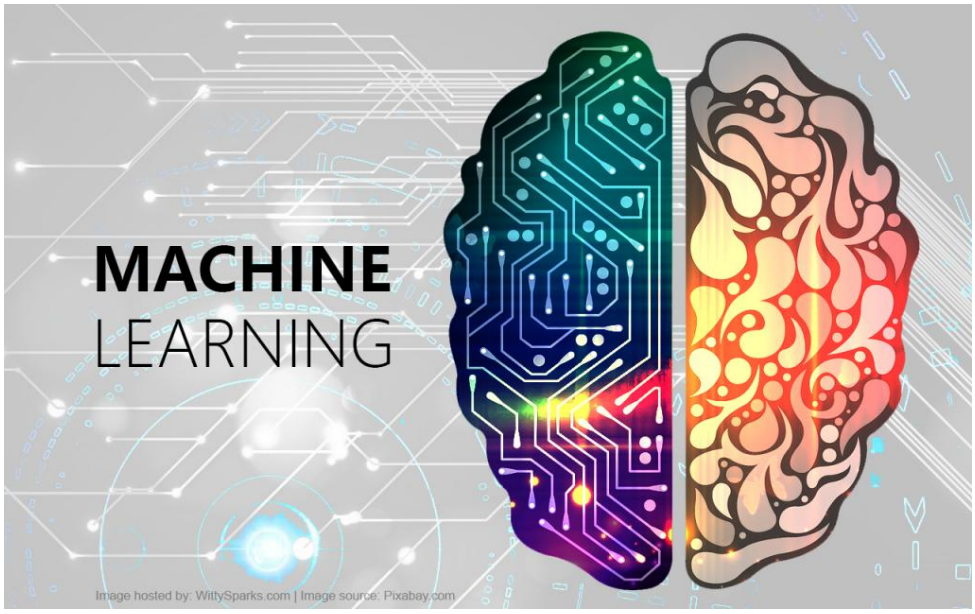


PROJECT REPORT ON
MACHINE LEARNING METHODS FOR HATE SPEECH RECOGNITION IN TWITTER TEXT DATA

Computer Science And Enginearing

Subhasis Sahoo(21BCSL07)
Bishal Mohanty(20BCSB98)
May 2023

PROJECT REPORT ON
MACHINE LEARNING METHODS FOR HATE SPEECH RECOGNITION IN TWITTER TEXT DATA

Computer Science And Enginearing

Subhasis Sahoo(21BCSL07)
Bishal Mohanty(20BCSB98)
May 2023

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

() Principal Adviser

Approved for the Stanford University Committee on Graduate Studies

---

Preface

Welcome to ”Comparative analysis of different Machine Learning methods for Hate Speech Recognition in Twitter text data”.

This thesis is intended to provide a comprehensive overview of the application of machine learning techniques in the detection of hate speech. Hate speech and offensive language create lot of emotional pain to certain people and thus its' detection and subsequent actions would make Twitter a safer platform.

Machine learning is a rapidly evolving field that has shown great potential in various areas of social media by analyzing large amounts of data, machine learning algorithms can identify patterns and classify certain language into hate speech or not.

In this thesis, we will explore the different machine learning techniques used in the detection of hate speech detection, including supervised and unsupervised learning, along with different algorithms. We will also discuss the challenges and limitations of these techniques, as well as their potential applications in large corporations like Twitter.

Acknowledgments

I would like to express my sincere gratitude to all those who have supported me during the course of this Skill Lab and Project Thesis.

Firstly, I would like to thank my project guide, Dr. Satyananda Champati Rai, for his invaluable guidance, support, and encouragement throughout the project. His expertise and insight have been invaluable in shaping this thesis into its final form.

I would also like to thank the Lab Attendant, Mr. Dibakar Pradhan, for his support and guidance, who provided me with insightful feedback and suggestions throughout the thesis.

I would like to express my heartfelt thanks to our family for their unwavering support, encouragement, and patience throughout this project. Their constant support and belief in me have been a source of inspiration and motivation.

Finally, we would like to thank all our friends who have supported me throughout this project, providing me with valuable feedback and suggestions, and always being there to lend a helping hand.

Once again, thank you to all those who have contributed to the successful completion of this project.

