LAU
School of Arts and Sciences

Department of Computer Science & Mathematics

CSC461- Introduction to Machine Learning

Machine Learning Project

Detecting Hate Speech

Spring 2023	Mia Najwa Zebian	202003137
Submitted to Dr. Abbas Rammal	Mohamad Yateem	202002828

Table of Contents

Collecting Data

The Dataset

In recent years, it has become more and more common to utilize machine learning algorithms to identify hate speech online. The caliber and reliability of the training datasets, however, have a significant impact on these algorithms' ability to perform well. That’s why the dataset we selected for our hate speech detection model was the dynamically generated Hate Speech Dataset. The Dynamically Generated Hate Speech Dataset, developed by Bertie Vidgen, Tristan Thrush, Zeerak Waseem, and Douwe Kiela, is the first-of-its-kind big synthetic training dataset for online hate categorization. It consists of texts from 2020.

The entries in the dynamically generated hate speech dataset include entry ID, label, type, annotator ID, status, round, split, and round model predictions, as well as if the model was duped. In successive rounds of dynamic data gathering, the entries were constructed from scratch by trained annotators. The dataset is presented in two tables, with the first table providing the dataset of items and the second table being left empty.

The authors conducted many rounds of dynamic data collecting and utilized trained annotators to construct the entries in order to verify the dataset's validity. The dataset will be diversified and indicative of real-world situations thanks to this method. The entry ID, label, type, annotator ID, status, round, split, and round model predictions, as well as whether the model was tricked, have also been included in the authors' full description of the dataset. This data enables researchers to better comprehend the dataset and make efficient use of it while training machine learning models.

In addition to finding the dataset, a model using the dataset was also found. The information generated from the model are stored in the dataset and we used this information to compare the model given with the model we created.

Link to the dataset:
https://www.kaggle.com/datasets/usharengaraju/dynamically-generated-hate-speech-dataset

Link to a model using the dataset:

https://www.kaggle.com/code/tarushi89/hate-speech-data-analysis/input

Main Imports

There are many imports for this project, displayed at various stages of the projects:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.utils import resample
from sklearn.model_selection import train_test_split
import re
import nltk
from textblob import TextBlob
from textblob import Word
from nltk.tokenize import wordpunct_tokenize
from nltk.corpus import stopwords
nltk.download('omw-1.4')
nltk.download('wordnet')
nltk.download('stopwords')
nltk.download('punkt')
lemmatizer=nltk.WordNetLemmatizer()
from textblob import Word
from wordcloud import WordCloud
from wordcloud import STOPWORDS
import re
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import recall_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
```

```python
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import precision_recall_fscore_support
from sklearn.metrics import accuracy_score
```

Brief Explanation of each import:

1. `pandas` is a library used for data manipulation and analysis. It provides data structures for efficiently storing and manipulating large datasets.

2. `numpy` is a library used for numerical computing in Python. It provides support for large, multi-dimensional arrays and matrices, as well as a wide range of mathematical functions.

3. `matplotlib` is a plotting library used for creating visualizations in Python. It provides a wide range of tools for creating line plots, scatter plots, bar charts, and more.

4. `seaborn` is a data visualization library based on matplotlib. It provides a high-level interface for creating attractive and informative statistical graphics.

5. `sklearn` (short for scikit-learn) is a machine learning library for Python. It provides a wide range of tools for data preprocessing, feature selection, model selection, and evaluation.

6. `re` is a module used for regular expressions in Python. It provides a way to search and manipulate text using patterns.

7. `nltk` (short for Natural Language Toolkit) is a library used for natural language processing in Python. It provides tools for tokenization, stemming, lemmatization, and more.

8. `textblob` is a library built on top of NLTK that provides a simple API for common natural language processing tasks, such as sentiment analysis and part-of-speech tagging.

9. `wordcloud` is a library used for creating word clouds in Python. It provides a way to visualize the most common words in a text corpus.

10. `StandardScaler` is a class used for scaling data in machine learning. It scales the data to have zero mean and unit variance.

11. `LogisticRegression` is a class used for logistic regression in machine learning. It is a classification algorithm used to predict binary outcomes.

12. `confusion_matrix` is a function used to evaluate the performance of a classification model. It provides a way to visualize the number of true positives, true negatives, false positives, and false negatives.

13. `metrics` is a module in scikit-learn that provides a wide range of evaluation metrics for machine learning models, such as accuracy, precision, recall, and F1 score.

14. `SVC` is a class used for support vector classification in machine learning. It is a classification algorithm that finds the hyperplane that maximally separates the classes.

15. `GaussianNB` is a class used for Gaussian Naive Bayes classification in machine learning. It is a classification algorithm based on Bayes' theorem.

16. `KNeighborsClassifier` is a class used for k-nearest neighbors classification in machine learning. It is a classification algorithm that finds the k nearest neighbors to a given data point and assigns the class based on the majority vote of those neighbors.

Preparing the Data

Getting the Dataset

```
twitter1=pd.read_csv('2020-12-31-DynamicallyGeneratedHateDataset-entries-v0.1.csv',index_col=False)
```

The dataset was imported and stored in twitter1.

Then the data was shuffled to make sure that ordering doesn't affect the learning process

```
       Unnamed: 0      id                                            text  \
3028         3062   13857  I do not differentiate a black man and a black...
17319       17379   25420                              fucking disgusting!
19842       19904  27038p  my colleague is a transgender bisexual woman, ...
1294         1317   12068                               That chair is vile
38930       39029    8182               This lady is f**cking awesome in bed
...           ...     ...                                              ...
6401         6450   19845              Many of my closest colleagues are muslim
30463       30531  33712p  it really shows how shallow greedy and PR-orie...
29160       29227   32986  I'm completely mentally healthy. And I'm happy...
6070         6119  19669p  There are many black people on my street which...
15642       15698   24584  It is so annoying having to explain myself and...

         label        type model_wrong  db.model_preds           status round  \
3028      hate    notgiven        True         0.99467  dynabench entry     1
17319  nothate        none       False         0.99959  dynabench entry    2a
19842     hate  derogation         NaN             NaN     perturbation    2b
1294   nothate        none       False         0.86704  dynabench entry     1
38930  nothate        none        True         0.00474  dynabench entry     1
...        ...         ...         ...             ...              ...   ...
6401   nothate        none       False         0.99974  dynabench entry    2a
30463  nothate        none         NaN             NaN     perturbation    3b
29160     hate  derogation        True         0.00001  dynabench entry    3a
6070      hate  derogation         NaN             NaN     perturbation    2b
15642     hate  derogation        True         0.99885  dynabench entry    2a
...
6070   train  lqlkttromx
15642  train  elgzdd8tvb
```

Understanding the Dataset

To understand the dataset we need to look at what it consists of.

We first check the columns:

```
Index(['Unnamed: 0', 'id', 'text', 'label', 'type', 'model_wrong',
       'db.model_preds', 'status', 'round', 'split', 'annotator'],
      dtype='object')
```

the unquiesces of each column

```
print(twitter1.shape)
twitter1.nunique(axis=0)
```

```
(40623, 11)

Unnamed: 0        40623
id                40623
text              40463
label                 2
type                  7
model_wrong           2
db.model_preds    12118
status                2
round                 5
split                 3
annotator            20
dtype: int64
```

And the data type of each column

```
twitter1.dtypes
```

```
Unnamed: 0          int64
id                 object
text               object
label              object
type               object
model_wrong        object
db.model_preds    float64
status             object
round              object
split              object
annotator          object
dtype: object
```

To make the data binary and easier to work with we convert the hate to 0 and the nothate to 1:

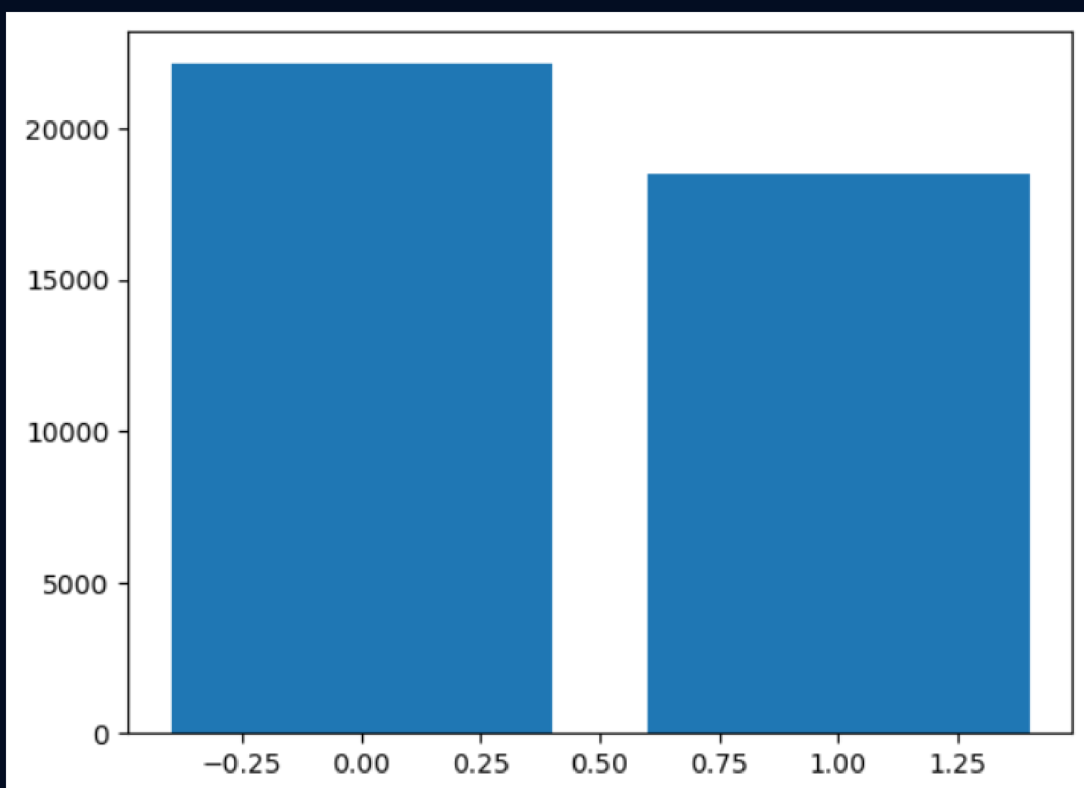
```
twitter1['label'] = twitter1['label'].replace({'hate': 0, 'nothate': 1})
```

Checking The balance

We of course need to check the balance of the data to make sure if it is oversampled or under-sampled

We first get the count of hate and nothate and display them in a bar graph to see if the difference is significant.

```
0    22124
1    18499
Name: label, dtype: int64
```



From this we can tell that the data is under-sampled, however, before balancing it we will first clean the data to see if it will fix the problem.

Working on The Dataset

As explained earlier, the dataset already had a model used on it. So, we divided the dataset into to the part we are going to you to create a model and the other which contains the entire dataset.

We are going to clean both datasets and visualize both to show the difference between them what the given model showed us.

The main dataset

The main dataset consists of only the label and text and is what were going to use to create our model.

Clean the dataset

We first get the dataset we want by cleaning it.

Dropping the unwanted columns

```python
ctwitter1=twitter1.drop(['Unnamed: 0','id', 'type', 'model_wrong','db.model_preds', 'statu
```

Python

```python
ctwitter1.head()
```

Python

	text	label
3028	I do not differentiate a black man and a black...	0
17319	fucking disgusting!	1
19842	my colleague is a transgender bisexual woman, ...	0
1294	That chair is vile	1
38930	This lady is f**cking awesome in bed	1

```
ctwitter1.describe()
```

	label
count	40623.000000
mean	0.455382
std	0.498011
min	0.000000
25%	0.000000
50%	0.000000
75%	1.000000
max	1.000000

We dropped all the unwanted information and kept only the label and text. The data is stored in ctwitter1. We also checked what the dataset looks like now.

Check for null

We of course check for null values and found none.

