an example to understand how the above formula applies to Bayes' theorem. Naive Bayesian Plug-In (CNB) The Naive Bayesian Algorithm is a series of very popular and widely used machine learning algorithms for classification. There are many ways to implement the Bayesian naive algorithm, for example: B. Gaussian naive algorithm.Naive Bayes, Standard Algorithm for Fitting Naive Bayesian Polynomials Naive Bayesian polynomials are not suitable for skewed datasets. This class is larger than the number of examples belonging to other classes. This means the distribution. (Barrister, 2021)

Chapter 6. Conclusion

The aim of this project is to understand the meaning behind hate-speech and utilizing python to create a hate speech detection medium to capture hate words in an everyday conversation in the world of social media. From research we have discussed that hate speech is “Hate Speech on Social Media is any kind of direct or indirect communication/post that attacks or uses debasing or discriminatory

language with reference to a person or a group based on who they are, and or on their religion, ethnicity, nationality, race, colour, descent, gender or other identity factor”

We have achieved our aims and decided the logistic regression model is the best suited for k-fold validation with the dataset. The advantage of this is the logistic regression model has a percentage of 0.95% k-fold cross-validation. The weakness is of this is there is a percentage of 6% of correct hate due to the usage of imbalanced data. One way to overcome this is use a balanced dataset. The objective was the re-engineer the dataset and use alternate models to highlight hate speech in a conversation. We have met the objectives by removing repeated words and special characters.We have done so by pre-processing the dataset.

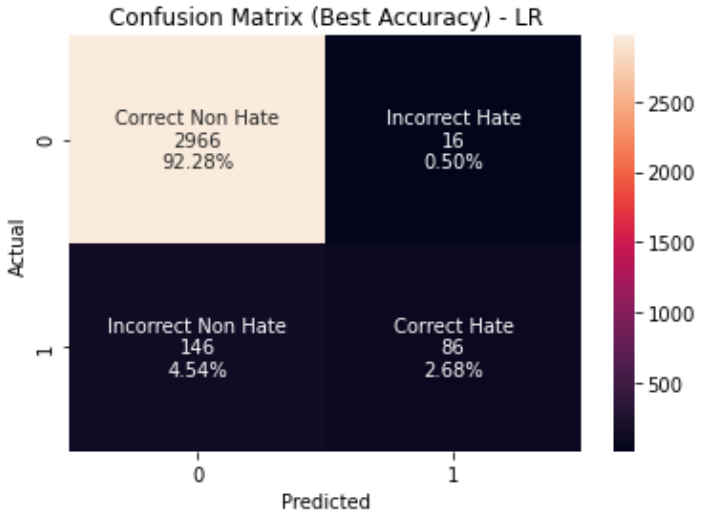


We have used alternate models to run tests on the dataset for detecting hate words.

The models we have used are:

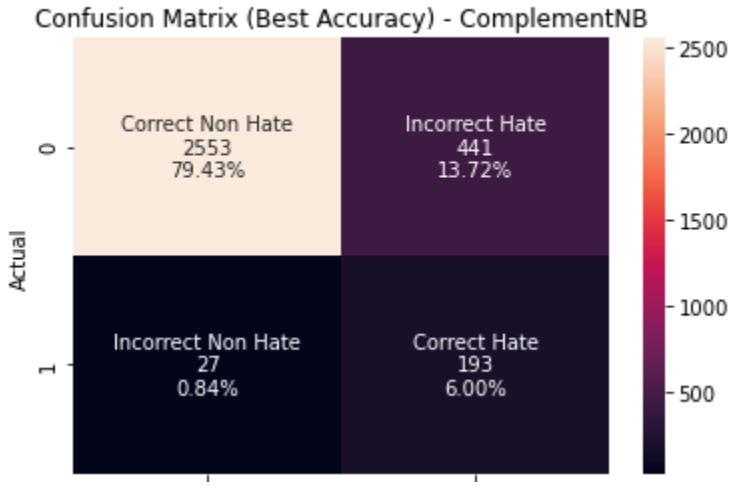
Logistic Regression (highest k-fold validation)

```
Average k-fold  accuracy for LR:  0.9474438299139057
Best k-fold  accuracy for LR:  0.9495955196017424
```

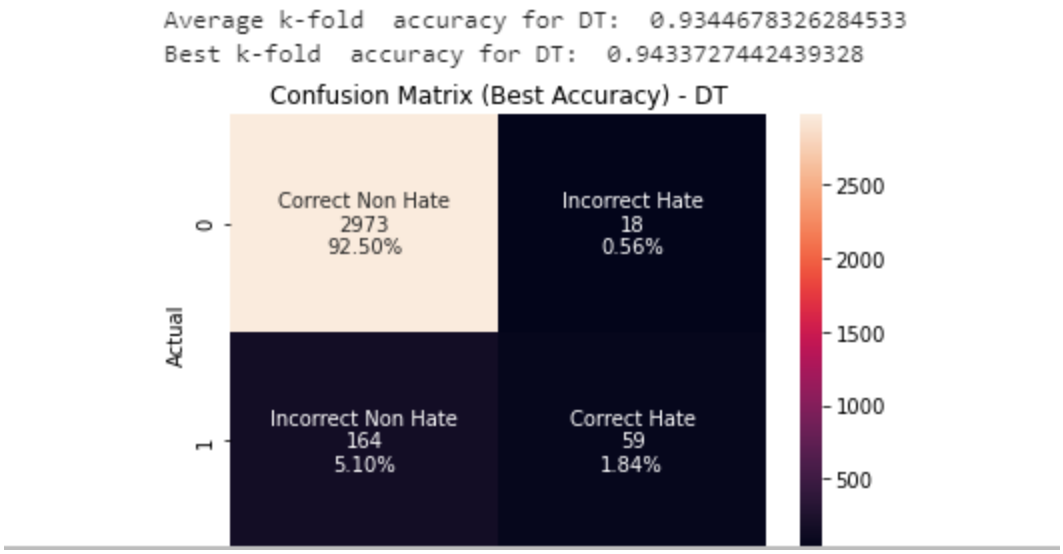


ComplementNB (highest correct hate percentage)