Contents

List of Tables

List of Figures

Contents

List of Figures

Chapter 1

Introduction to Machine Learning

1.1 Supervised Learning

Supervised learning, also known as supervised machine learning, is a subcategory of machine learning and artificial intelligence (**?**). In supervised learning, the training data provided to the machines

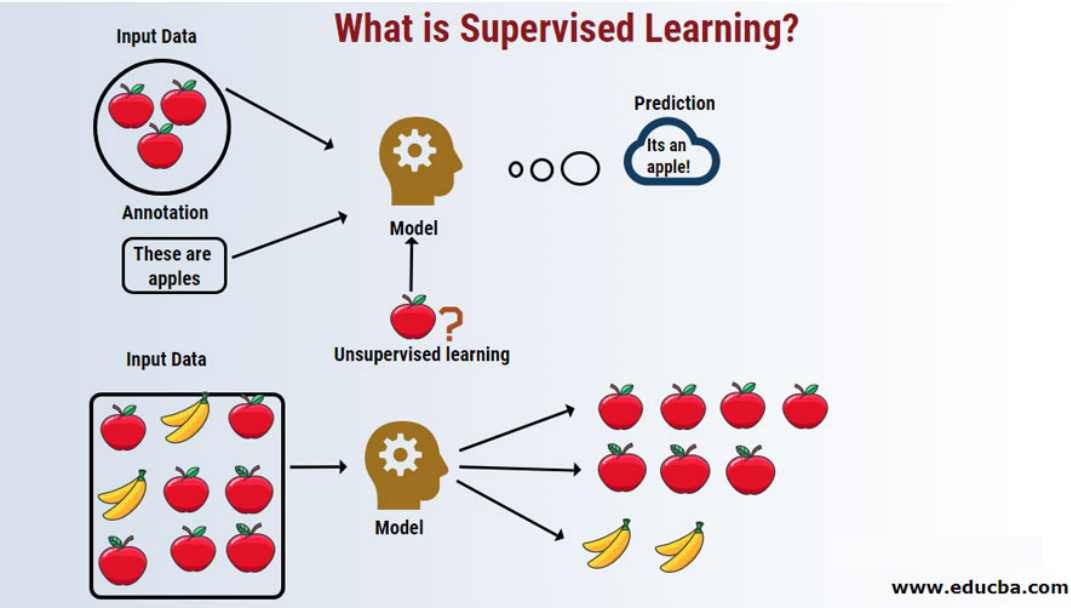


Figure 1.1: Pictorial Representation Of working of Supervised Learning

work as the supervisor that teaches the machines to predict the output correctly. It applies the same concept as a student learns in the supervision of the teacher.
Supervised learning is a process of providing input data as well as correct output data to the machine learning model. The aim of a supervised learning algorithm is to find a mapping function to map the input variable(x) with the output variable(y).

1.2 Unsupervised Learning

Unsupervised learning, uses machine learning algorithms to analyze and cluster unlabeled data sets. These algorithms discover hidden patterns or data groupings without the need for human intervention.

Unsupervised learning cannot be directly applied to a regression or classification problem because

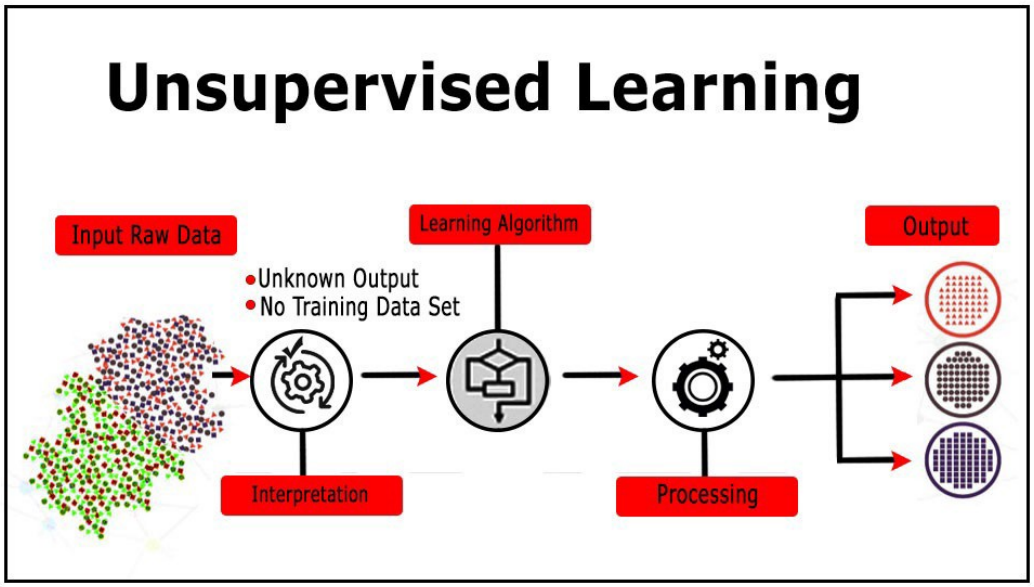


Figure 1.2: Pictorial Representation Of working of Unsupervised Learning

unlike supervised learning, we have the input data but no corresponding output data. The goal of unsupervised learning is to find the underlying structure of dataset, group that data according to similarities, and represent that dataset in a compressed format.