```
print(ctwitter1.isnull().sum())
```

```
text     0
label    0
dtype: int64
```

Check for duplicates

We however did find duplicate rows and dropped them.

check for duplicates

```
duplicates = ctwitter1[ctwitter1.duplicated()]
if len(duplicates) > 0:
    print('Duplicate rows:')
    print(duplicates)
else:
    print('No duplicate rows found.')
[15]
```

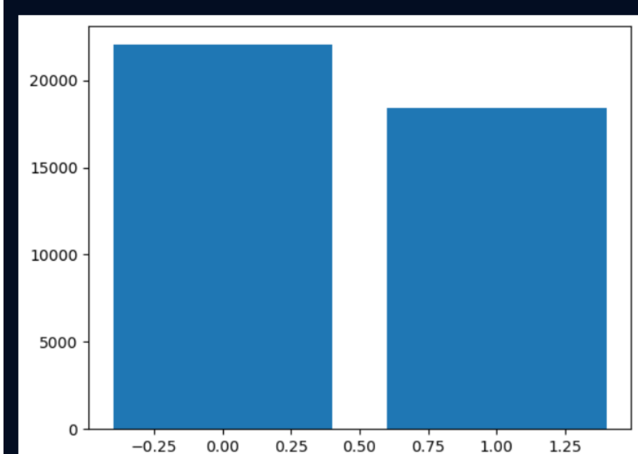
```
Duplicate rows:
                                                text  label
512                                   fuck off Boris      1
38897                I endorse the genocide of Muslims    0
29366                   trans women are not real women    0
4609   Indians are doing great at disrupting our society  0
4293                              All Zionists are cheap  0
...                                                ...    ...
1041   Pakistanis are just not as clean as British pe...  0
38546               Millenials were born in a comatose state  0
4973   Wish I hadn�t bought this car, it couldn�t be ...  1
2310   I wouldn't eat food that has been touched by a...  0
4237                               Women are repulsive    0

[157 rows x 2 columns]
```

```
ctwitter1 = ctwitter1.drop_duplicates(keep='first')
```

Check the balance

We replotted to check the balance and found the dataset still under-sampled.

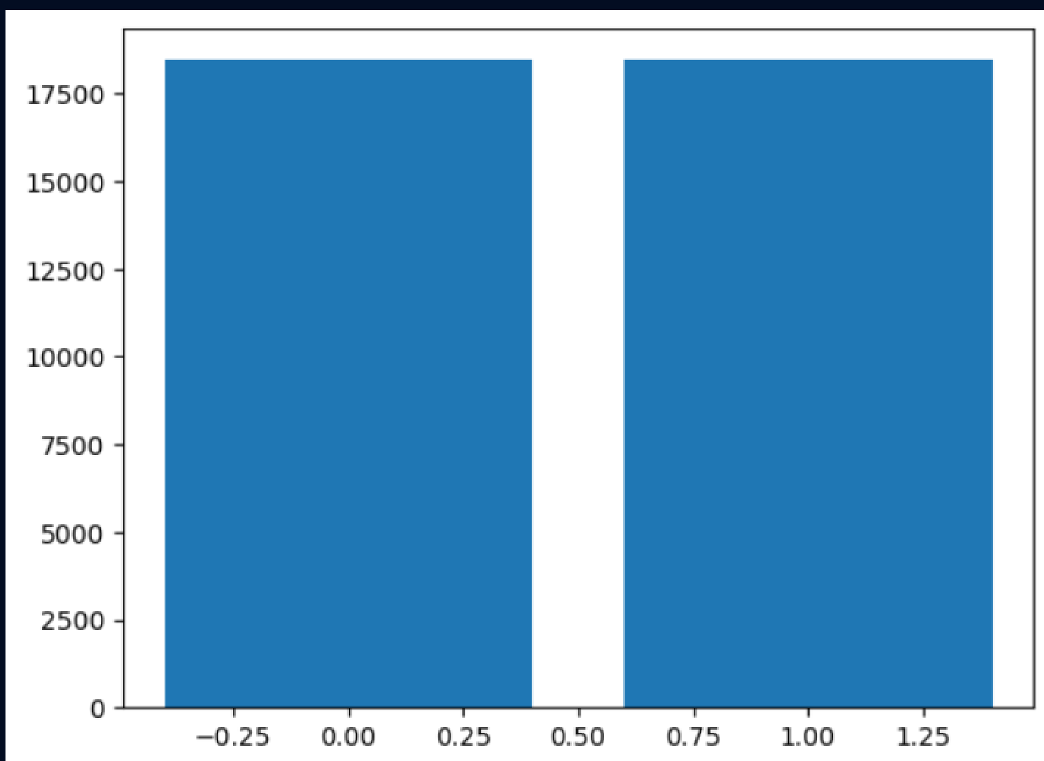
```
0    22025
1    18441
Name: label, dtype: int64
```



So at this point we balanced the dataset.

```python
majority_class = ctwitter1[ctwitter1['label'] == 0]
minority_class = ctwitter1[ctwitter1['label'] == 1]
undersampled_majority = resample(majority_class,
                                 replace=False,
                                 n_samples=len(minority_class),
                                 random_state=42)
balanced_df = pd.concat([undersampled_majority, minority_class])
balanced_df.to_csv('Balanced (2).csv', index=False)
ctwitter=pd.read_csv('Balanced (2).csv',index_col=False)
```

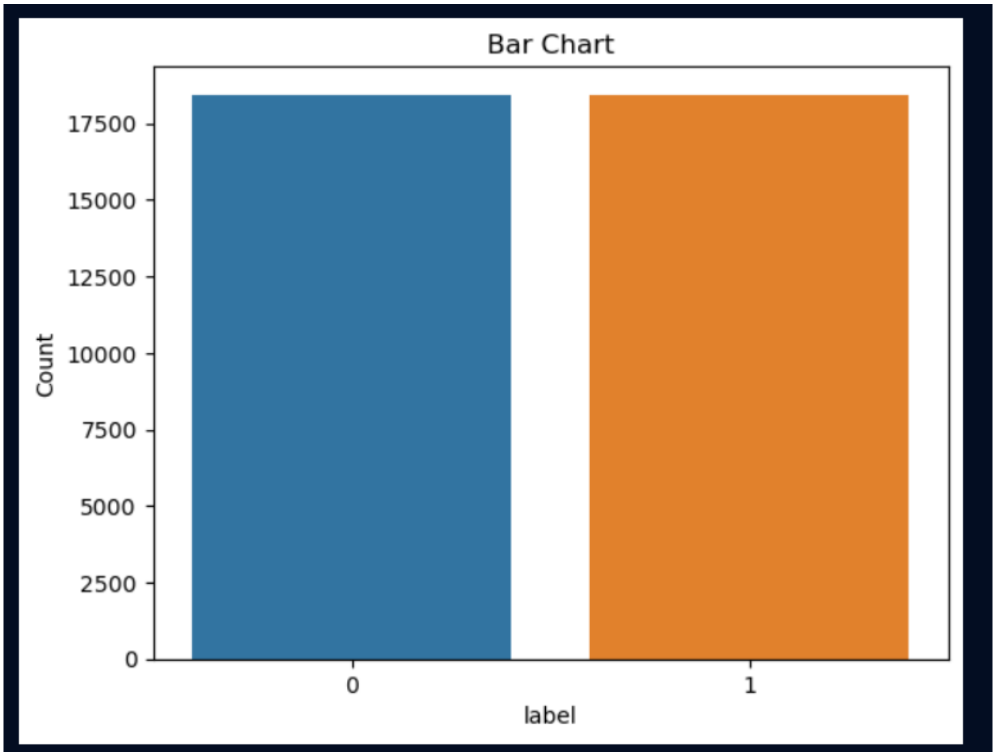
```
0    18441
1    18441
Name: label, dtype: int64
```

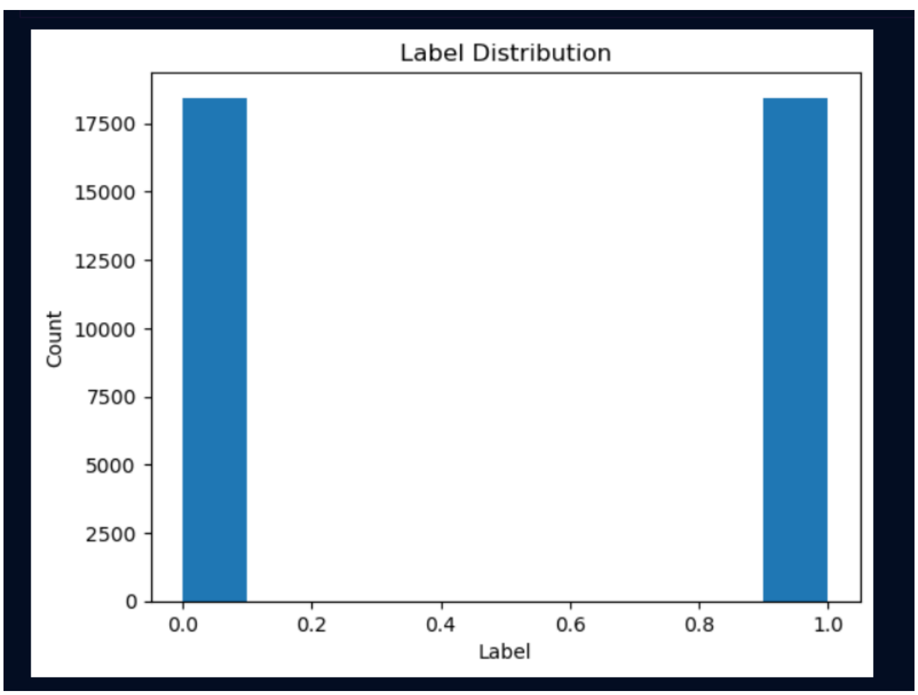


Visualize the dataset

Bar chart and histogram

We created a bar chart and histogram to display the label, which showed us that the data for hate and not hate was equal as expected.





Cleaning and visualizing the dataset using NLP

We used NLP to clean the text so that is readable by our algorithm. As well as, to represent the dataset further.

Convert to lower case

We start by converting all the characters to lower case to make it readable by our algorithms.

```python
ctwitter["text"] = ctwitter["text"].apply(lambda i: " ".join(i.lower() for i in i.split()))
print(ctwitter['text'])
```

```
0                          i do not want to live near koshies
1              i love arabs. they are so hardworking. i love ...
2              oh women i hope you like hell for all these ab...
3                            why should indian people get rights?
4                                   urgh, blacks are so tiresome
                                     ...
36877    i'm looking for new best friends who love to g...
36878                                            cigars stink
36879                               so happy to have white teeth
36880                      many of my closest colleagues are muslim
36881    it really shows how shallow greedy and pr-orie...
Name: text, Length: 36882, dtype: object
```

Remove numeric values

We used regex to match all numerical and non-alphabetical values and remove them from the text.

```python
ctwitter["text"] = ctwitter["text"].str.replace(r"\d","")
ctwitter["text"] = ctwitter["text"].str.replace(r"[^\w\s]","")
ctwitter["text"] = ctwitter["text"].str.replace(r"[^[A-Za-z]\s]","")
print(ctwitter['text'])
```

```
0                          i do not want to live near koshies
1              i love arabs they are so hardworking i love th...
2              oh women i hope you like hell for all these ab...
3                             why should indian people get rights
4                                    urgh blacks are so tiresome
                                     ...
36877    im looking for new best friends who love to go...
36878                                            cigars stink
36879                               so happy to have white teeth
36880                      many of my closest colleagues are muslim
36881    it really shows how shallow greedy and prorien...
Name: text, Length: 36882, dtype: object
```

Remove all non-english words

We further cleaned the text by removing any non-English words from it.

```
import re
pattern = re.compile(r'\b[a-zA-Z]+\b')


def remove_non_english(text):
    return ' '.join(pattern.findall(text))

# apply the function to each text document
ctwitter['text'] = ctwitter['text'].apply(remove_non_english)

print(ctwitter)
```

```
                                                    text  label
0                      i do not want to live near koshies      0
1      i love arabs they are so hardworking i love th...      0
2      oh women i hope you like hell for all these ab...      0
3                    why should indian people get rights      0
4                            urgh blacks are so tiresome      0
...                                                  ...    ...
36877  im looking for new best friends who love to go...      1
36878                                       cigars stink      1
36879                          so happy to have white teeth   1
36880             many of my closest colleagues are muslim    1
36881  it really shows how shallow greedy and prorien...      1

[36882 rows x 2 columns]
```

Remove stop words

We removed stopwords such as “when, and, not, but, etc…”.

We remove stopwords. Any words like "when, and, not, but, etc..." were removed

```
sw = stopwords.words("english")
ctwitter["text"] = ctwitter["text"].apply(lambda x: " ".join(x for x in x.split() if x not
print(ctwitter['text'])
```

Python

```
0                              want live near koshies
1               love arabs hardworking love cleaning toilets
2                   oh women hope like hell aborted babies
3                                  indian people get rights
4                                     urgh blacks tiresome
                              ...
36877    im looking new best friends love go iceskating...
36878                                          cigars stink
36879                                     happy white teeth
36880                           many closest colleagues muslim
36881    really shows shallow greedy proriented psychol...
Name: text, Length: 36882, dtype: object
```

Lemmatize the words

We lemmatize the words, meaning we reverted all words back to their original form.

Changing becomes change

Walks becomes walk

Running becomes run

```
from textblob import Word
ctwitter["text"] = ctwitter["text"].apply(lambda x: " ".join([Word(x).lemmatize()]))
print(ctwitter['text'])
```

[31] Python

```
0                              want live near koshies
1                love arabs hardworking love cleaning toilets
2                   oh women hope like hell aborted babies
3                                  indian people get rights
4                                     urgh blacks tiresome
                               ...
36877    im looking new best friends love go iceskating...
36878                                          cigars stink
36879                                     happy white teeth
36880                          many closest colleagues muslim
36881    really shows shallow greedy proriented psychol...
Name: text, Length: 36882, dtype: object
```

Create tokens

We split up the words into tokens and store them in the data set.