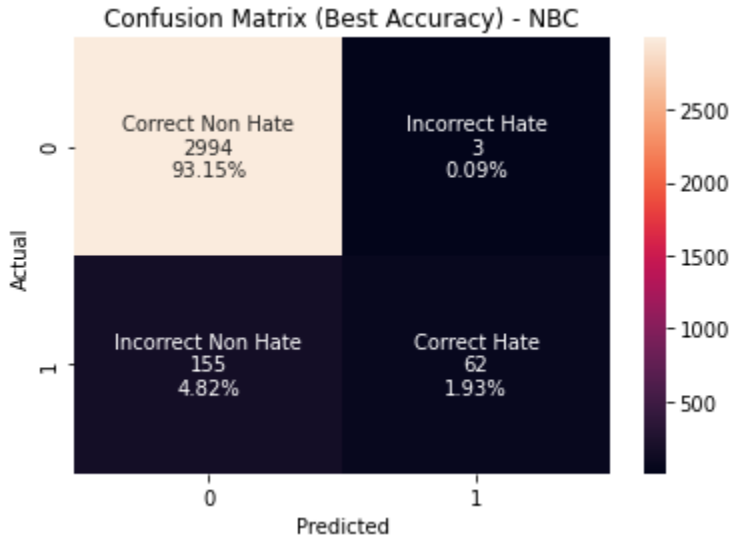
```
Average k-fold  accuracy for ComplementNB:  0.8442916058769494
Best k-fold  accuracy for ComplementNB:  0.8543870566272558
```



Decision Tree



MultinomialNB



Appendix A

"The General Data Protection Regulation (GDPR) mentions several legal grounds for the lawfulness of processing of personal data of data subjects. A lawful basis for processing personal data consists of at least one of those legal grounds and can vary per personal data processing activity and purpose."

“The need for a lawful basis for processing personal data under the GDPR *(with the necessary exceptions)* isn’t new. In its Recitals and Articles the GDPR says pretty much the same as its predecessor, the Data Protection Directive *(Directive 95/46/EC)* did on several fronts. Yet, there are impactful changes too.

The main Recitals and Articles on lawful processing and lawful processing grounds as such, to begin with, are among those where not too much has changed.

Recital 39 of the GDPR recitals has this to say about the lawfulness, fairness, transparency and purpose of personal data processing: any personal data processing must be lawful and fair, it should be transparent to data subjects which personal data regarding them are processed and the principle of transparency requires that ANY information and communication regarding personal data processing is easily accessible and easy to understand. Along with the need to use clear and plain language the latter is explicitly mentioned in the scope of the purposes of the processing.” (i-scoop, 2017)

References

I have included a couple of examples (they should be in alphabetical order and when same authors with different works, organise by publication year):

Davelouis, F., & Gray, G. (2015). Travel Time as a Proximity Measure for Geo-Spatial K-Means Clustering. IT&T, 44.

McHugh,D. and Keyes, L. (2015) Traffic Prediction and Analysis using a Big Data and Visualisation Approach, GIS Research UK (GISRUK), University of Leeds, April 2015

O'Brien M. and M. Hofmann (2017) The Potential of Learning Outcomes analytics, EdTech'17, Irish Learning Technology Association (ILTA), Sligo, Ireland./////

Aditya Mishra (2018). *Metrics to Evaluate your Machine Learning Algorithm*. [online] Medium. Available at: https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234.

Book, F. (2019). *Community Standards | Facebook*. [online] Facebook.com. Available at: https://www.facebook.com/communitystandards/hate_speech.

Derczynski, L. (2021). *leondz/hatespeechdata*. [online] GitHub. Available at: https://github.com/leondz/hatespeechdata [Accessed 3 May 2021].

docs.rapidminer.com. (n.d.). *Split Validation - RapidMiner Documentation*. [online] Available at: https://docs.rapidminer.com/latest/studio/operators/validation/split_validation.html#:~:text=Split%20Validation%20is%20a%20way [Accessed 2 May 2021].

Dohmen, C. (2019). *Bachelorarbeit Carolin Dohmen Detecting Hate Speech in Social Media -A Machine Learning Approach*. [online] . Available at: https://reposit.haw-hamburg.de/bitstream/20.500.12738/8671/1/thesis.pdf.

Haynes, A. and Schweppe, J. (2017). *Life Cycle of Hate Crimes*. [online] . Available at: https://www.iccl.ie/wp-content/uploads/2018/05/Hate-Crime-Report-LR-WEB.pdf [Accessed 3 May 2021].

Internet Matters. (n.d.). *Impact of online hate on young people*. [online] Available at: https://www.internetmatters.org/hub/question/what-is-the-real-world-impact-of-online-hate-speech-on-young-people/.

Justice, T.D. of (n.d.). *Legislating for Hate Speech and Hate Crime in Ireland Report*. [online] The Department of Justice. Available at: http://www.justice.ie/en/JELR/Pages/Legislating_for_Hate_Speech_and_Hate_Crime_in_Ireland_Report.

Lally, C. (2020). *Number of hate crimes fall 26% on previous year*. [online] The Irish Times. Available at: https://www.irishtimes.com/news/crime-and-law/number-of-hate-crimes-fall-26-on-previous-year-1.4280969 [Accessed 3 May 2021].

Laub, Z. (2019). *Hate Speech on Social Media: Global Comparisons*. [online] Council on Foreign Relations. Available at: https://www.cfr.org/backgrounder/hate-speech-social-media-global-comparisons.

Malmasi, S. and Zampieri, M. (2017). Detecting Hate Speech in Social Media. *RANLP 2017 - Recent Advances in Natural Language Processing Meet Deep Learning*. [online] Available at: https://acl-bg.org/proceedings/2017/RANLP%202017/pdf/RANLP062.pdf.

Matamoros-Fernández, A. and Farkas, J. (2021). Racism, Hate Speech, and Social Media: A Systematic Review and Critique. *Television & New Media*, 22(2), pp.205–224.

Mujtaba, H. (2020). *What is Cross Validation in Machine learning? Types of Cross Validation*. [online] GreatLearning Blog: Free Resources what Matters to shape your Career! Available at: https://www.mygreatlearning.com/blog/cross-validation/ [Accessed 3 May 2021].