1.3 Reinforcement Learning

Reinforcement learning is a machine learning training method based on rewarding desired behaviors and/or punishing undesired ones.

In general, a reinforcement learning agent is able to perceive and interpret its environment, take actions and learn through trial and error. Reinforcement Learning is the science of decision making. It is about learning the optimal behavior in an environment to obtain maximum reward. In RL, the data is accumulated from machine learning systems that use a trial-and-error method. Data is not part of the input that we would find in supervised or unsupervised machine learning.

Reinforcement learning uses algorithms that learn from outcomes and decide which action to take next. After each action, the algorithm receives feedback that helps it determine whether the choice

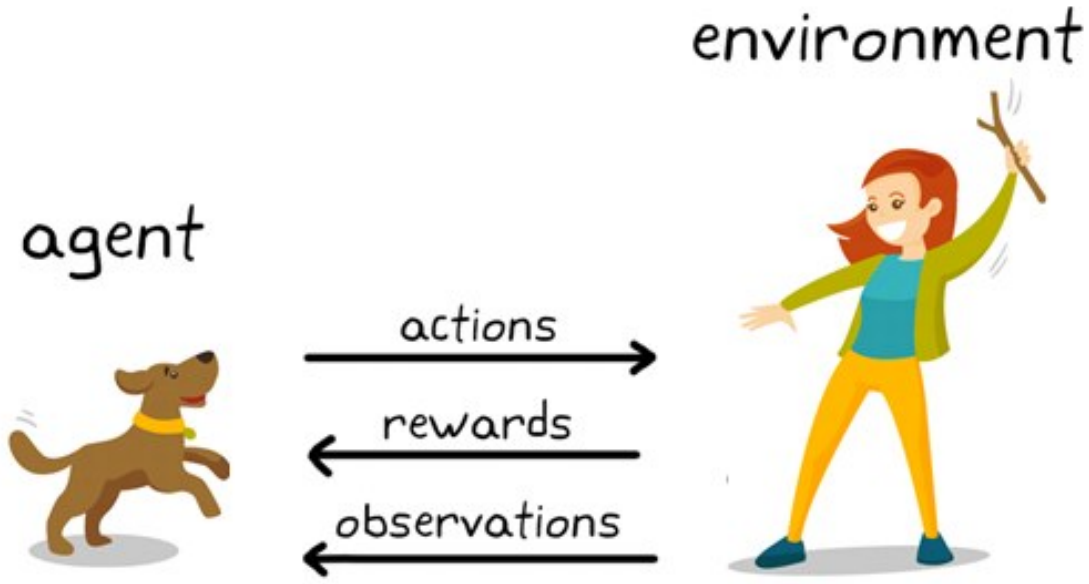


Figure 1.3: Pictorial Representation Of working of Reinforcement Learning

it made was correct, neutral or incorrect. It is a good technique to use for automated systems that have to make a lot of small decisions without human guidance.

1.4 Training Error

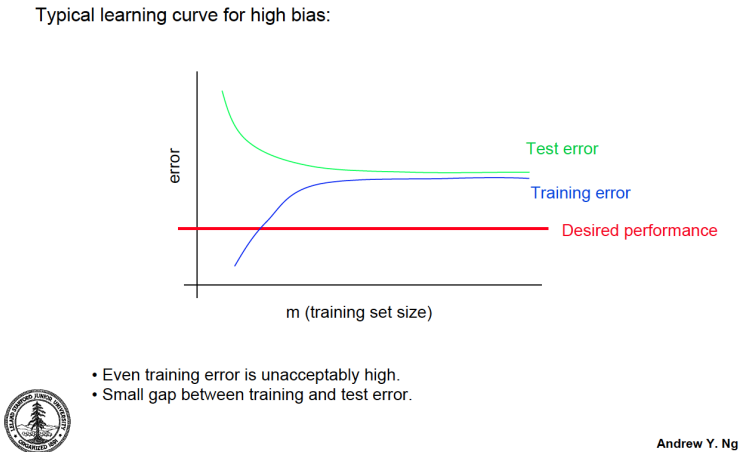


Figure 1.4: Pictorial Representation Of Training error

Training error is simply an error that occurs during model training, that is, a data set inappropriately handled during pre-processing or in feature selection. Training error is the prediction error we get applying the model to the same data from which we trained. Training error is much easier to compute than test error.
Train error is often lower than test error as the model has already seen the training set. It's then going to fit the training set with lower error than it was going to occur on the test set. And the more we over fit, the harder we fit the data, the lower the training error looks. On the other hand, the test error can be quite a bit higher.