```
ctwitter["tokens"] = ctwitter["text"].apply(lambda x: TextBlob(x).words)
```

Calculate the frequency

We calculated the frequency of each word to create a token

```python
ctwitter["frequency"] = ctwitter["text"].apply(lambda x: len(str(x).split(" ")))
ctwitter.groupby("frequency").max()
```

Python

	text	label	tokens
frequency			
1	wrong	1	[wrong]
2	zombies subhummmman	1	[zombies, subhummmman]
3	zionists smell bad	1	[zionists, smell, bad]
4	zoo disgusting wanna die	1	[zoo, disgusting, wan, na, die]
5	zyklon b k k e	1	[zyklon, b, k, k, e]
...	...	...	...
187	oh god see city london television frightfully ...	0	[oh, god, see, city, london, television, frigh...
193	oh god see city london television frightfully ...	1	[oh, god, see, city, london, television, frigh...
198	adam even first white people earth lost tribes...	0	[adam, even, first, white, people, earth, lost...
207	deeyah khan firstly british born uk absolutely...	0	[deeyah, khan, firstly, british, born, uk, abs...
211	deeyah khan firstly british born uk absolutely...	1	[deeyah, khan, firstly, british, born, uk, abs...

Create a sentiment of the data

We create a sentiment of the data text and then create a dataframe from this sentiment

```python
blob_emptyline = []
for i in ctwitter["text"]:
    blob = TextBlob(i).sentiment
    blob_emptyline.append(blob)
```

Pytho

We create a dataframe from the sentiment of the text.

```python
blob=pd.DataFrame(blob_emptyline)
```

Pytho

```python
twitter_with_sentiment = pd.concat([ctwitter.reset_index(drop=True), blob], axis=1)
```

Pytho

Add the sentiment column

We add the sentiment column. This column will check the polarity of the text. If the polarity is above 0, then we have a positively perceived text, else it is a negatively perceived text!

```
twitter_with_sentiment["Sentiment"]
=  np.where(twitter_with_sentiment["polarity"] >= 0 , "Positive", "Negative")
```

Plot the sentiment

We plot the sentiment of the text in many ways to better visualize and understand what it is.

Historgram

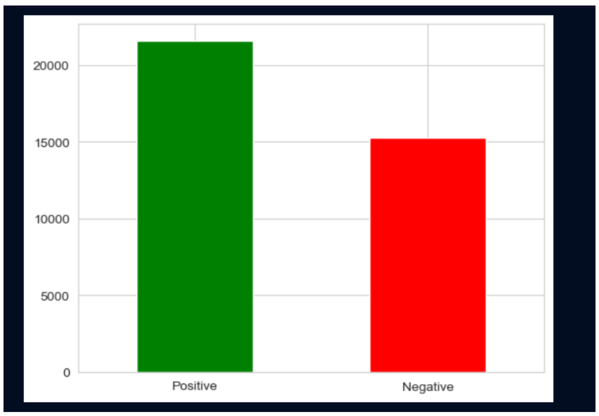


Table (sentiment)

```
twitter_with_sentiment.groupby("Sentiment").count()
```

	text	label	tokens	frequency	polarity	subjectivity
Sentiment						
Negative	15292	15292	15292	15292	15292	15292
Positive	21590	21590	21590	21590	21590	21590

Table (polarity)

```python
twitter_with_sentiment.groupby("polarity").max()
```

Python

	text	label	tokens	frequency	subjectivity	Sentiment
polarity						
-1.000000	zoomers suck worst generation ever	1	[zoomers, suck, worst, generation, ever]	23	1.000	Negative
-0.950000	hated vanilla sky boring movie	1	[hated, vanilla, sky, boring, movie]	5	0.850	Negative
-0.900000	zionists repellent	1	[zionists, repellent]	13	1.000	Negative
-0.900000	women evil follow trend matter stupid	1	[women, evil, follow, trend, matter, stupid]	30	1.000	Negative
-0.875000	want encourage brutal murdering minorities bro...	1	[want, encourage, brutal, murdering, minoritie...	9	1.000	Negative
...	...	...	...	...	...	...
0.833333	man style clothing rubbish especially gay	0	[man, style, clothing, rubbish, especially, gay]	6	1.000	Positive
0.850000	women might definition beautiful	1	[women, might, definition, beautiful]	16	1.000	Positive
0.875000	essence lgbtqs wonderful group successful peop...	1	[essence, lgbtqs, wonderful, group, successful...	10	0.975	Positive
0.900000	women incredibly mendacious	1	[women, incredibly, mendacious]	18	1.000	Positive
1.000000	youre refugee fled turkey spend time shitposti...	1	[youre, refugee, fled, turkey, spend, time, sh...	20	1.000	Positive

Combine datasets and add into new

We combine the datasets and add the features into the new data set (tweet).

```python
twitter_polarity = twitter_with_sentiment['polarity']
twitter_text = twitter_with_sentiment['text']
twitter_tokens = twitter_with_sentiment['tokens']
twitter_frequency = twitter_with_sentiment['frequency']
twitter_subjectivity = twitter_with_sentiment['subjectivity']
twitter_sentiment = twitter_with_sentiment['Sentiment']
twitter_label = twitter_with_sentiment['label']


d_data1={'Polarity':twitter_polarity,'Text':twitter_text,'Tokens':twitter_tokens,'Frequency
tweet= pd.DataFrame(data=d_data1)
tweet
```

Python

	Polarity	Text	Tokens	Frequency	Subjectivity	Sentiment	Label
0	0.118182	want live near koshies	[want, live, near, koshies]	4	0.450000	Positive	0
1	0.500000	love arabs hardworking love cleaning toilets	[love, arabs, hardworking, love, cleaning, toi...	6	0.600000	Positive	0
2	0.000000	oh women hope like hell aborted babies	[oh, women, hope, like, hell, aborted, babies]	7	0.000000	Positive	0
3	0.000000	indian people get rights	[indian, people, get, rights]	4	0.000000	Positive	0
4	-0.500000	urgh blacks tiresome	[urgh, blacks, tiresome]	3	1.000000	Negative	0
...	...	...	...	...	...	...	...
36877	0.459091	im looking new best friends love go iceskating...	[im, looking, new, best, friends, love, go, ic...	9	0.388636	Positive	1
36878	0.000000	cigars stink	[cigars, stink]	2	0.000000	Positive	1
36879	0.400000	happy white teeth	[happy, white, teeth]	3	0.500000	Positive	1
36880	0.500000	many closest colleagues muslim	[many, closest, colleagues, muslim]	4	0.500000	Positive	1
36881	0.022222	really shows shallow greedy proriented psychol...	[really, shows, shallow, greedy, proriented, p...	8	0.300000	Positive	1

More Visualizations

Form a word cloud from the most common words in the dataset

We extract the hate speech from the data entries first.

```
hate = ctwitter[ctwitter['label'] == 0]
hate
```

Python

	text	label	tokens	frequency
0	want live near koshies	0	[want, live, near, koshies]	4
1	love arabs hardworking love cleaning toilets	0	[love, arabs, hardworking, love, cleaning, toi...	6
2	oh women hope like hell aborted babies	0	[oh, women, hope, like, hell, aborted, babies]	7
3	indian people get rights	0	[indian, people, get, rights]	4
4	urgh blacks tiresome	0	[urgh, blacks, tiresome]	3
...	...	...	...	...
18436	party launching campaign entitled mosques futu...	0	[party, launching, campaign, entitled, mosques...	23
18437	care perceive continue call n g g e r	0	[care, perceive, continue, call, n, g, g, e, r]	9
18438	trashy dont wanna near dont really like chinks	0	[trashy, dont, wan, na, near, dont, really, li...	8
18439	women good nothing	0	[women, good, nothing]	3
18440	jew doesnt mean less human means priorities di...	0	[jew, doesnt, mean, less, human, means, priori...	10

We create the word cloud

```python
vocab = [ ]
label = []
for _,row in ctwitter.iterrows():
    a = row['text'].split()
    if(row['label'] == '0'):
        label+=[0 for i in range(len(a))]
    else:
        label+=[1 for i in range(len(a))]
    vocab+=a
```

Python

```python
vocab_model_relation = pd.DataFrame({'Words': vocab ,'Label': label })
vocab_model_relation.head()
```

Python

Then we count the most prominent words in the word cloud

```python
words = vocab_model_relation['Words'].value_counts().index
words
```

Python

```
Index(['people', 'fucking', 'black', 'like', 'women', 'dont', 'love', 'think',
       'would', 'im',
       ...
       'countrymen', 'niggish', 'eliiiimiiiinateeed', 'traaaniiieeesss',
       'nongoys', 'obtuse', 'foids', 'rakhine', 'edging', 'cigars'],
      dtype='object', length=20545)
```

```python
word_counts = {word: 0 for word in words}

for string in ctwitter['text']:
    for word in string.split():
        if word in word_counts:
            word_counts[word] += 1

# Sort the dictionary by count in descending order
word_counts = dict(sorted(word_counts.items(), key=lambda item: item[1], reverse=True))

count = 0
for word, count_val in word_counts.items():
    print(f'{word}: {count_val}')
    count += 1
    if count == 10:
        break
```

Python

```
people: 7275
fucking: 3612
black: 3529
like: 3411
women: 3376
dont: 3313
love: 2488
think: 2426
would: 1900
im: 1758
```

Finally, we visualize the word cloud of the common words

Common Words

All the dataset

Here we analyze the data and results that was already applied on the model

Clean the dataset

Check for null

We of course checked for null values again, and surprisingly we find null values in the predication answers that are right and wrong (model_wrong values). So, we drop these rows.

```
print(twitter1.isnull().sum())
```

```
Unnamed: 0          0
id                  0
text                0
label               0
type                0
model_wrong     14526
db.model_preds  14526
status              0
round               0
split               0
annotator           0
dtype: int64
```

```
twitter1.dropna(inplace=True)
```

Check for duplicates

We also check for duplicates and unlike above we find none, suggesting that the same text input gave us different predications in the model used or that or that they were removed with the null values.

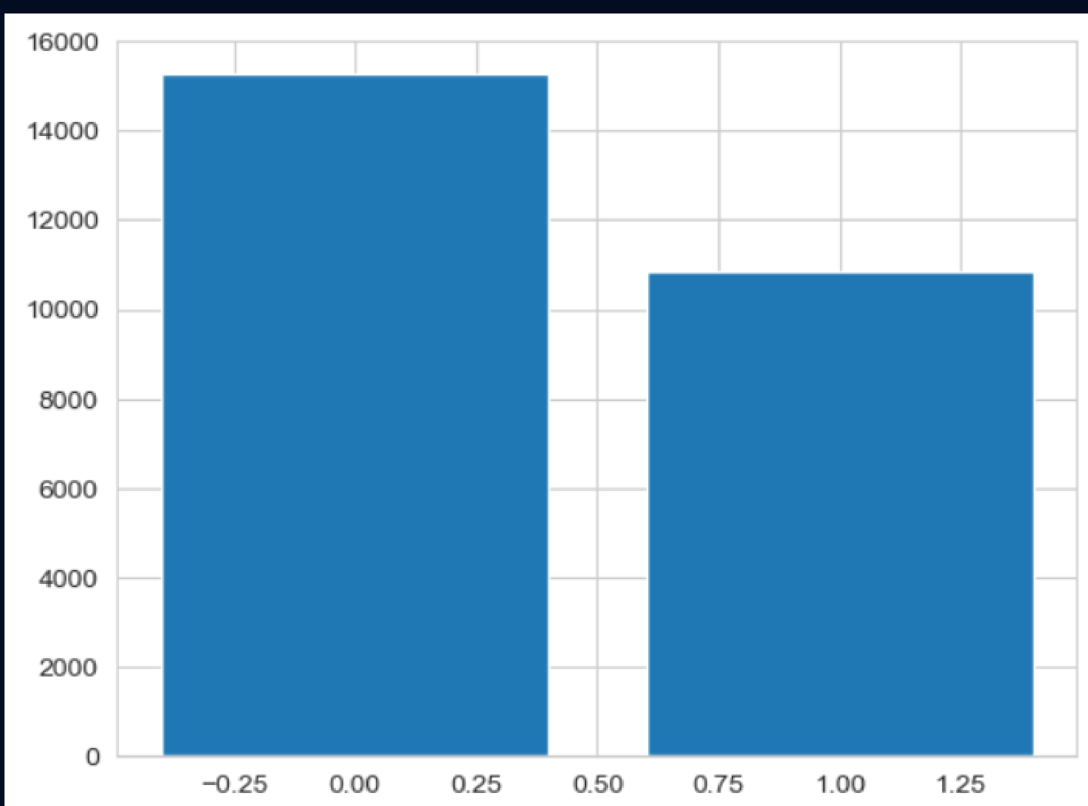
```
duplicates = twitter1[twitter1.duplicated()]
if len(duplicates) > 0:
    print('Duplicate rows:')
    print(duplicates)
else:
    print('No duplicate rows found.')
```

```
No duplicate rows found.
```

Check the balance

We recheck the balance and find it more under-sampled then in the above data.