Pereira-Kohatsu, J.C., Quijano-Sánchez, L., Liberatore, F. and Camacho-Collados, M. (2019). Detecting and Monitoring Hate Speech in Twitter. *Sensors*, 19(21), p.4654.

Sarang Narkhede (2018). *Understanding Confusion Matrix*. [online] Medium. Available at: https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62.

Siapera, E., Moreo, E. and Zhou, J. (2018). *HATE TRACK: TRACKING AND MONITORING RACIST SPEECH ONLINE i Hate Track Tracking And Monitoring Racist Speech Online*. [online] . Available at: https://www.ihrec.ie/app/uploads/2018/11/HateTrack-Tracking-and-Monitoring-Racist-Hate-Speech-Online.pdf [Accessed 3 May 2021].

Suresh, A. (2020). *What is a confusion matrix?* [online] Medium. Available at: https://medium.com/analytics-vidhya/what-is-a-confusion-matrix-d1c0f8feda5.

Twitter Help Center (2019). *Hateful conduct policy*. [online] Twitter.com. Available at: https://help.twitter.com/en/rules-and-policies/hateful-conduct-policy.

Waseem, Z. and Hovy, D. (2016). *Hateful Symbols or Hateful People? Predictive Features for Hate Speech Detection on Twitter*. [online] Association for Computational Linguistics, pp.88–93. Available at: https://www.aclweb.org/anthology/N16-2013.pdf.

Zach (2020). *An Easy Guide to K-Fold Cross-Validation*. [online] Statology. Available at: https://www.statology.org/k-fold-cross-validation/ [Accessed 3 May 2021].

Facebook.com. (2019). *Community Standards | Facebook*. [online] Available at: https://www.facebook.com/communitystandards/hate_speech.

support.google.com. (n.d.). *Hate speech policy - YouTube Help*. [online] Available at: https://support.google.com/youtube/answer/2801939?hl=en.

Ilga-europe.org. (2019). *Hate crime & hate speech | ILGA-Europe*. [online] Available at: https://www.ilga-europe.org/what-we-do/our-advocacy-work/hate-crime-hate-speech.

Guterres, A. (2019). UNITED NATIONS STRATEGY AND PLAN OF ACTION ON HATE SPEECH. [online] . Available at: https://www.un.org/en/genocideprevention/documents/UN%20Strategy%20and%20Plan%20of%20Action%20on%20Hate%20Speech%2018%20June%20SYNOPSIS.pdf.

Anon., 2020. *nahid hassan.* [Online]
Available at: https://nahid-hassan.github.io/Data-Preprocessing-in-Machine-learning/

Anon., n.d. [Online]
Available at: https://www.aclweb.org/anthology/2020.alw-1.18.pdf

Anon., n.d. [Online]
Available at: https://www.aclweb.org/anthology/2020.alw-1.18.pdf

Anon., n.d. [Online]
Available at: https://www.mdpi.com/2076-0760/9/11/188/htm

Anon., n.d. [Online]
Available at: https://thecleverprogrammer.com/2020/08/19/hate-speech-detection-model/

Anon., n.d. [Online]
Available at: https://link.springer.com/article/10.1007/s11042-020-10082-6

Barrister, 2021. [Online]
Available at: https://www.coursehero.com/file/89074805/Introduction-to-Naive-Bayes-Classification-Algorithm-in-Python-and-R-HackerEarth-Blogpdf/

Chaudhary, M., 2020. [Online]
Available at: https://medium.com/@cmukesh8688/tf-idf-vectorizer-scikit-learn-dbc0244a911a

Double, 2021. [Online]
Available at:
https://www.reddit.com/r/SportsPredictionsheel/comments/myuc75/bayesian_sports_prediction/

Hewitt, E., 2005. [Online]
Available at: https://flylib.com/books/en/1.207.1.175/1/

JavaTPoint, 2018 . [Online]
Available at: https://www.javatpoint.com/data-preprocessing-machine-learning

khan, h., 2020. [Online]
Available at: https://python3.foobrdigital.com/2020/09/page/20/

McCullum, N., 2020. [Online]
Available at: https://www.freecodecamp.org/news/the-ultimate-guide-to-the-numpy-scientific-computing-library-for-python/

Milton, J., 1995. [Online]
Available at: https://planetmath.org/BayesTheorem

Naseem, U., 2020. [Online]
Available at: https://www.x-mol.com/paper/1324533177228955648

ONE, P., 2019. [Online]
Available at: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6701757/

Razzak, U. N. a. I., 2020. [Online]
Available at: https://www.researchgate.net/publication/345317615_A_survey_of_pre-processing_techniques_to_improve_short-
text_quality_a_case_study_on_hate_speech_detection_on_twitter

Routledge, R., 2018. [Online]
Available at: https://www.britannica.com/topic/Bayess-theorem

Science, D., 2020. [Online]
Available at: https://eev.equationpldb.pw/data-science-javatpoint.html

shahbazchandio, 2020 . [Online]
Available at: https://python3.foobrdigital.com/category/pro-main/machine-learning-pro-main/machine-learning-machine-learning-pro-main/ml-introduction/

1. Hewitt, E., 2005. [Online]
Available at: https://flylib.com/books/en/1.207.1.175/1/

2. JavaTPoint, 2018 . [Online]
Available at: https://www.javatpoint.com/data-preprocessing-machine-learning

3. JavaTPoint, n.d. [Online]
Available at: https://www.javatpoint.com/data-preprocessing-machine-learning

4. khan, h., 2020. [Online]
Available at: https://python3.foobrdigital.com/2020/09/page/20/

5. McCullum, N., 2020. [Online]
Available at: https://www.freecodecamp.org/news/the-ultimate-guide-to-the-numpy-scientific-computing-library-for-python/

6. Science, D., 2020. [Online]
Available at: https://eev.equationpldb.pw/data-science-javatpoint.html

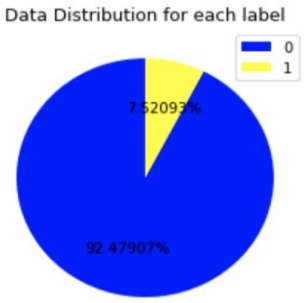
7. shahbazchandio, 2020 . [Online]
Available at: https://python3.foobrdigital.com/category/pro-main/machine-learning-pro-main/machine-learning-machine-learning-pro-main/ml-introduction/