1.5 Cross Validation Error

Cross validation is a technique used in machine learning to evaluate the performance of a model on unseen data. It involves dividing the available data into multiple folds or subsets, using one of these folds as a validation set, and training the model on the remaining folds. This process is repeated multiple times, each time using a different fold as the validation set. Finally, the results from each validation step are averaged to produce a more robust estimate of the model's performance.

The main purpose of cross validation is to prevent overfitting, which occurs when a model is

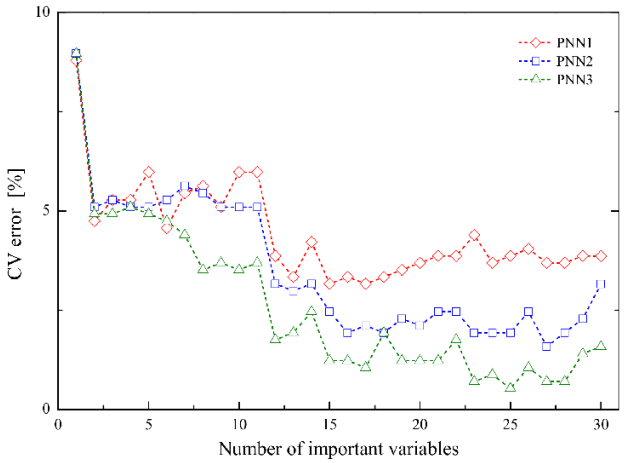


Figure 1.5: Pictorial Representation Of Cross validation error

trained too well on the training data and performs poorly on new, unseen data. By evaluating the model on multiple validation sets, cross validation provides a more realistic estimate of the model's generalization performance, i.e., its ability to perform well on new, unseen data.

1.6 Test Error

We get this by using two completely disjoint data sets: one to train the model and the other to calculate the classification error. Both data set need to have values for y. The first data set is called

training data and the second, test data.

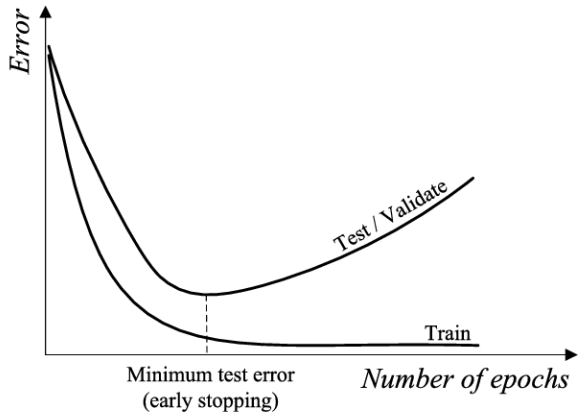


Figure 1.6: Pictorial Representation Of Test Error

1.7 BIAS

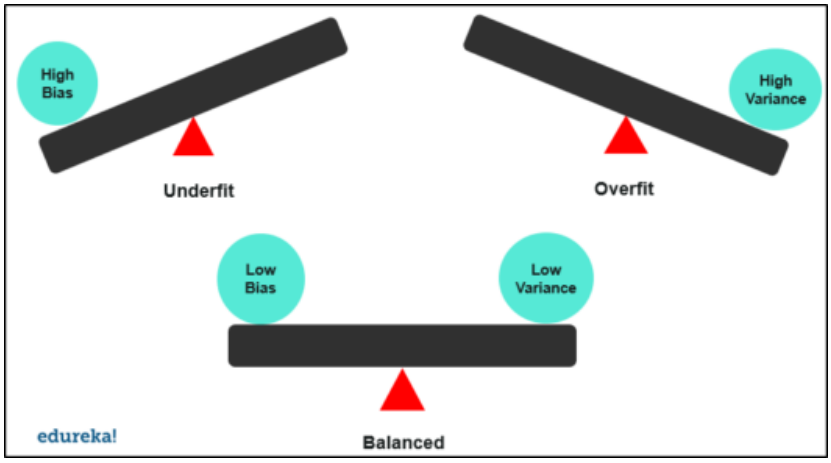


Figure 1.7: Pictorial Representation Of working of BIAS

While making predictions, a difference occurs between prediction values made by the model and actual values/expected values, and this difference is known as bias errors or Errors due to bias. It can be defined as an inability of machine learning algorithms such as Linear Regression to capture the true relationship between the data points. Each algorithm begins with some amount of bias because bias occurs from assumptions in the model, which makes the target function simple to learn.