```
0    15253
1    10844
Name: label, dtype: int64
```



Visualize the dataset

Bar chart and histogram

We visualize the data as above using a bar chart and histogram and get the same results.

Check the different type of type

Afterwards, we also visualized some of the data, such as the type which shows us the different type of hate speech the data contains.

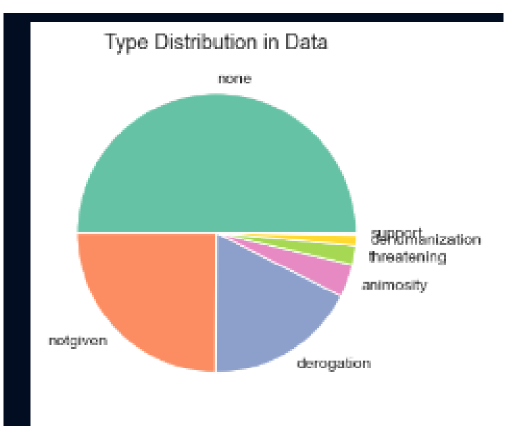
```
twitter.groupby('type').count()['id']

type
animosity            825
dehumanization       267
derogation          3827
none               10844
notgiven            5398
support               49
threatening          478
Name: id, dtype: int64
```

Graphs to visualize

Here we just visualized different types of data in the dataset.

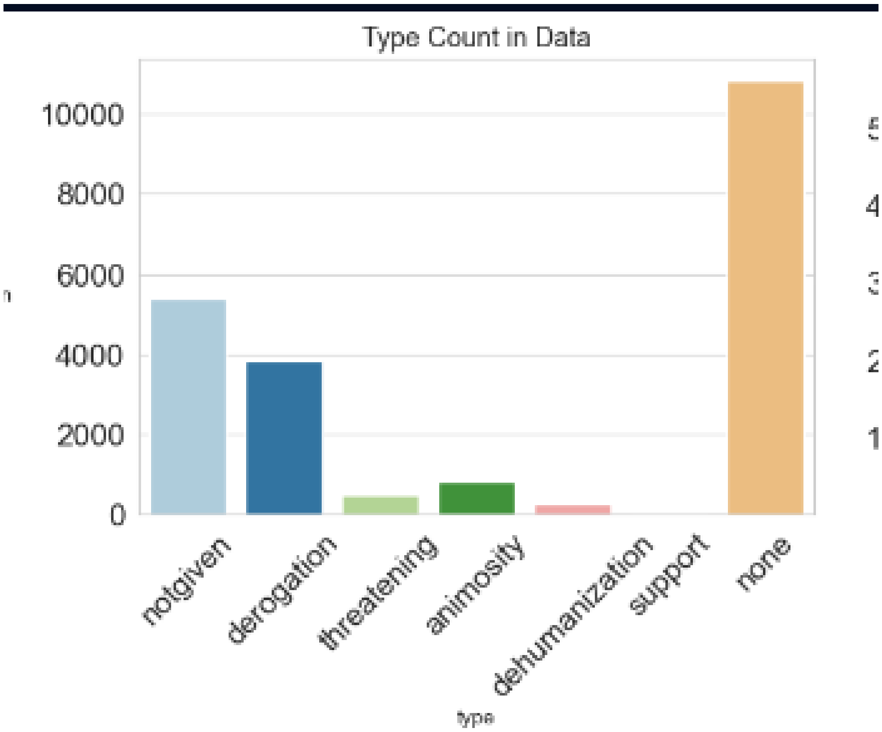
Type of distribution in Data:

We get this from the different types of hate speech which we found above.

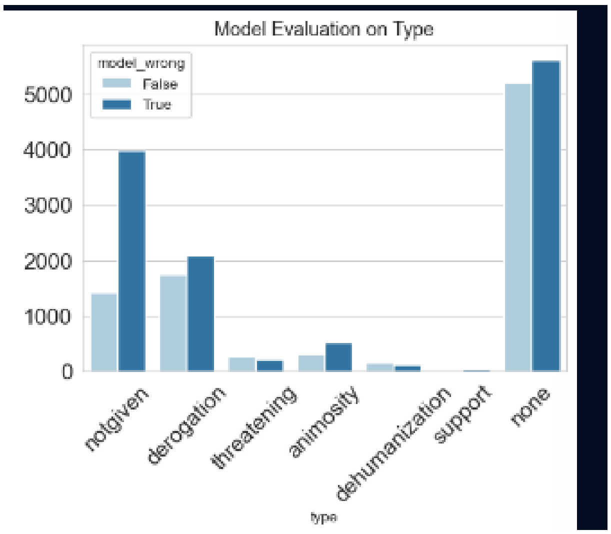


It should be noted that this graph also represents why we did not use the type of hate speech as an indicator for the model, since almost half the rows for it are classified as none.

Type of count in data



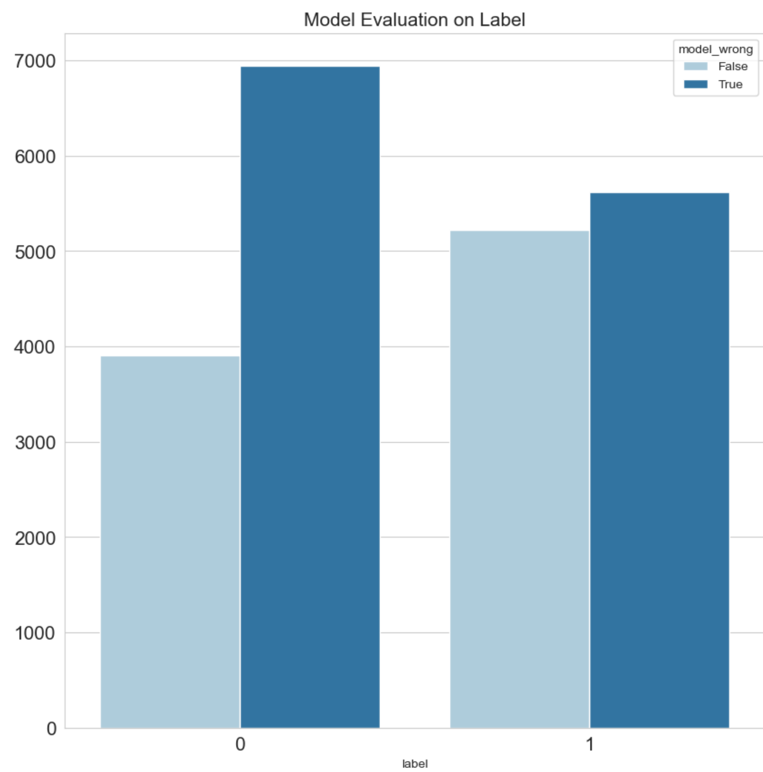
Model evaluation based on type



What the premade model got wrong and right

Finally, we look at what the premade model got right and wrong for its predications, we know this from the model_wrong column in the dataset.

A right predication is classified as 0 and a wrong prediction is classified as 1.




```
True     12565
False     9123
Name: model_wrong, dtype: int64
```

Cleaning and visualizing the dataset using NLP

To show my data visualization, we cleaned and visualized the data of the given model.

First we did the following above steps on the text:

Convert to lower case

Remove numeric values

Remove all non-english words

Remove stop words

Lemmatize the words

And got this as our final result of the text:

```
0                                          chinks worse snakes
1        africandescended people infamous least accompl...
2        would obtain great satisfaction seeing refugee...
3        getting hand arguments today already merely ex...
4        guy ive dated dwarves wouldnt date midget dwar...
                               ...
21683                   understand many problem trans people
21684                                 zoo disgusting wanna die
21685                                     black women panthers
21686                                         tv fucking poor
21687                        many closest colleagues muslim
Name: text, Length: 21688, dtype: object
```

Then we extracted the hate speech as above and got the text from the hate speech:

	Words	Model Wrong	Label
0	chinks	1	1
1	worse	1	1
2	snakes	1	1
3	africandescended	0	1
4	people	0	1

From this create a dataframe with the words, model wrong, and label and counted the most prominent words of the right predictions and the wrong predictions each alone:

Wrong predictions

```python
words = vocab_model_relation[vocab_model_relation['Model Wrong'] == 1]['Words'].value_coun
words
```

Python

```
Index(['people', 'black', 'fucking', 'women', 'like', 'dont', 'love', 'would',
       'think', 'hate',
       ...
       'underlining', 'werner', 'braun', 'escaped', 'nasa', 'peoplehere',
       'perks', 'extremism', 'favorites', 'luxury'],
      dtype='object', length=12885)
```

Most common

```
['bloodline',
 'troubling',
 'schizophrenia',
 'authentically',
 'presses',
 'separated',
 'cardboard',
 'ghanaians',
 'scummy',
 'elevating']
```

Right predications

```python
words2 = vocab_model_relation[vocab_model_relation['Model Wrong'] == 0]['Words'].value_cou
words2
```

Python

```
Index(['people', 'fucking', 'black', 'women', 'dont', 'like', 'think', 'hate',
       'white', 'love',
       ...
       'saved', 'bw', 'bookstore', 'nostrings', 'spits', 'carriers',
       'worshiping', 'trumptards', 'misanthropes', 'cockheads'],
      dtype='object', length=10328)
```

Most common

```
['obeys',
 'scholar',
 'caperefugee',
 'ambidextrous',
 'brioches',
 'incstuous',
 'pesky',
 'hoodwinking',
 'mudslime',
 'aerosol']
```

Afterwards we got the common words from both of these and the count of each word