1. Dataset loading:

[1]:
```
import warnings
warnings.filterwarnings("ignore")

import pandas as pd
df= pd.read_csv("train.csv",encoding='latin-1')
df=df[['label','tweet']]
df.columns=["Target","text"]
df.head()



no_hate=df[df['Target']==0]
hate=df[df['Target']==1]
```

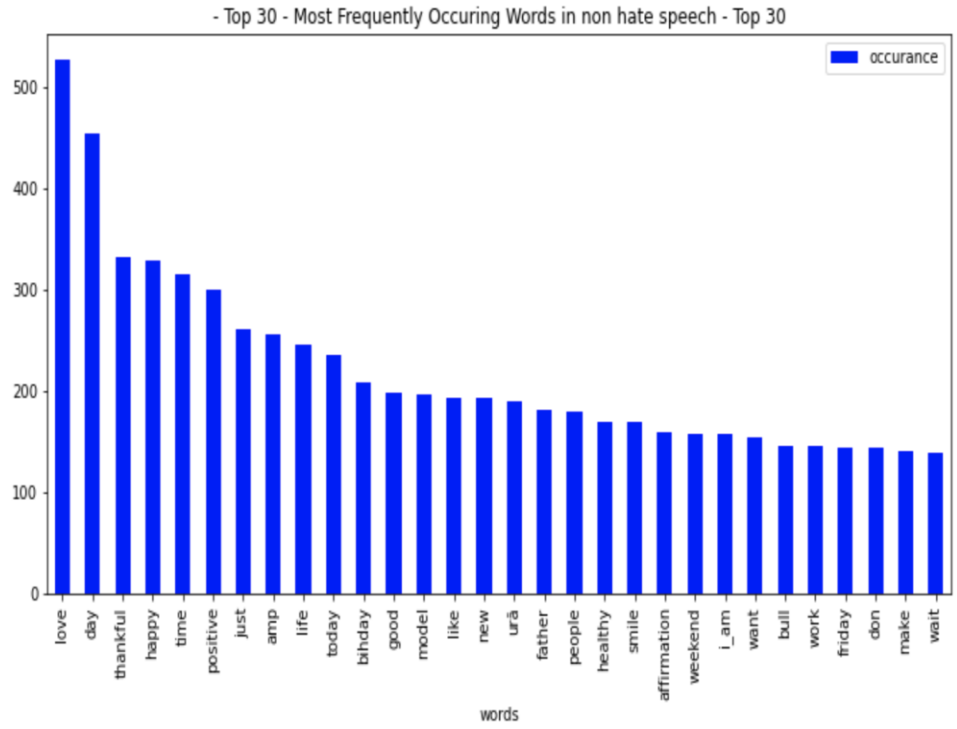
2. Data visualization:

[2]:
```
### Pie chart showing the class distribution in data

import matplotlib.pyplot as plt

class_names = (df['Target'].unique()) ## This line will store the name of the labels, since in our case
## labels are stored in the form of numbers i.e 0 for non hate , 1 for hate speech
## So this variable will store only 0 and 1


## This variable will store the occurance of 0 and 1 in our data set
values_of_classes = list(df['Target'].value_counts())

colors = ['blue', 'yellow', 'brown', 'black']
patches, texts,per = plt.pie(values_of_classes,autopct='%0.5f%%', colors=colors, startangle=90)
plt.legend(patches, class_names, loc="best")


plt.title('Data Distribution for each label')
plt.show()
```



[3]:
```
#Taken from
#https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html#

from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(stop_words = 'english')

from wordcloud import WordCloud
no_hate['text'] = no_hate['text'].str.replace('@user',' ')

#Getting vectors for each word of text
words = vectorizer.fit_transform(no_hate.text)

#Summing the vectors across the rows
sum_words = words.sum(axis=0)

#Getting the frequency of each word
words_frequency = [(word, sum_words[0, i]) for word, i in vectorizer.vocabulary_.items()]

#Sorting the words with in ascending order
words_frequency = sorted(words_frequency, key = lambda x: x[1], reverse = True)

#Making a dataframe of the occurances
occurance = pd.DataFrame(words_frequency, columns=['words', 'occurance'])

occurance.head(30).plot(x='words', y='occurance', kind='bar', figsize=(12, 6), color = 'blue')
plt.title("- Top 30 - Most Frequently Occuring Words in non hate speech - Top 30")
```

[31]:
```
Text(0.5, 1.0, '- Top 30 - Most Frequently Occuring Words in non hate speech - Top 30')
```

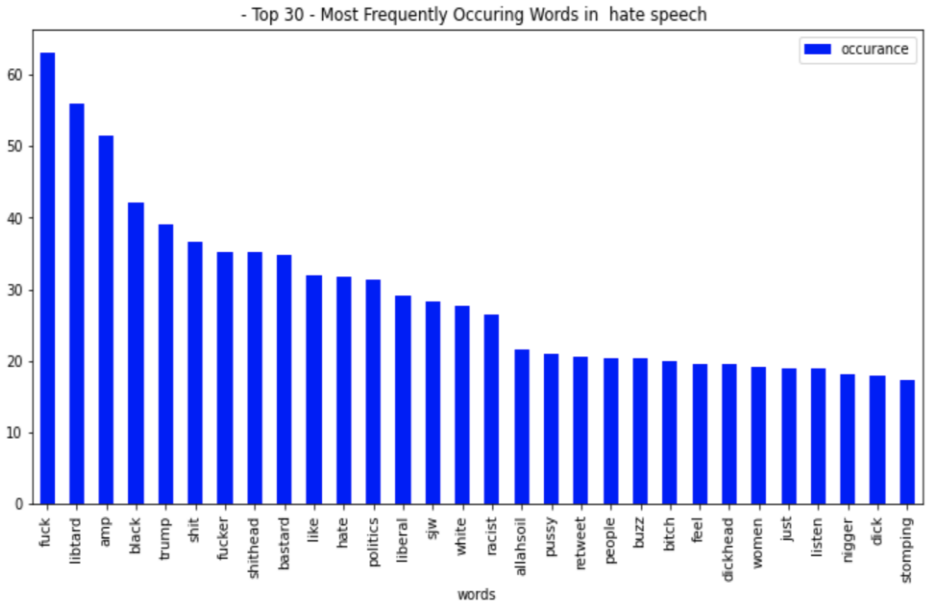


[4]:
```
wordcloud = WordCloud( width = 1200, height = 1200,background_color = 'white').generate_from_frequencies(dict(words_frequency))

plt.figure(figsize=(10,8))
plt.imshow(wordcloud)
plt.title("WordCloud - Frequently Occuring Words in non hate speech", fontsize = 12)
```