1.8 Variance

Variance is the measure of spread in data from its mean position. In machine learning variance is the amount by which the performance of a predictive model changes when it is trained on different subsets of the training data. More specifically, variance is the variability of the model that how much it is sensitive to another subset of the training data set i.e. how much it can adjust on the new subset of the training data set.

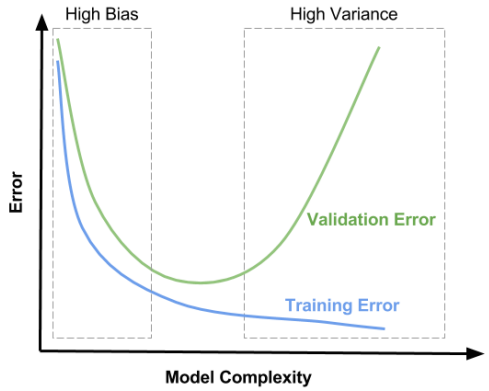


Figure 1.8: Pictorial Representation Of working of Variance

1.9 Confusion Matrix

A Confusion matrix is an N x N matrix used for evaluating the performance of a classification model, where N is the number of target classes. The matrix compares the actual target values with those predicted by the machine learning model.

A confusion matrix is a matrix that summarizes the performance of a machine learning model on a set of test data. It is often used to measure the performance of classification models, which aim to predict a categorical label for each input instance. The matrix displays the number of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) produced by the model on the test data.

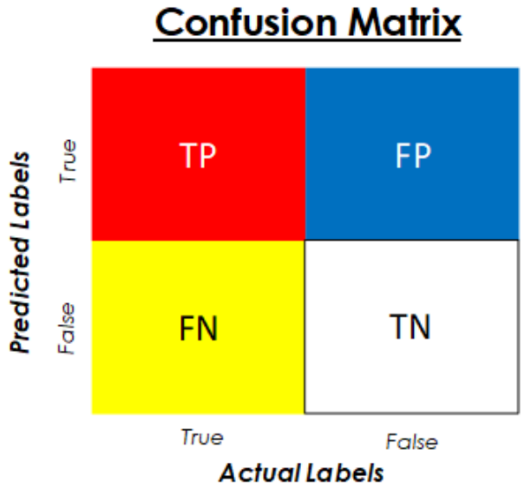


Figure 1.9: Pictorial Representation Of Confusion Matrix

1.10 Recall

Recall in Machine Learning is defined as the ratio of Positive samples that were properly categorized as Positive to the total number of Positive samples. The recall of the model assesses its ability to recognize Positive samples. The more Positive samples identified, the larger the recall.

$$Recall = \frac{truepositives}{truepositives + falsenegatives}$$

Figure 1.10: Pictorial Representation Of Recall

1.11 F Score

The F-score is a way of combining the precision and recall of the model, and it is defined as the harmonic mean of the model's precision and recall. It is commonly used for evaluating information retrieval systems such as search engines, and also for many kinds of machine learning models, in particular in natural language processing.

1.12 R^2

The R2 score is a very important metric that is used to evaluate the performance of a regression-based machine learning model. It is known as the coefficient of determination. It works by measuring the amount of variance in the predictions explained by the dataset. Simply put, it is the difference between the samples in the dataset and the predictions made by the model. R-squared