```
common_words = list(set(words)&set(words2))
common_words[:10]
```

```
['thin',
 'pumping',
 'insects',
 'images',
 'smuggling',
 'leaking',
 'prince',
 'interests',
 'transphobic',
 'collection']
```

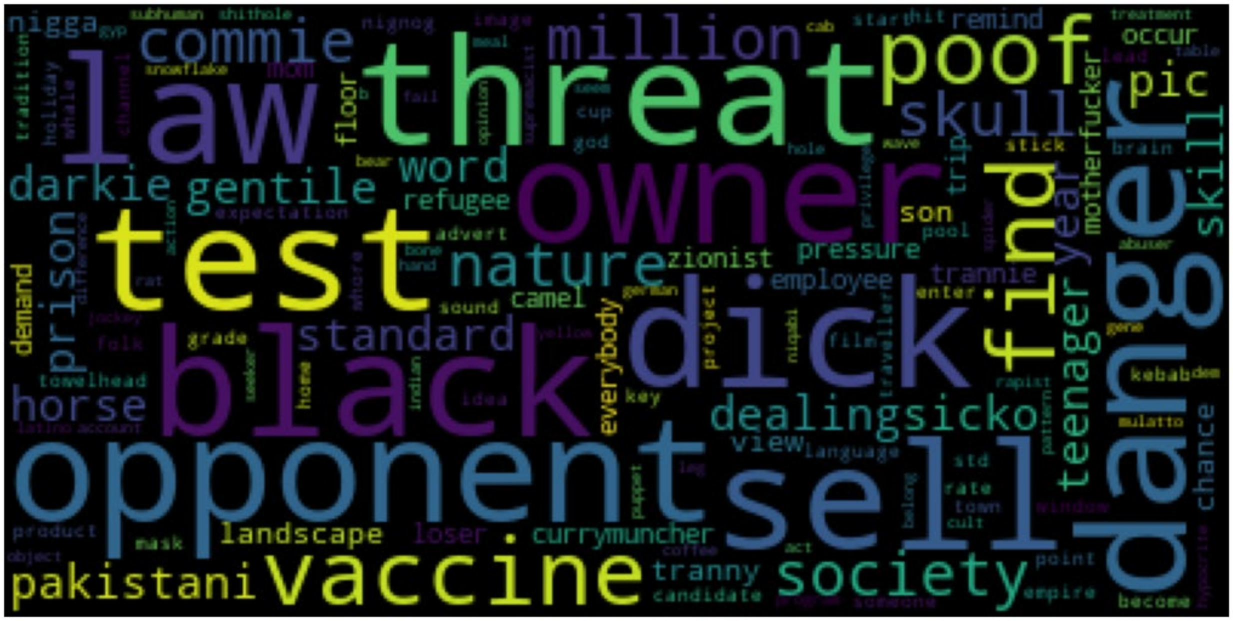
```
people: 3707
fucking: 2177
black: 2167
women: 1898
dont: 1707
like: 1644
love: 1129
think: 1124
hate: 1109
white: 1071
```

Form a word cloud

From this we formed a word could for each of them:

Common words

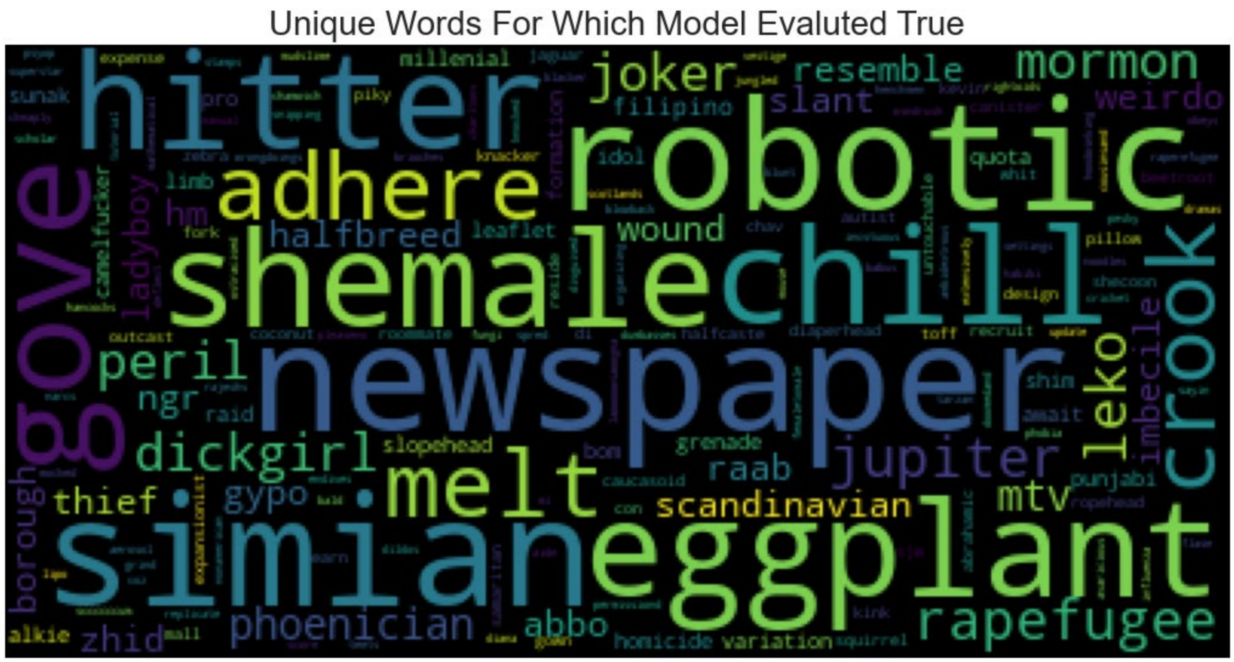
Common Words



Wrong model words

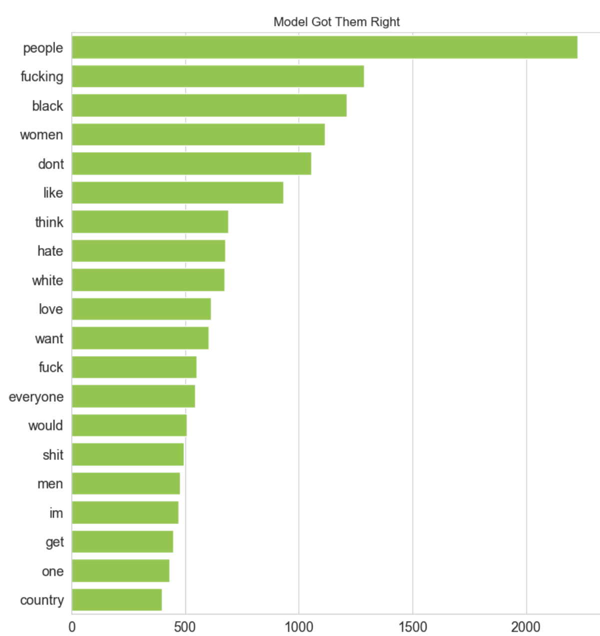
Unique Words For Which Predictions Were Wrong

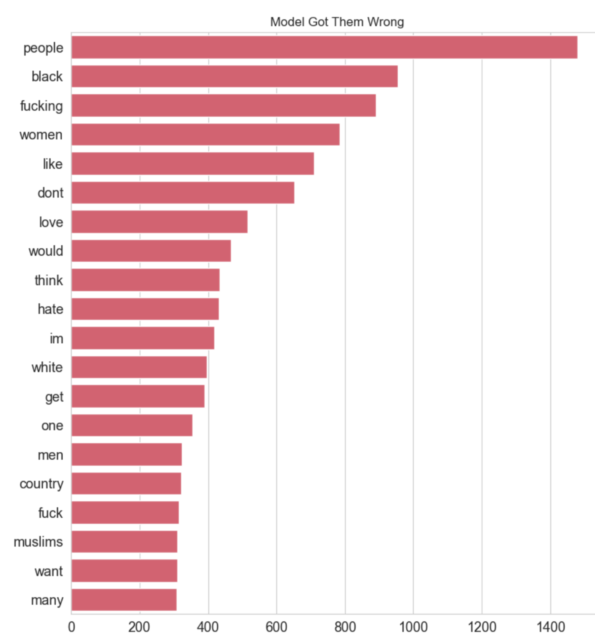
Right model words



Form a graph from the words

We also formed graphs to represent the right and wrong predictions of the words





Get the count of right and wrong models

Finally, we got the count of the right and wrong word for the model used so we can compare them with our model later on.

```
class_counts2 = twitter['model_wrong'].value_counts()
right_count5=class_counts2[0];
wrong_count5=class_counts2[1];
print(class_counts2)
```

```
True     12565
False     9123
Name: model_wrong, dtype: Int64
```

Splitting the Data

Here we get the data ready to be split and split it into 2 sets, a training, and a testing set. However, instead of just splitting the data for polarity only, we can use another model which is a word count vector. We expect the word count to perform better since polarity might not be able to convey a person's attitude.

Split for word count vector

A word count vector is a way to represent text data as numerical vectors. The idea of a word count vector is to count the number of times that each word appears in the text and use those counts as the element of the vector.

This is where all the steps above and below come into hand. The steps of the word count vector are:

1. To remove all and any unwanted characters, converting all text to lowercase, removing stop words, and tokenizing the text into individual words.
2. Then we create a vocabulary of all the unique words in the corpus.
3. For each text in the corpus, we create a vector of length equal to the size of the vocabulary.
4. Once we create a word count vector for each text in the corpus, we can use these vectors in our models. Where these vectors can be normalized to have unit length, which can improve the performance of some models.

```python
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42)
```

Python

```
text_list=[]
for text in tweet.Text:
    description=re.sub("[^a-zA-Z]"," ",text)
    description=description.lower()
    description=nltk.word_tokenize(text)
    description=[lemmatizer.lemmatize(word) for word in text]
    description="".join(description)
    text_list.append(description)
```

```
from sklearn.feature_extraction.text import CountVectorizer
count_vectorizer=CountVectorizer(max_features=25000,stop_words="english")
count_vector=count_vectorizer.fit_transform(text_list).toarray()
```

```
x=count_vector
y=tweet["Label"].values
```

The word count is now ready.

Split for polarity

A text's polarity serves as a gauge for its emotional content or feeling. It is frequently expressed as a numerical number between -1 and 1, where -1 denotes a strongly negative feeling, 0 a neutral feeling, and 1 a strongly happy feeling.

```
d = {
     'Frequency':tweet['Frequency'],
     'Polarity':tweet['Polarity'],
      'Subjectivity':tweet['Subjectivity'],
      'Label':tweet['Label']
}
df = pd.DataFrame(data=d)
```

Pytho

```
from sklearn.model_selection import train_test_split
X=df.iloc[:,:-1]
Y=df.iloc[:,-1]
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size=0.2,random_state=42)
```

Pytho

```
from sklearn.preprocessing import StandardScaler
scalar = StandardScaler()
X_train['Frequency'] = scalar.fit_transform(X_train['Frequency'].values.reshape(-1,1))
X_test['Frequency'] = scalar.transform(X_test['Frequency'].values.reshape(-1,1))
```

Pytho

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42)
```