[4]: Text(0.5, 1.0, 'WordCloud – Frequently Occuring Words in non hate speech')

[5]:
```
## Replacing the word @user
## Refernece : https://stackoverflow.com/questions/28986489/how-to-replace-text-in-a-column-of-a-pandas-dataframe

hate['text'] = hate['text'].str.replace('@user',' ')


## Getting vectors for each word of text
words = vectorizer.fit_transform(hate.text)
## Summing the vectors across the rows
sum_words = words.sum(axis=0)

## Getting the frequency of each word
words_frequency = [(word, sum_words[0, i]) for word, i in vectorizer.vocabulary_.items()]
## Sorting the words with in ascending order
words_frequency = sorted(words_frequency, key = lambda x: x[1], reverse = True)
## Making a dataframe of the occurances
occurance = pd.DataFrame(words_frequency, columns=['words', 'occurance'])

occurance.head(30).plot(x='words', y='occurance', kind='bar', figsize=(12, 6), color = 'blue')
plt.title("- Top 30 - Most Frequently Occuring Words in  hate speech ")
```

[5]: Text(0.5, 1.0, '- Top 30 - Most Frequently Occuring Words in hate speech ')



[6]:

```
wordcloud = WordCloud(background_color = 'white', width = 1200, height = 1200).generate_from_frequencies(dict(words_frequency))

plt.figure(figsize=(10,8))
plt.imshow(wordcloud)
plt.title("Frequently Occuring Words in hate speech", fontsize = 12)


# In[ ]:
```

3. Data preprocessing:

[7]:
```
import re as reg
import nltk
import time
import seaborn as sns
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
from sklearn.metrics import confusion_matrix
import pandas as pd
from nltk.stem.porter import PorterStemmer
import numpy as np
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import KFold
import joblib as joblib
import matplotlib.pyplot as plt
from bs4 import BeautifulSoup
```

[8]:
```
## Shuffling the data frame

## Reference : https://stackoverflow.com/questions/29576430/shuffle-dataframe-rows

from sklearn.utils import shuffle
df = shuffle(df)
```

[9]:
```
## For processing the tweets the some parts are taken from this
## gitlab repository : https://gitlab.lipi.go.id/ndung/hpc/tree/master


def replacing_emoji(text_of_tweet):
    """
    This function will replace the emojis with suitable text in the tweets
    """
    # Laughing emojis -- :D, : D, :-D, xD, x-D, XD, X-D
    text_of_tweet = reg.sub(r'(:\s?D|:-D|x-?D|X-?D)', ' EMO_POS ', text_of_tweet)
    # Love emojis -- <3, :*
    text_of_tweet = reg.sub(r'(<3|:\*)', ' EMO_POS ', text_of_tweet)
    # Winking emojis -- ;-), ;), ;-D, ;D, (;,  (-;
    text_of_tweet = reg.sub(r'(;-?\)|;-?D|\(-?;)', ' EMO_POS ', text_of_tweet)
    # Smiling emojis -- :), : ), :-), (:, ( :, (-:, :')
    text_of_tweet = reg.sub(r'(:\s?\)|:-\)|\(\s?:|\(-:|:\'\))', ' EMO_POS ', text_of_tweet)
    # Sad emojis-- :-(, : (, :(, ):, )-:
    text_of_tweet = reg.sub(r'(:\s?\(|:-\(|\)\s?:|\)-:)', ' EMO_NEG ', text_of_tweet)
    # Cry emojis-- :,(, :'(, :"(
    text_of_tweet = reg.sub(r'(:,\(|:\'\(|:"\()', ' EMO_NEG ', text_of_tweet)
    return text_of_tweet



def preprocess_tweet_word(tweet_word):
    """
    This function will remove the repitition of words and ',- symbols from the text of tweets
    """
    # Remove punctuation
    tweet_word = tweet_word.strip('\'"?!,.():;')
    # Converting more than two letter repetitions to two letter
    # funnnnny --> funny
    tweet_word = reg.sub(r'(.)\1+', r'\1\1', tweet_word)
    # Remove - & '
    tweet_word = reg.sub(r'(-|\')', '', tweet_word)
    return tweet_word


def is_valid_tweet_word(tweet_word):
    ## Reference :https://stackoverflow.com/questions/27198984/regex-match-if-strings-starts-with-a-letter-or-number-and-then
    # Check if tweet_word begins with an alphabet
    return (reg.search(r'^[a-zA-Z][a-z0-9A-Z\._]*$', tweet_word) is not None)
```

```
def preprocess_text_of_tweet(text_of_tweet,use_stemmer=False):
    """
    This function will remove the URLS , mentioins , hashtags in the text
    """
    processed_text_of_tweet = []
    # Convert to lower case
    text_of_tweet = text_of_tweet.lower()
    # Replaces URLs with the tweet_word URL
    text_of_tweet = reg.sub(r'((www\.[\S]+)|(https?://[\S]+))', ' URL ', text_of_tweet)
    # Replace two+ dots with space
    text_of_tweet = reg.sub(r'\.{2,}', ' ', text_of_tweet)
    # Strip space, " and ' from text_of_tweet
    text_of_tweet = text_of_tweet.strip(' "\'')
    # Replace emojis with either EMO_POS or EMO_NEG
    text_of_tweet = replacing_emoji(text_of_tweet)
    # Replace @handle with the tweet_word USER_MENTION
    text_of_tweet = reg.sub(r'@[\S]+', 'USER_MENTION', text_of_tweet)
    # Replaces #hashtag with hashtag
    text_of_tweet = reg.sub(r'#(\S+)', r' \1 ', text_of_tweet)
    # Remove RT (retext_of_tweet)
    text_of_tweet = reg.sub(r'\brt\b', '', text_of_tweet)
    # Replace multiple spaces with a single space
    text_of_tweet = reg.sub(r'\s+', ' ', text_of_tweet)
    tweet_words = text_of_tweet.split()

    for tweet_word in tweet_words:
        tweet_word = preprocess_tweet_word(tweet_word)
        if is_valid_tweet_word(tweet_word):
            if use_stemmer:
                tweet_word = str(porter_stemmer.stem(tweet_word))
            processed_text_of_tweet.append(tweet_word)

    return ' '.join(processed_text_of_tweet)


def preprocess_csv(lines,use_stemmer=False):
    """
    This function will make a data frame for the processed data
    """
    total = len(lines)
    pre_processed=[]
    for i in range(total):
        pre_processed.append(preprocess_text_of_tweet(lines[i]))

    return pre_processed
```