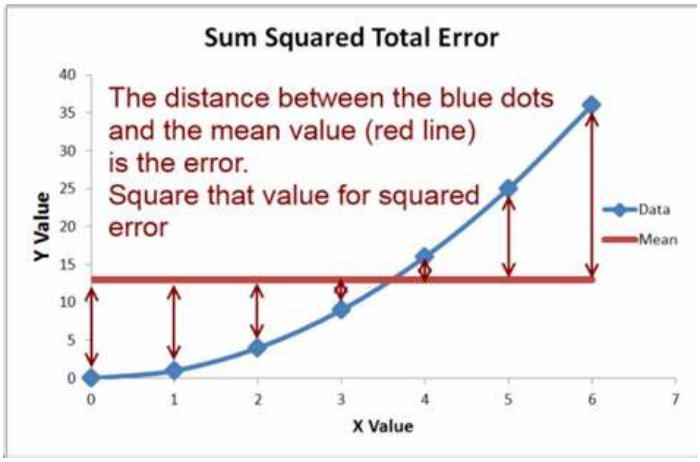


Figure 1.11: Pictorial Representation Of R^2

is a statistical measure that represents the goodness of fit of a regression model. The ideal value for r-square is 1. The closer the value of r-square to 1, the better is the model fitted. R-square is a comparison of the residual sum of squares (SSres) with the total sum of squares(SStot). The total sum of squares is calculated by summation of squares of perpendicular distance between data points and the average line.

1.13 Adjusted R^2

R-squared is a statistical measure that represents the goodness of fit of a regression model. The ideal value for r-square is 1. The closer the value of r-square to 1, the better is the model fitted.

$$Adjusted\ R^2 = 1 - \frac{(1 - R^2)(N - 1)}{N - p - 1}$$

Figure 1.12: Pictorial Representation Of Adjusted R^2

1.14 Model Assessment and Selection

To perform model assessment and selection for comparative analysis of different machine learning methods for hate speech recognition in Twitter text data, you can follow the below steps:

1. Data Preprocessing: The first step is to preprocess the data to ensure it is clean and in the right format. This can include removing special characters, stop words, and stemming or lemmatizing the text.

2. Feature Extraction: The next step is to extract features from the preprocessed text data. This can be done using techniques such as Bag of Words (BoW), Term Frequency-Inverse Document Frequency (TF-IDF), or Word Embeddings.

3. Model Selection: Once the data is preprocessed and the features are extracted, you can proceed with selecting the machine learning models to evaluate. This can include traditional models such as Naive Bayes, Decision Trees, Random Forests, Support Vector Machines (SVMs), or more complex models such as Deep Learning models like Convolutional Neural Networks (CNNs) or Recurrent Neural Networks (RNNs).

4. Model Training and Evaluation: After selecting the models, the next step is to train them on the preprocessed data and evaluate their performance. You can use standard evaluation metrics such as accuracy, precision, recall, F1-score, and area under the receiver operating characteristic curve (AUROC) to compare the performance of different models.

5. Cross-validation: To ensure that the evaluation is robust, you can perform cross-validation by splitting the data into training and testing sets and evaluating the models multiple times using different splits of the data.

6. Model Selection: After evaluating the performance of different models, you can select the best-performing models based on the evaluation metrics and use them for further analysis and interpretation.

7. Interpretation: Finally, you can interpret the results and draw conclusions about the effectiveness of different machine learning methods for hate speech recognition in Twitter text data. You can also identify the strengths and weaknesses of each model and make recommendations for future research and development.

1.15 Project Objective

The objective of the project is to perform a comparative analysis of different machine learning methods for hate speech recognition in Twitter text data. This includes selecting appropriate machine learning algorithms, preprocessing the data, and evaluating the performance of the models using appropriate metrics such as accuracy, precision, recall, and F1-score.

The project aims to identify the most effective machine learning methods for hate speech recognition in Twitter text data, as well as to explore the strengths and weaknesses of different approaches. The project also seeks to provide insights into the factors that affect the performance of machine learning models in this domain, such as the quality and size of the training data, the choice of features, and the tuning of hyperparameters.

Ultimately, the goal of the project is to develop a comprehensive understanding of the state-of-the-art in hate speech recognition in Twitter text data, and to identify best practices and areas for future research. The results of the project can be used to inform the development of more effective and robust machine learning models for hate speech recognition, which can help to improve online safety and combat hate speech and other forms of harmful content.

1.16 Chapter Outline

The thesis on Comparative analysis of different Machine Learning methods for Hate Speech Recognition in Twitter text data aims to improve the detection of hate speech through the use of machine learning algorithms. It begins with an introduction to the importance of a good environment in scoial media where young minds are exposed to the world. The thesis then provides data on different types of hate speech and offensive langauge and algorithms to curb the same. The dataset used in the study is described, along with data preprocessing techniques and data visualization and exploration. Machine learning models are then discussed, along with evaluation metrics for assessing model performance and the experimental design and methodology. The experimental results and performance evaluation of the machine learning models are presented, including comparisons with existing approaches and analysis of feature importance and model interpretability. The thesis concludes with a summary of the main

findings, implications and limitations of the study, recommendations for future research, and technical information in the appendix.