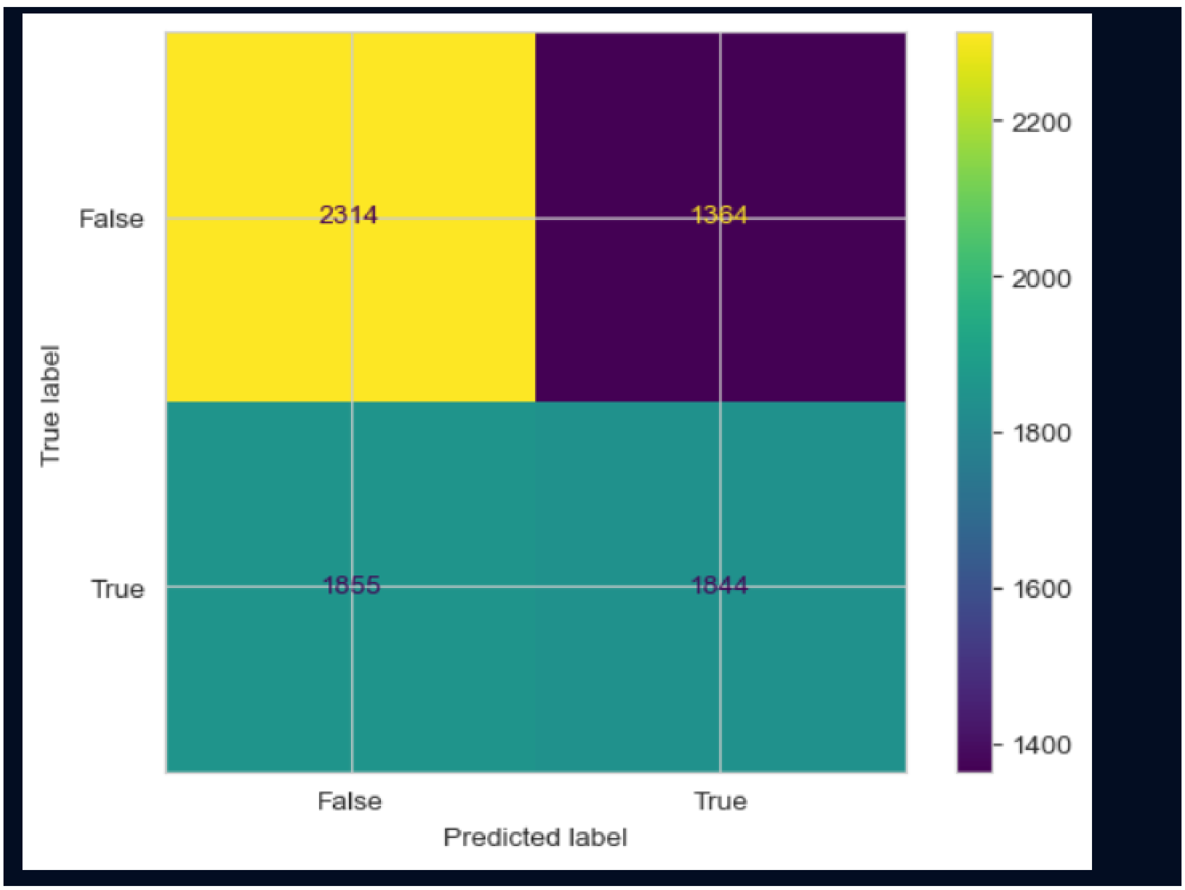
Pytho

Choosing, Training, and Evaluating a Model

Models

Logistic Regression

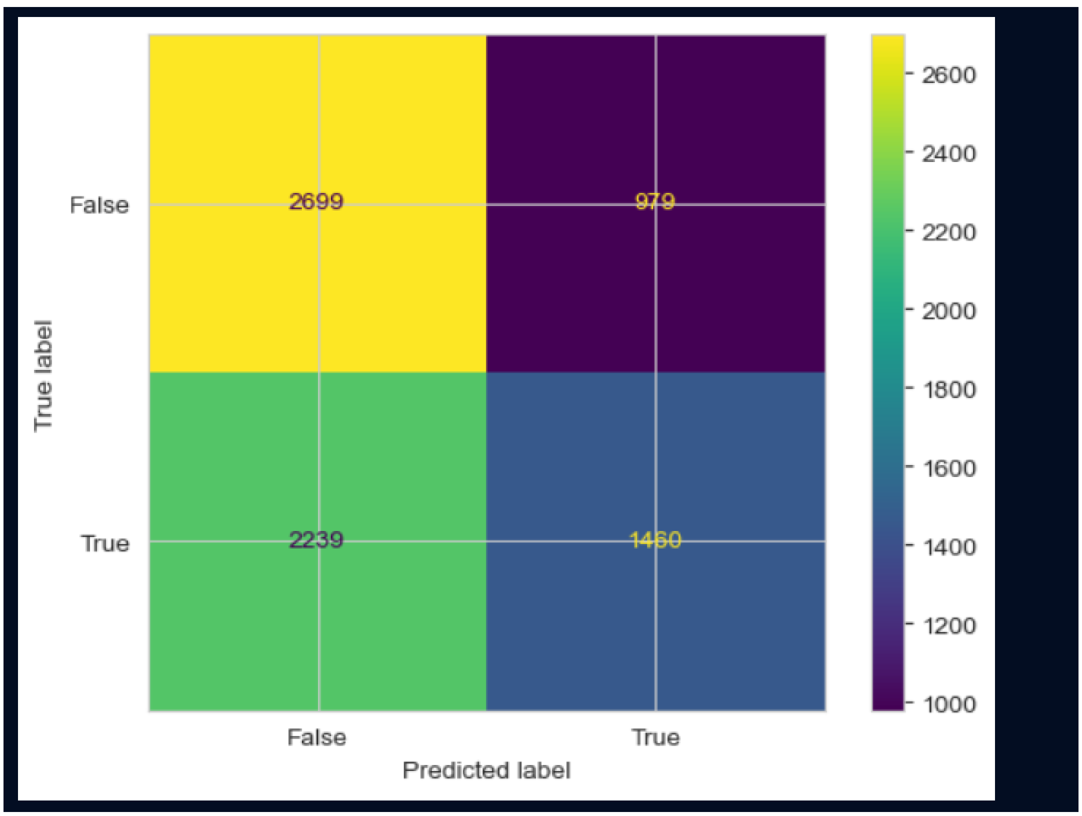
Logistic regression is a classification algorithm that is commonly used in machine learning to predict binary outcomes. In the context of sentiment analysis, logistic regression can be used to predict the polarity of a piece of text meaning whether it has a positive or negative sentiment.




```
Recall is: 0.4985131116517978
Precision is: 0.5748129675810474
Accuracy is: 0.5636437576250508
F1 Score: 0.5339510641378312
```

Linear SVM

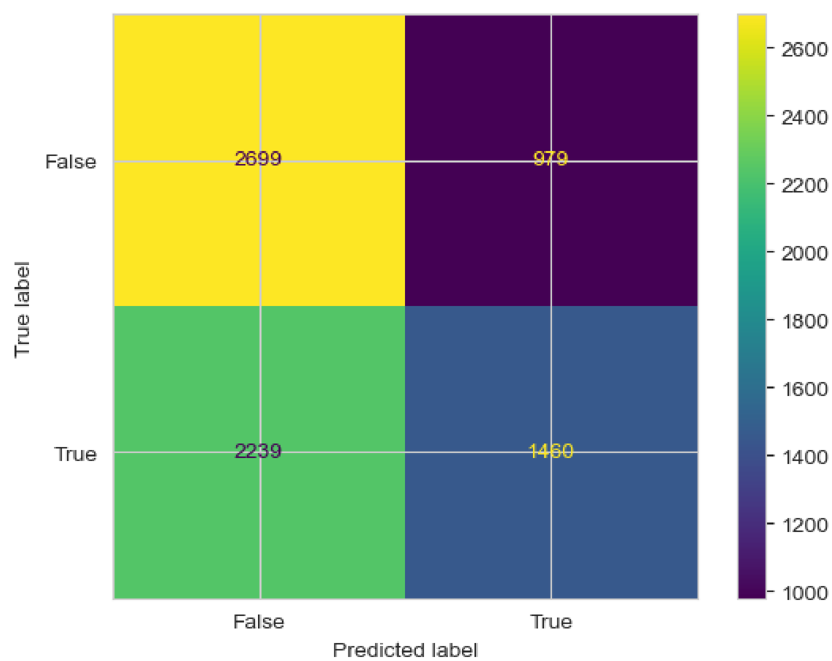
Another classification technique that may be used for sentiment analysis is linear SVM (Support Vector Machine). It is a binary classification technique, similar to logistic regression, that can forecast whether a piece of text will be positive or negative. The way that Linear SVM operates in the context of sentiment analysis is by identifying the hyperplane that most effectively distinguishes between positive and negative samples in the training data. The two classes in the feature space are divided by the hyperplane, a linear boundary. Each dimension in the feature space (which has several dimensions) correlates to a feature, such as a word in the lexicon.




```
Recall is: 0.3947012706136794
Precision is: 0.5986059860598606
Accuracy is: 0.5637793140843161
F1 Score: 0.4757249918540241
```

Native Bayes

Another classification technique that may be applied to sentiment analysis is naive Bayes. It is a probabilistic algorithm based on the Bayes theorem, which states that the probability of a hypothesis (in this case, the polarity of a piece of text) given some observed evidence (the words in the text) is proportional to the probability of the evidence given the hypothesis, multiplied by the prior probability of the hypothesis. Naive Bayes, which is used in sentiment analysis, calculates the likelihood of each word in the lexicon given each class (positive or negative). This is done by counting the frequency of each word in each class and utilizing that information to predict the likelihood that word will appear in the class, using the training data.

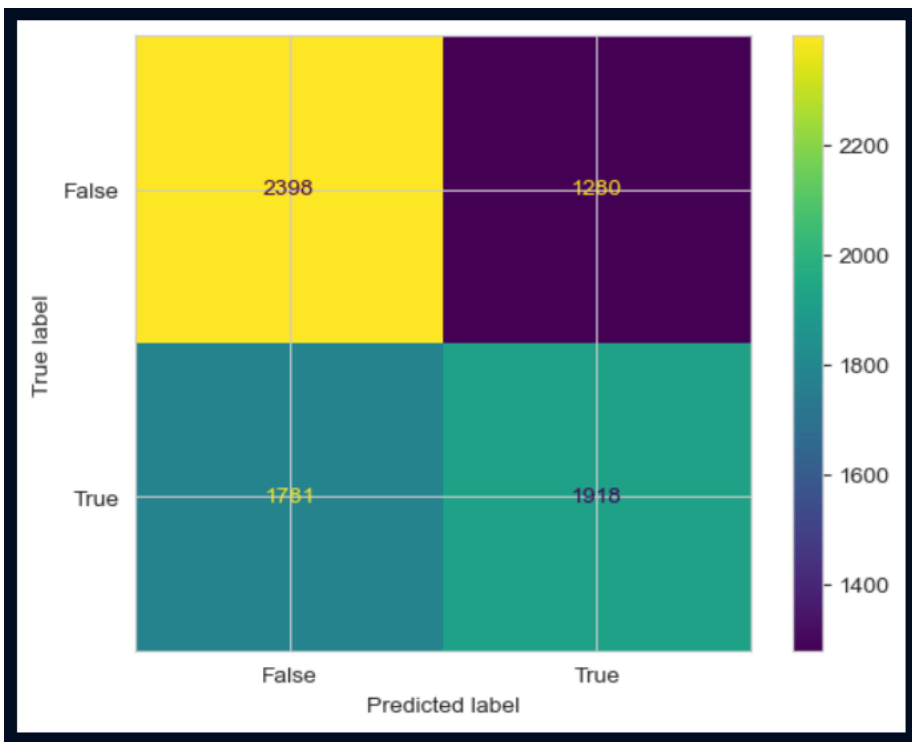



```
Recall is: 0.48202216815355503
Precision is: 0.5872859025032938
Accuracy is: 0.570421580588315
F1 Score: 0.5294729027468449
```

K-nearest with K=15

Another classification technique that may be applied to sentiment analysis is K-nearest neighbors (KNN). It is a non-parametric approach that determines the K feature space neighbors that are closest to a given data point, and then assigns the class based on the consensus of those neighbors. KNN represents each piece of text as a vector in the feature space where each dimension corresponds to a feature (for example, a word in the lexicon) in the context of sentiment analysis. The KNN algorithm saves the feature vectors and the polarity labels that go with them throughout training.

It should be noted that we tried multiple different values for K and K=15 was the most efficient.

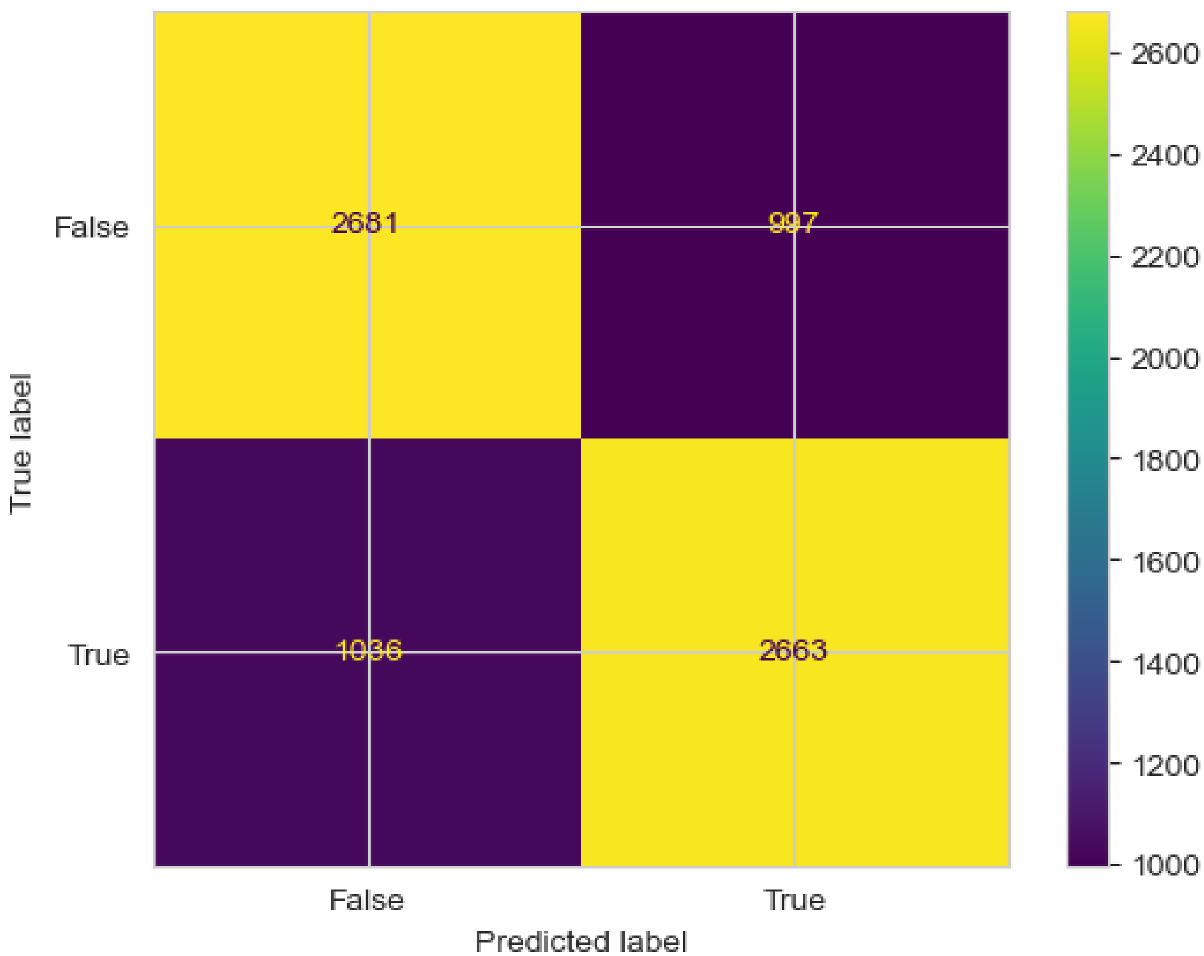



```
Recall is: 0.5185185185185185
Precision is: 0.5997498436522827
Accuracy is: 0.5850616781889657
F1 Score: 0.5561838480498768
```