4. Training ,comparing the models performance :

[10]:
```
## Guide for performing cross validation was taken from this reference :
## https://my.oschina.net/Bettyty/blog/751627

def k_fold_cross_validation(training_columns, labels, vectorizer, classifier, name_of_model):


    kf = KFold( shuffle=True,n_splits=10) # Initializing KFold object with 10 as k-value
    accuracies=[]
    training_columns_train_array = []
    labels_train_array = []
    training_columns_test_array = []
    labels_test_array = []
    iter = 0
    print("Applying K fold cv algorithm for :", (name_of_model))
    for training_indices, test_indices in kf.split(training_columns):
        iter += 1
        print("iter ", iter)
        training_columns_train_cv, labels_train_cv = training_columns.iloc[training_indices], labels.iloc[training_indices]
        training_columns_test_cv, labels_test_cv = training_columns.iloc[test_indices], labels.iloc[test_indices]
        training_columns_train_array.append(training_columns_train_cv) # adding training score for  the given iteration
        training_columns_test_array.append(training_columns_test_cv) # adding test score for iteration
        labels_train_array.append(labels_train_cv)  # adding training labels for  the given iteration
        labels_test_array.append(labels_test_cv)  # adding test labels for  the given iteration
        vectorizer.fit(training_columns_train_cv) # learning vocabulary of training set
        training_columns_train_vectorizer_cv = vectorizer.transform(training_columns_train_cv)
        print("Shape of training training_columns: ", training_columns_train_vectorizer_cv.shape)
        training_columns_test_vectorizer_cv = vectorizer.transform(training_columns_test_cv)
        print("Shape of test training_columns: ", training_columns_test_vectorizer_cv.shape)
        classifier.fit(training_columns_train_vectorizer_cv.toarray(), labels_train_cv)
        score = classifier.score(training_columns_test_vectorizer_cv.toarray(), labels_test_cv) # Calculating accuracy
        accuracies.append(score) # adding k-fold  accuracy for each iter
    print("List of k-fold  accuracies for {}: ".format(name_of_model), accuracies)
    average_accuracy = np.mean(accuracies)
    print("Average k-fold  accuracy for {}: ".format(name_of_model), average_accuracy)
    print("Best k-fold  accuracy for {}: ".format(name_of_model), max(accuracies))
    maximum_accuracy_index = accuracies.index(max(accuracies)) # best k-fold  accuracy
    maximum_accuracy_training_columns_train = training_columns_train_array[maximum_accuracy_index] # training training_columns corresponding to best k-fold  accuracy
    maximum_accuracy_training_columns_test = training_columns_test_array[maximum_accuracy_index] # test training_columns corresponding to best k-fold  accuracy
    maximum_accuracy_labels_train = labels_train_array[maximum_accuracy_index] # training labels corresponding to best k-fold  accuracy
    maximum_accuracy_labels_test = labels_test_array[maximum_accuracy_index] # test labels corresponding to best k-fold  accuracy



    return average_accuracy, maximum_accuracy_training_columns_train, maximum_accuracy_training_columns_test, maximum_accuracy_labels_train, maximum_accuracy_labels_test,accuracies
```

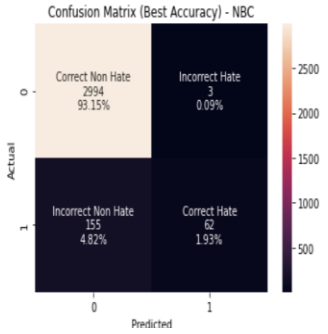
```
    return average_accuracy, maximum_accuracy_training_columns_train, maximum_accuracy_training_columns_test, maximum_accuracy_labels_train, maximum_accuracy_labels_test,accuracies


def evaluate(maximum_accuracy_training_columns_train, maximum_accuracy_training_columns_test, maximum_accuracy_labels_train, maximum_accuracy_labels_test, vectorizer, labels, clf, name_of_model): #### Creates Confusion matrix for SVC
    vectorizer.fit(maximum_accuracy_training_columns_train)
    maximum_accuracy_training_columns_train_vectorizer = vectorizer.transform(maximum_accuracy_training_columns_train)
    maximum_accuracy_training_columns_test_vectorizer = vectorizer.transform(maximum_accuracy_training_columns_test)
    clf.fit(maximum_accuracy_training_columns_train_vectorizer, maximum_accuracy_labels_train)
    labels_pred = clf.predict(maximum_accuracy_training_columns_test_vectorizer) # Prediction on test training_columns
    ## Plotting confusion matrix is taken from :
    ## https://medium.com/@dtuk81/confusion-matrix-visualization-fc31e3f30fea
    cf_matrix = confusion_matrix(maximum_accuracy_labels_test, labels_pred)
    d={ 0: 'Non Hate', 1: 'Hate'}
    sentiment_df = labels.drop_duplicates().sort_values()

    group_names=["Correct Non Hate","Incorrect Hate","Incorrect Non Hate","Correct Hate"]

    group_counts = ["{0:0.0f}".format(value) for value in
                cf_matrix.flatten()]
    group_percentages = ["{0:.2%}".format(value) for value in
                     cf_matrix.flatten()/np.sum(cf_matrix)]
    labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in zip(group_names,group_counts,group_percentages)]
    labels = np.asarray(labels).reshape(2,2)


    sns.heatmap(cf_matrix, annot=labels, fmt='', xticklabels=sentiment_df.values, yticklabels=sentiment_df.values)
    plt.ylabel('Actual')
    plt.xlabel('Predicted')
    plt.title("Confusion Matrix (Best Accuracy) - {}".format(name_of_model))
    plt.show()
    return
```