1.17 Conclusion

In conclusion, machine learning is a rapidly growing field with significant potential to transform industries and drive innovation. From image and speech recognition to medical diagnosis and predictive maintenance, machine learning is being used to solve a wide range of real-world problems. However, machine learning also presents challenges such as overfitting, bias, and lack of interpretability. These challenges must be carefully considered and addressed in order to ensure the reliability and effectiveness of machine learning models. Despite these challenges, the future of machine learning looks bright. As computing power and data storage capabilities continue to increase, the potential applications of machine learning will only continue to expand.

Chapter 2

Mathematical foundation of ML

In this Chapter we are going to learn about various Mathematical foundation of ML such as:-

2.1 Matrix Calculus

Matrix calculus is a specialized notation for doing multi-variable calculus, especially over spaces of matrices. It collects the various partial derivatives of a single function with respect to many variables, and/or of a multivariate function with respect to a single variable, into vectors and matrices that can be treated as single entities.

Linear Algebra: Linear algebra is used to represent and manipulate data in machine learning. It includes concepts such as matrices, vectors, and tensors.

Calculus: Calculus is used to optimize the performance of machine learning models. It includes concepts such as derivatives, integrals, and optimization algorithms.

Probability and Statistics: Probability and statistics are used to model uncertainty in machine learning. It includes concepts such as probability distributions, statistical inference, and hypothesis testing.

2.2 Eigen values & Eigen vector

$$Av = \lambda v$$

An eigenvector or characteristic vector of a linear transformation is a nonzero vector that changes at most by a scalar factor when that linear transformation is applied to it. The corresponding eigenvalue is the factor by which the eigenvector is scaled.

2.3 Random variable and distribution

A random variable is a numerical description of the outcome of a statistical experiment. The probability distribution for a random variable describes how the probabilities are distributed over the values of the random variable.

Normal distribution

The normal distribution, also known as the Gaussian distribution, is a widely used probability distribution in machine learning. It is a bell-shaped distribution that is characterized by two parameters, mean and variance. The normal distribution is commonly used to model noise or error in the data.

Bernoulli distribution

The Bernoulli distribution is a discrete probability distribution that takes on two values, usually 0 and 1. It is often used to model binary outcomes, such as the probability of a coin flip.

Binomial distribution

The binomial distribution is a discrete probability distribution that is used to model the number of successes in a fixed number of independent trials. It is often used in machine learning to model the probability of a binary classification problem.

Poisson distribution

The Poisson distribution is a discrete probability distribution that is used to model the number of events that occur in a fixed interval of time or space. It is commonly used to model rare events, such as the number of customers visiting a store in a given time interval.

Gaussian distribution

Gaussian distribution is a bell-shaped curve, and it is assumed that during any measurement values will follow a normal distribution with an equal number of measurements above and below the mean value.

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

2.4 Kernel

Kernel is a computer program at the core of a computer's operating system and generally has complete control over everything in the system. It is the portion of the operating system code that is always resident in memory and facilitates interactions between hardware and software components.

Polynomial Kernel

The polynomial kernel is a kernel function commonly used with support vector machines and other kernelized models, that represents the similarity of vectors in a feature space over polynomials of the original variables, allowing learning of non-linear models.

$$K(x,y) = (x^T y + c)^d$$

Gaussian Kernel

The Gaussian kernel is the physical equivalent of the mathematical point. It is not strictly local, like the mathematical point, but semi-local. It has a Gaussian weighted extent, indicated by its inner scale s.

$$K(x,y) = \exp(-\gamma||x-y||^2)$$

Chapter 3

ML Algorithm Codes

3.1 Import Dataset

```
import pandas as pd
data = pd.read_csv('Boston.csv')
#print
df = pd.DataFrame(data)
print(df)
```

```
        crim    zn  indus  chas    nox     rm   age     dis  rad  tax  \
0    0.00632  18.0   2.31     0  0.538  6.575  65.2  4.0900    1  296
1    0.02731   0.0   7.07     0  0.469  6.421  78.9  4.9671    2  242
2    0.02729   0.0   7.07     0  0.469  7.185  61.1  4.9671    2  242
3    0.03237   0.0   2.18     0  0.458  6.998  45.8  6.0622    3  222
4    0.06905   0.0   2.18     0  0.458  7.147  54.2  6.0622    3  222
..       ...   ...    ...   ...    ...    ...   ...     ...  ...  ...
501  0.06263   0.0  11.93     0  0.573  6.593  69.1  2.4786    1  273
502  0.04527   0.0  11.93     0  0.573  6.120  76.7  2.2875    1  273
503  0.06076   0.0  11.93     0  0.573  6.976  91.0  2.1675    1  273
504  0.10959   0.0  11.93     0  0.573  6.794  89.3  2.3889    1  273
505  0.04741   0.0  11.93     0  0.573  6.030  80.8  2.5050    1  273

     ptratio   black  lstat  medv
0       15.3  396.90   4.98  24.0
1       17.8  396.90   9.14  21.6
2       17.8  392.83   4.03  34.7
3       18.7  394.63   2.94  33.4
4       18.7  396.90   5.33  36.2
```

```
df.shape
X = df.iloc[:,:13]
X
```