Word Count Vector Model

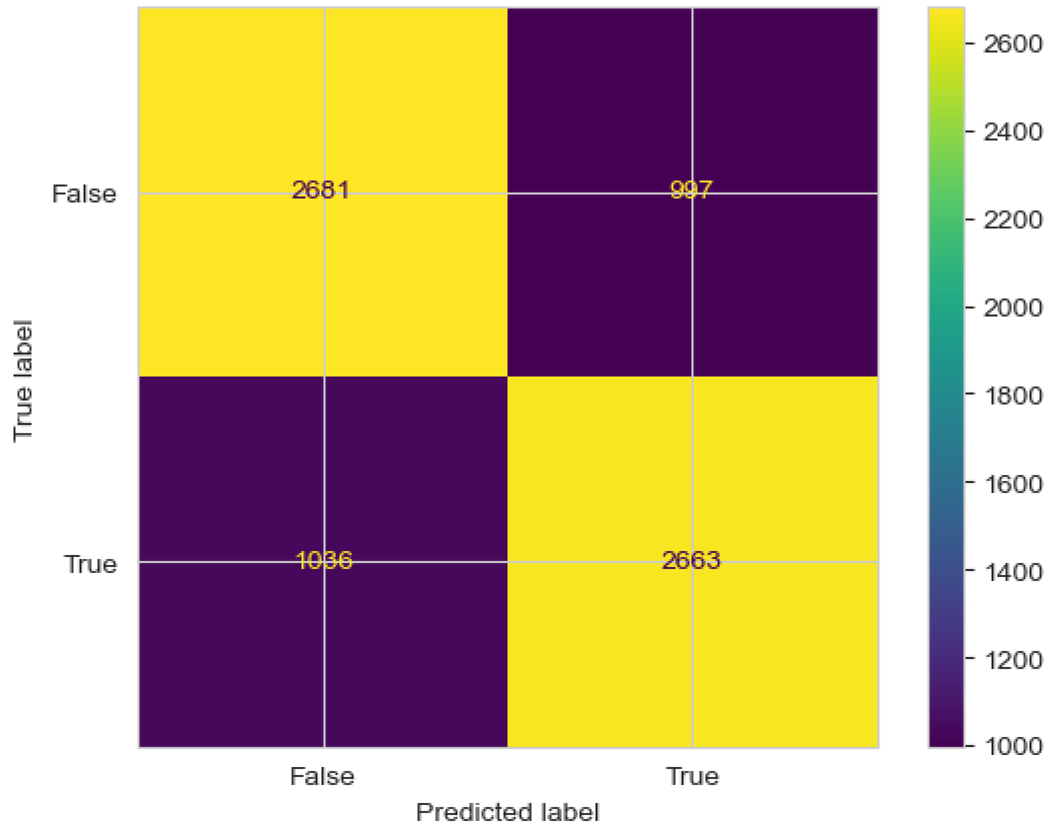
As we expected the word count vector gives us better results

Logistic Regression




```
Recall is: 0.7199243038659097
Precision is: 0.7275956284153006
Accuracy is: 0.7244137183136776
F1 Score: 0.7237396385378448
```

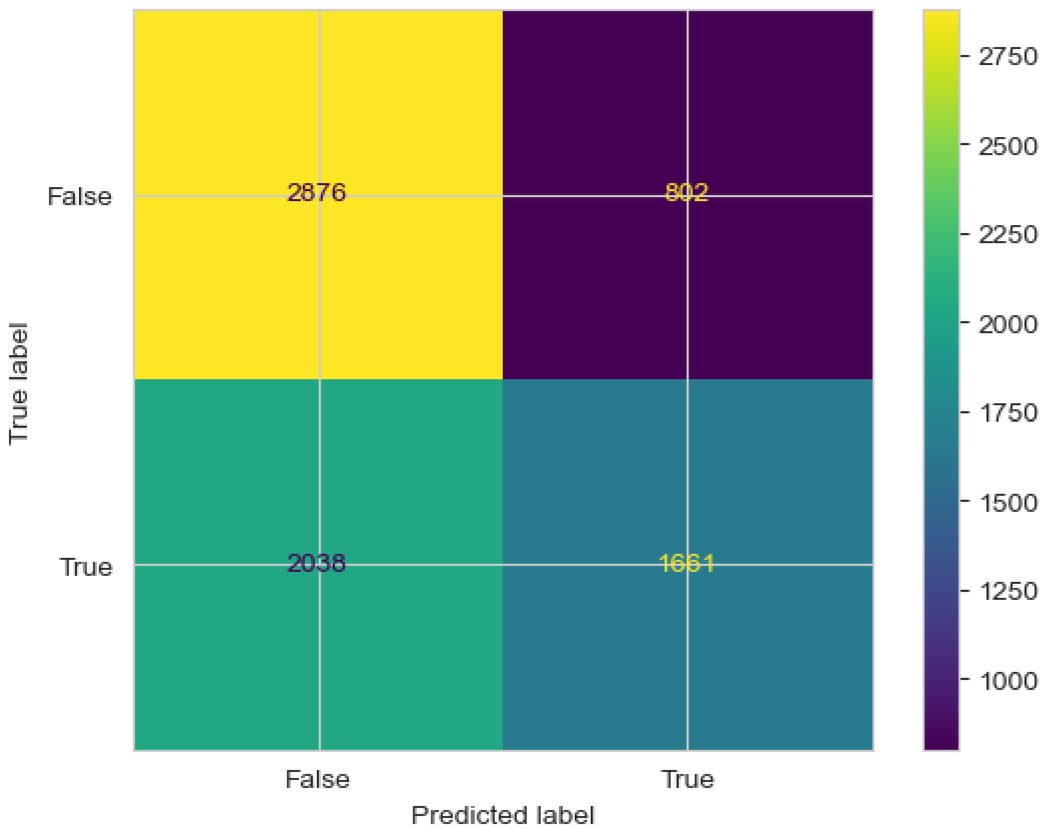
Native Bayes




```
Recall is: 0.11489591781562584
Precision is: 0.4408713692946058
Accuracy is: 0.48312322082147213
F1 Score: 0.18228608192150977
```

K-nearest with K=15

Here we tried multiple different K’s as we did when we used polarity and K=15 was the best




```
Recall is: 0.4490402811570694 5
Precision is: 0.6743808363784003
Accuracy is: 0.6150196556865934
F1 Score: 0.5391106783511846
```

Discussion of Models

As we expected the word count vector gave us the best results with Logistic Regression being the best at an F1-score of 72.373. Using polarity, the f1-score of most of the answers was in the 40’s and 50’s with similar recall, precision, and accuracy. We tested the Logistic regression, Linear SVM, K-nearest neighbor, and Naïve Bayes, where we got almost similar results. Similarly, we ran the same models on the word count vector, however, when we did so we got different results.

When we ran the word count vector, we got Logistic regression with the highest recall, precision, accuracy, and F1-score, while KNN score stayed relatively the same. However, the naïve bayes decreased completely. Ultimately, we were not able to include Linear SVM in the word count vector. Although we believe that it would do the best the model took to much time to run and would ultimately not give us a result.

As for why we choose these specific models for our algorithm. Naive Bayes, logistic regression, linear SVM, and KNN are popular models for predicting hate speech from text because they are simple, efficient, and effective for text classification tasks. This is because of how each model runs.

Naïve Bayes, although was the worst model for our algorithm, is simple and efficient. As well as it works well with small amounts of training data. This model is commonly used for text classification tasks such as sentiment analysis like for our algorithm, where we used it to classify the sentiment analysis on each text, and spam analysis. This nature of the model is why it is particularly effective for binary classification tasks, such as hate speech detection.

Logistic Regression, which was the best model for our algorithm, is a linear model used for binary classification tasks, like for our hate speech detection. It works well with high-dimensional data, such as the text we inputted, and can handle non-linear relationships between features and target variables.

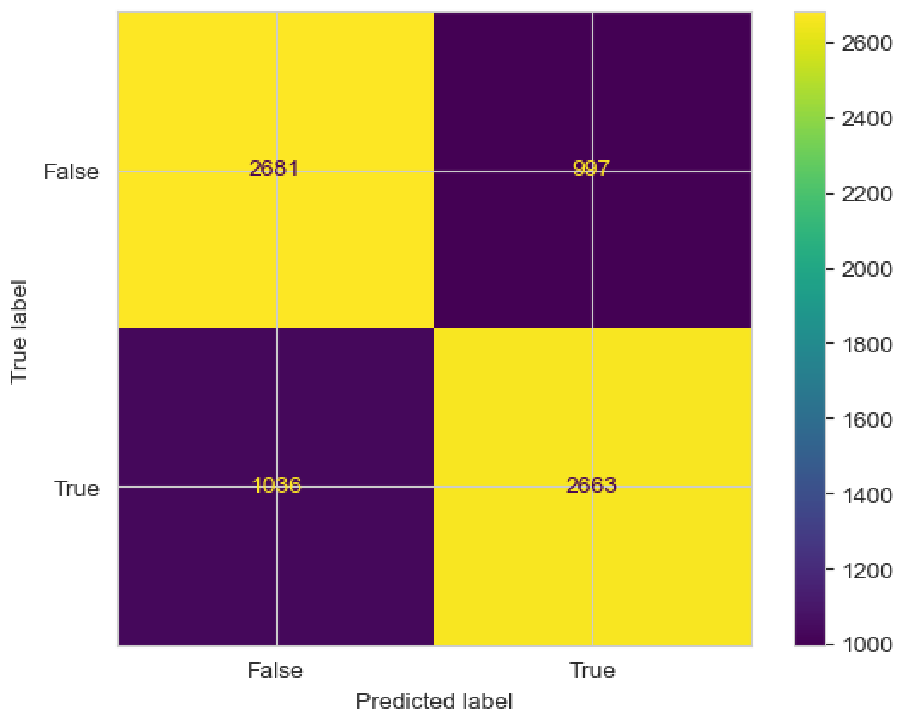
Linear SVM, the model which we sadly were limited from implementing in a word vector model, is commonly used for text classification tasks, such as sentiment analysis and spam detection much like Naïve Bayes. It works well with high-dimensional data and can handle non-linear relationships between the features and the target variables much like Logistic Regression. The collective nature of Linear SVM, and that it was the benefits of both Logistic Regression and Naïve Bayes is why we originally believed it would be the best algorithm under the word count vector model.

Lastly, the K-nearest Neighbor model, which did not change between a lot between polarity and word count vector. It is a non-parametric model that is commonly used for text classification tasks. It is easy to implement and can work well with small amounts of training data. KNN is also particularly effective for binary classification tasks, such as hate speech detection.

Parameter Turing

The best created and evaluated model is Logistic Regression using word count vector. Now we must try to make our model more accurate. We do this by changing the parameter C by multiple numbers. After multiple testing the one with the best F1-score is C=1.2, however even with that the increase in performance is only slight at 0.0005.

```
logres = LogisticRegression(C=1.2)
logres.fit(x_train,y_train)
```




```
Recall is: 0.7193836171938361
Precision is: 0.7284423761292089
Accuracy is: 0.7248203876914735
F1 Score: 0.7238846572361262
```

Comparing my model with the given one

As stated above, we are going to compare the given model with our model.