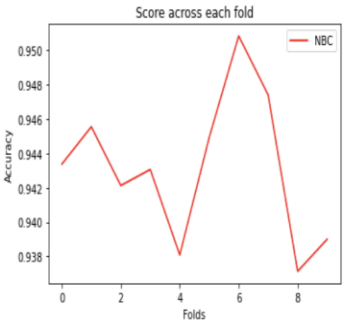
11]:
```
df['normalized']=preprocess_csv(df['text'].values)
```

12]:
```
from sklearn.naive_bayes import MultinomialNB

data = df.normalized
targets = df.Target

tfidf = TfidfVectorizer(sublinear_tf=True, min_df=30, norm='l2', ngram_range=(1,3)) # min_df=30 is a clever way of feature engineering

NBC_clf = MultinomialNB() # NBC Model
NBC_average_acc, training_data_with_max_acc, test_data_with_max_acc, targets_with_max_acc_train, targets_with_max_acc_test,nbc_accuracies = k_fold_cross_validation(data, targets, tfidf, NBC_clf, "NBC") # NBC cross-validation
evaluate(training_data_with_max_acc, test_data_with_max_acc, targets_with_max_acc_train, targets_with_max_acc_test, tfidf, targets, NBC_clf, "NBC") # NBC confusion matrix

plt.plot(nbc_accuracies,color='r',label="NBC")
plt.legend(loc="best")
plt.xlabel("Folds") ## Setting x-label of fig 1
plt.ylabel("Accuracy"); ## Setting y-label of fig 1
plt.title("Score across each fold") ## Setting subtitle of fig 1
```

```
plt.legend(loc="best")
plt.xlabel("Folds") ## Setting x-label of fig 1
plt.ylabel("Accuracy"); ## Setting y-label of fig 1
plt.title("Score across each fold") ## Setting subtitle of fig 1
```

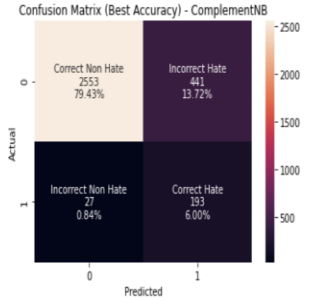
```
Applying K fold cv algorithm for : NBC
iter  1
Shape of training training_columns:  (28923, 2204)
Shape of test training_columns:  (3214, 2204)
iter  2
Shape of training training_columns:  (28923, 2222)
Shape of test training_columns:  (3214, 2222)
iter  3
Shape of training training_columns:  (28923, 2216)
Shape of test training_columns:  (3214, 2216)
iter  4
Shape of training training_columns:  (28923, 2201)
Shape of test training_columns:  (3214, 2201)
iter  5
Shape of training training_columns:  (28923, 2209)
Shape of test training_columns:  (3214, 2209)
iter  6
Shape of training training_columns:  (28923, 2204)
Shape of test training_columns:  (3214, 2204)
iter  7
Shape of training training_columns:  (28923, 2209)
Shape of test training_columns:  (3214, 2209)
iter  8
Shape of training training_columns:  (28924, 2219)
Shape of test training_columns:  (3213, 2219)
iter  9
Shape of training training_columns:  (28924, 2213)
Shape of test training_columns:  (3213, 2213)
iter  10
Shape of training training_columns:  (28924, 2215)
Shape of test training_columns:  (3213, 2215)
List of k-fold  accuracies for NBC:  [0.9433727442439328, 0.9455507156191661, 0.9421281891723708, 0.9430616054760423, 0.9380833851897946, 0.9449284380833852, 0.9508400746733043, 0.9474011826953004, 0.937130407718643, 0.938997821350762
Average k-fold  accuracy for NBC:  0.9431494564222701
Best k-fold  accuracy for NBC:  0.9508400746733043
```

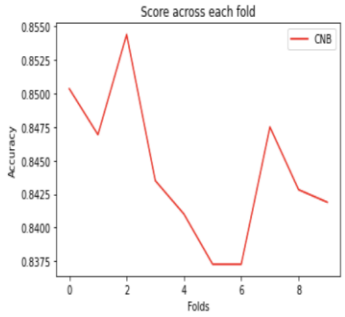




[14]:
```
#ref: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.ComplementNB.html#sklearn.naive_bayes.ComplementNB


import numpy as np
from sklearn.naive_bayes import ComplementNB


CNB_rng = np.random.RandomState(1)

xlabel = CNB_rng.randint(5, size=(6, 100))
ylabel = np.array([1, 2, 3, 4, 5, 6])


B_clf = ComplementNB() # CNB Model
B_clf.fit(xlabel, ylabel, sample_weight=None)
B_clf.fit(xlabel, ylabel)
CNB_average_acc, training_data_with_max_acc, test_data_with_max_acc, targets_with_max_acc_train, targets_with_max_acc_test,bnc_accuracies = k_fold_cross_validation(data, targets, tfidf, B_clf, "ComplementNB") # NBC cross-validation
evaluate(training_data_with_max_acc, test_data_with_max_acc, targets_with_max_acc_train, targets_with_max_acc_test, tfidf, targets, B_clf, "ComplementNB") # NBC confusion matrix


plt.plot(bnc_accuracies,color='r',label="CNB")
plt.legend(loc="best")

plt.title("Score across each fold") ## Setting subtitle of fig 1
plt.xlabel("Folds")
plt.ylabel("Accuracy"); ## Setting y-label of fig 1
```