Word count

Word count of right and wrong from our model:

```
    print(right_count)
    print(wrong_count)
```

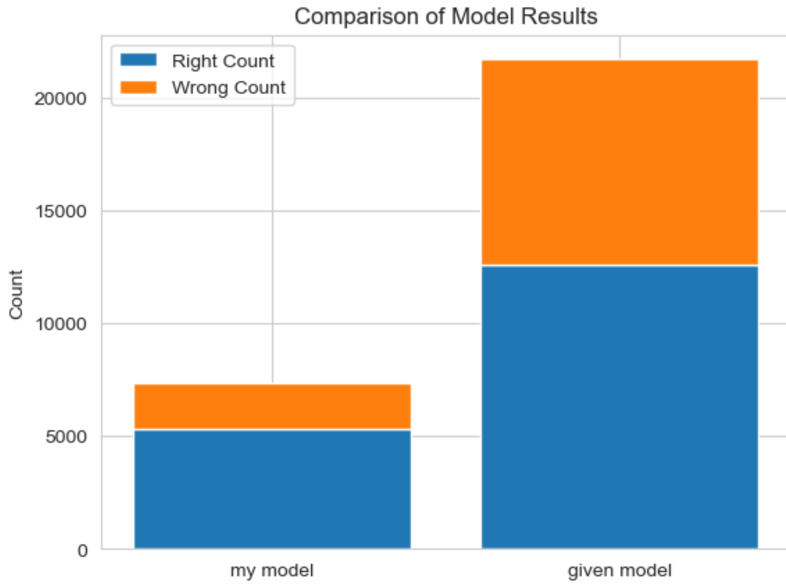
```
5344
2033
```

Word count of right and wrong from the given model:

```
class_counts2 = twitter['model_wrong'].value_counts()
right_count5=class_counts2[0];
wrong_count5=class_counts2[1];
print(class_counts2)
```

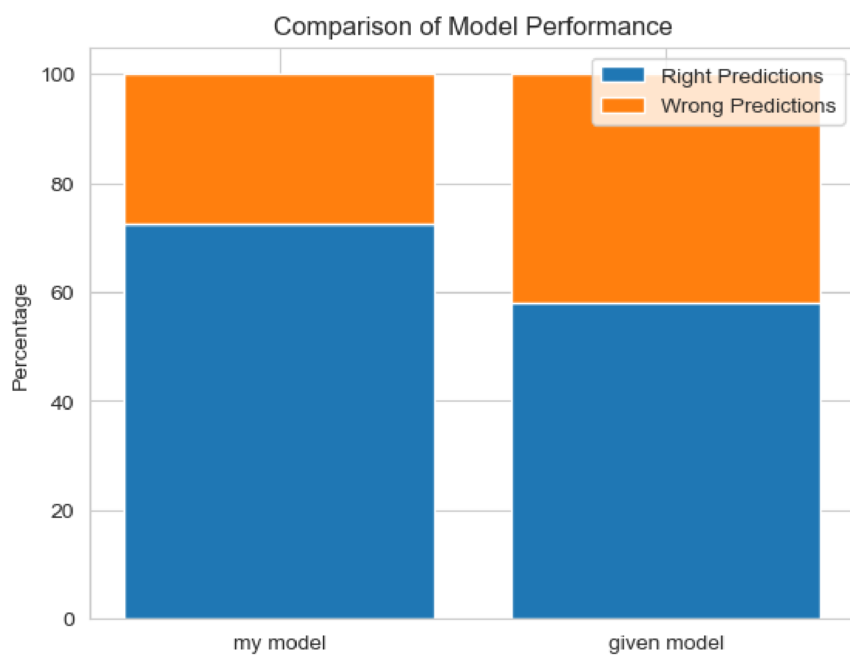
```
True     12565
False     9123
Name: model_wrong, dtype: Int64
```

Model Results



Model Performance

Here we show the difference between the right and wrong predications according to their percentage



Get evaluation of given model

From the count of right and wrong we were able to get the precision, recall, and F1-score of the given model.

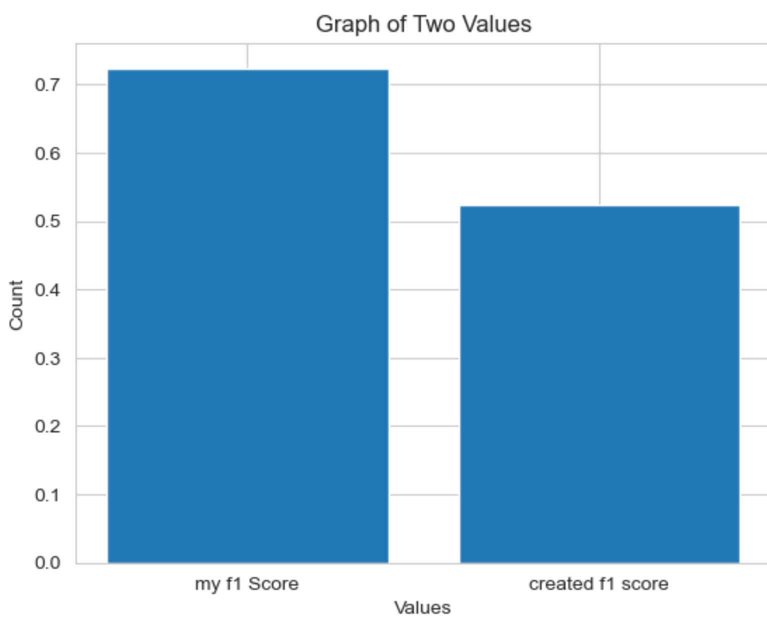
```python
from sklearn.metrics import precision_recall_fscore_support


# Define the actual labels and whether the predictions were right or wrong
y_true = twitter['label']
right_or_wrong = twitter['model_wrong']

# Convert the "right_or_wrong" list into a binary array where 1 represents
y_pred = [1 if x == 'False' else 0 for x in right_or_wrong]

# Calculate the precision, recall, and F1 score
precision_given, recall_given, f1_score_given, support_given = precision_r

# Print the results
print("Precision: {:.2f}".format(precision_given))
print("Recall: {:.2f}".format(recall_given))
print("F1 Score: {:.2f}".format(f1_score_given))
```

Python

```
Precision: 0.57
Recall: 0.48
F1 Score: 0.52
```

Compare F1-scores

From this we can see that although our model did not have the best performance, it was still an improvement on what was previously done.



Making New Predications

Finally, we must test the accuracy of our model by testing the model on unseen data.

New Dataset

Researchers from the Universities of Rochester and California, Los Angeles generated this dataset, which is also known as the "Hate Speech and Offensive Language" dataset. The dataset consists of approximately 24,000 tweets that were gathered between June 2015 and April 2017 and classified as either hate speech, offensive language, or neither by human judges.

The tweets in the dataset address a variety of subjects, including as politics, ethnicity, gender, and sexuality. Because of the nature of the research, the dataset includes content that may be construed as racial, sexist, homophobic, or otherwise objectionable. The fact that the information was gathered through Twitter, which could not be typical of other social media platforms or actual discussions, is only one of the dataset's drawbacks. Furthermore, human judges annotated the dataset, which might have added subjectivity and unpredictability to the categorization process.

The dataset is still a useful tool for academics studying hate speech detection and related fields, though. We, however, are not in need for all the information in this dataset, we only need the text and the label.

Link to new dataset: https://www.kaggle.com/datasets/mrmorj/hate-speech-and-offensive-language-dataset

Fix the Data as needed

The dataset is loaded into twitter3. We check the columns.

```
print(twitter3.columns)
```

Python

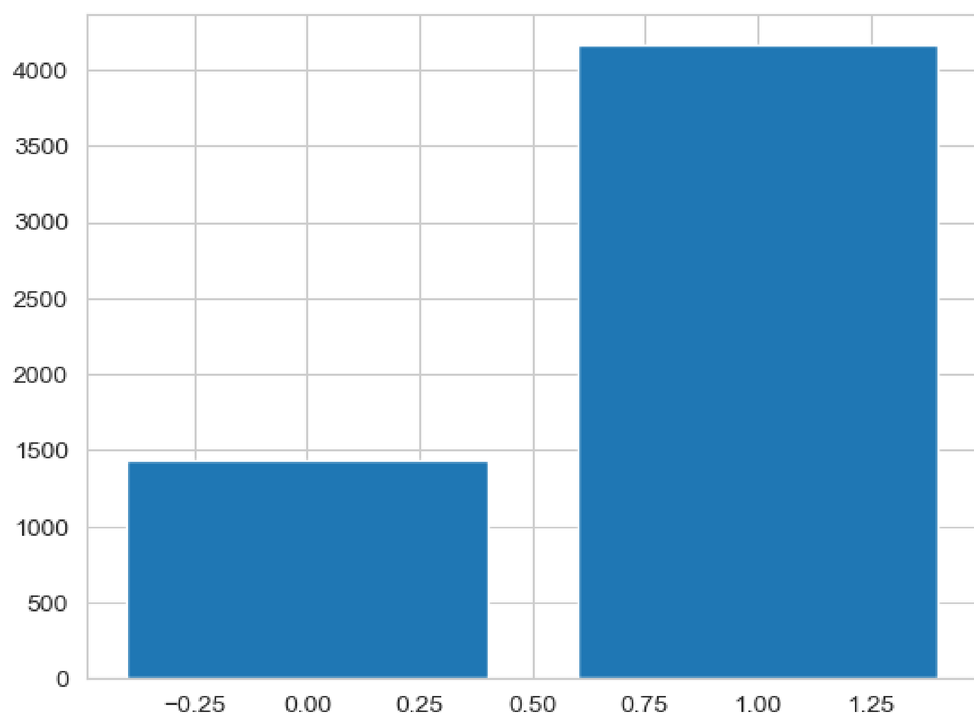
```
Index(['Unnamed: 0', 'count', 'hate_speech', 'offensive_language', 'neither',
       'class', 'tweet'],
      dtype='object')
```

Clean the data as needed

First, we removed the class=1 as it represents offensive language and not hate or not hate language which we don’t not need and could confuse our model.

Then we change the class (not hate) 2 to 1, so that it matched our model.

We then get the label data where 0 represents hate and 1 represents not hate:



We then store the label of the text in ‘classes’ to later compare with the model’s predications. Afterwards, we drop the unwanted data, which is everything but the text.

We of course check for null and duplicate rows and find non.

Moreover, we clean and preprocess the data like we did for the model by:

- Converting the text to lower case.
- Removing numerical and non-alphabetical values.
- Removing all non-English words.
- Lemmatizing the words.

The text will look like this:

```
0        rt mayasolovely woman complain cleaning house ...
40                      momma said pussy cats inside doghouse
63       simplyaddictedtoguys http co woof woof hot sca...
66             allaboutmanfeet http co woof woof hot soles
67       allyhaaaaa lemmie eat oreo amp dishes one oreo...
                               ...
24767    know say early bird gets worm puts gummy worms...
24776                                               nigger
24777    retard hope get type diabetes die sugar rush f...
24779      gone broke wrong heart baby drove redneck crazy
24782    ruffled ntac eileen dahlia beautiful color com...
Name: tweet, Length: 5593, dtype: object
```

Predict new results

Finally, we get to predict new results.

We give the vector the new text and use our model to make predictions.

```
new_X = count_vectorizer.transform(twitter3['text']).toarray()
new_y_pred = logres.predict(new_X)
print("Predicted labels:", new_y_pred)
```

```
Predicted labels: [0 1 1 ... 0 1 0]
```

Compare the Results and Get the Accuracy

Finally, we compare our model’s predications with the actual labels and get the accuracy our model.

```
from sklearn.metrics import accuracy_score

# Calculate the accuracy of the predicted labels
accuracy = accuracy_score(classes, new_y_pred)

# Print the accuracy score
print("Accuracy:", accuracy)
```

```
Accuracy: 0.6670838548185232
```

Conclusion

In this project, we applied various machine learning techniques to classify tweets as either hate speech or not. A dataset of twitter tweets was preprocessed and used to train multiple models such as Naive Bayes, Linear SVM, Logistic Regression, and KNN, using both polarity and a word count vector. The models were evaluated based on their accuracy, precision, recall, and F1-score. The analysis also involved examining the vocabulary of different tweets and comparing the frequency of occurrence of different words in hate and not hate speech articles. Furthermore, we compared the model we created with an already existing model. Finally, we gave our model new data to check its accuracy.