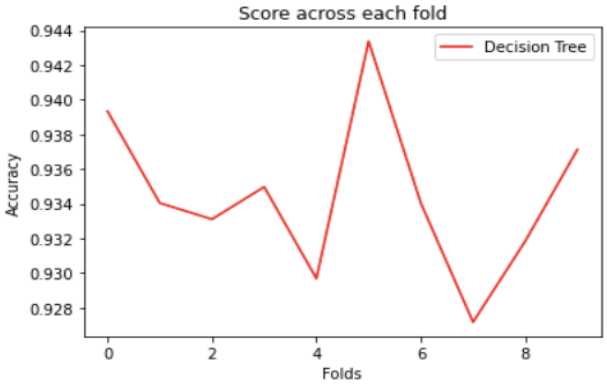
```
Applying K fold cv algorithm for : ComplementNB
iter  1
Shape of training training_columns:  (28923, 2204)
Shape of test training_columns:  (3214, 2204)
iter  2
Shape of training training_columns:  (28923, 2209)
Shape of test training_columns:  (3214, 2209)
iter  3
Shape of training training_columns:  (28923, 2202)
Shape of test training_columns:  (3214, 2202)
iter  4
Shape of training training_columns:  (28923, 2209)
Shape of test training_columns:  (3214, 2209)
iter  5
Shape of training training_columns:  (28923, 2193)
Shape of test training_columns:  (3214, 2193)
iter  6
Shape of training training_columns:  (28923, 2209)
Shape of test training_columns:  (3214, 2209)
iter  7
Shape of training training_columns:  (28923, 2214)
Shape of test training_columns:  (3214, 2214)
iter  8
Shape of training training_columns:  (28924, 2214)
Shape of test training_columns:  (3213, 2214)
iter  9
Shape of training training_columns:  (28924, 2216)
Shape of test training_columns:  (3213, 2216)
iter  10
Shape of training training_columns:  (28924, 2219)
Shape of test training_columns:  (3213, 2219)
List of k-fold  accuracies for ComplementNB:  [0.8503422526446796, 0.8469197261978842, 0.8543870566272558, 0.8434971997510889, 0.8410080896079651, 0.8372744243932794, 0.8372744243932794, 0.8474945533769063, 0.8428260192966075, 0.8418923124805477]
Average k-fold  accuracy for ComplementNB:  0.8442916058769494
Best k-fold  accuracy for ComplementNB:  0.8543870566272558
```





[ ]:
```
#ref: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix ##


LR_clf = LogisticRegression(random_state=0).fit(xlabel, ylabel)
LR_clf.predict(xlabel[:2, :])
LR_clf.predict_proba(xlabel[:2, :])



LR_average_acc, training_data_with_max_acc, test_data_with_max_acc, targets_with_max_acc_train, targets_with_max_acc_test,lr_accuracies = k_fold_cross_validation(data, targets, tfidf, LR_clf, "LR") # cross-validation
evaluate(training_data_with_max_acc, test_data_with_max_acc, targets_with_max_acc_train, targets_with_max_acc_test, tfidf, targets, LR_clf, "LR") # LR confusion matrix

## Comparision through graphs

plt.plot(lr_accuracies,color='r',label="LR")
plt.legend(loc="best")

plt.title("Score across each fold") ## Setting subtitle of fig 1
plt.xlabel("Folds")
plt.ylabel("Accuracy"); ## Setting y-label of fig 1


print("Coeficient results")
print(LR_clf.coef_)

print("Intercepts results")
print(LR_clf.intercept_)
```

[15]:
```
#Ref: https://scikit-learn.org/stable/modules/tree.html
#ref: https://scikit-learn.org/stable/auto_examples/datasets/plot_iris_dataset.html

from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import export_text
from sklearn.datasets import load_iris
#from sklearn import tree

xlabel = [[0, 0], [1, 1]]
ylabel = [0, 1]
DT_iris = load_iris()
xlabel, ylabel = DT_iris.data, DT_iris.target




DT_clf = DecisionTreeClassifier(random_state=0, max_depth=4)
DT = DT_clf.fit(xlabel, ylabel) #add 1
DT_clf = DT_clf.fit(DT_iris.data,DT_iris.target)
DT_average_acc, training_data_with_max_acc, test_data_with_max_acc, targets_with_max_acc_train, targets_with_max_acc_test,dt_accuracies = k_fold_cross_validation(data, targets, tfidf, DT_clf, "DT") # cross-validation
evaluate(training_data_with_max_acc, test_data_with_max_acc, targets_with_max_acc_train, targets_with_max_acc_test, tfidf, targets, DT_clf, "DT") # LR confusion matrix

## Comparision through graphs



plt.plot(dt_accuracies,color='r',label="Decision Tree")
plt.legend(loc="best")

plt.title("Score across each fold") ## Setting subtitle of fig 1
plt.xlabel("Folds")
plt.ylabel("Accuracy"); ## Setting y-label of fig 1

#tree.plot_tree(DT_clf)
```



[ ]:
```
## argument has accuracy values
plt.plot(nbc_accuracies,color='r',label="MNB")
plt.plot(bnc_accuracies,color='g',label="CNB")
plt.plot(lr_accuracies,color='b',label="LR")
plt.plot(dt_accuracies,color='m',label="DT")
plt.legend(loc="best")

plt.title("Score across each fold") ## Setting subtitle of fig 1
plt.xlabel("Folds") #Setting label info
plt.ylabel("Accuracy"); ## Setting y-label of fig 1
```

5. Fine tuning and saving the model:

[ ]:
```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import reciprocal, uniform
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(tfidf.fit_transform(data), targets, test_size=0.1, random_state=101)

## Hyper-parametertuning
param_distributions = { "alpha": uniform(1, 10), "fit_prior":[True,False]}
rnd_search_cv = RandomizedSearchCV(NBC_clf, param_distributions, n_iter=10, verbose=2, cv=3)
rnd_search_cv.fit(X_train, y_train)



rnd_search_cv.best_estimator_
```

[ ]:
```
rnd_search_cv.best_estimator_.fit(X_train, y_train)
```

[ ]:
```
import pickle
# now you can save it to a file
with open('hate_nohate_model.pkl', 'wb') as f:
    pickle.dump(rnd_search_cv.best_estimator_, f)
```