7]:

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33
...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0	0.573	6.593	69.1	2.4786	1	273	21.0	391.99	9.67
502	0.04527	0.0	11.93	0	0.573	6.120	76.7	2.2875	1	273	21.0	396.90	9.08
503	0.06076	0.0	11.93	0	0.573	6.976	91.0	2.1675	1	273	21.0	396.90	5.64
504	0.10959	0.0	11.93	0	0.573	6.794	89.3	2.3889	1	273	21.0	393.45	6.48
505	0.04741	0.0	11.93	0	0.573	6.030	80.8	2.5050	1	273	21.0	396.90	7.88

506 rows × 13 columns

```
Y = df.iloc[:,13:14]
Y
xnew = df.iloc[505:506,:13]
xnew
```

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat
505	0.04741	0.0	11.93	0	0.573	6.03	80.8	2.505	1	273	21.0	396.9	7.88

3.2 KNNC

K-Nearest Neighbors is one of the most basic yet essential classification algorithms in Machine Learning. It belongs to the supervised learning domain and finds intense application in pattern recognition, data mining, and intrusion detection.

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

scaler.fit(df.drop('TARGET CLASS', axis = 1))
scaled_features = scaler.transform(df.drop('TARGET CLASS', axis = 1))

df_feat = pd.DataFrame(scaled_features, columns = df.columns[:-1])
df_feat.head()

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
      scaled_features, df['TARGET CLASS'], test_size = 0.30)

# Remember that we are trying to come up
# with a model to predict whether
# someone will TARGET CLASS or not.
# We'll start with k = 1.

from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors = 1)

knn.fit(X_train, y_train)
pred = knn.predict(X_test)

# Predictions and Evaluations
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, pred))

print(classification_report(y_test, pred))

error_rate = []

# Will take some time
for i in range(1, 40):
```

```
    knn = KNeighborsClassifier(n_neighbors = i)
    knn.fit(X_train, y_train)
    pred_i = knn.predict(X_test)
    error_rate.append(np.mean(pred_i != y_test))

plt.figure(figsize =(10, 6))
plt.plot(range(1, 40), error_rate, color ='blue',
                linestyle ='dashed', marker ='o',
         markerfacecolor ='red', markersize = 10)

plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
```

3.3 KNNR

```
import numpy as np
import pandas as pd
dataFrame = pd.read_csv('Boston.csv')
rows = 506
cols=14
Dataset = np.zeros((rows-1,cols-1))
Ys = np.zeros((rows-1))
Wholedataset = np.array(dataFrame)

for i in range(rows-1):
    for j in range(cols-1):
        Dataset[i][j] = Wholedataset[i][j]
for i in range(rows-1):
    Ys[i] = Wholedataset[i][13]
newdata = np.array([0.04741,0,11.93,0,0.573,6.03,80.8,2.505,1,273,21,396.9,7.88])

distances = []
YValues = []
KR =5
sum1 = 0
for i in range(rows-1):
    distance= ((Dataset[i,:]-newdata) **2).sum()
    distances.append([distance,i])
distances = sorted(distances)
```

```
for i in range(KR):
    index = distances[i][1]
    YValues.append(YR[index])

for i in range(KR):
    sum1 = sum1 + YValues[i]
avg = sum1/KR

print("KNN-r",avg)
```

KNN-r 23.759999999999998

3.4 MLR

```
import numpy as np
#Transpose
XT = X.transpose()
print(XT)
```

```
                0        1        2        3        4        5    \
crim      0.00632  0.02731  0.02729  0.03237  0.06905  0.02985
zn       18.00000  0.00000  0.00000  0.00000  0.00000  0.00000
indus     2.31000  7.07000  7.07000  2.18000  2.18000  2.18000
chas      0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
nox       0.53800  0.46900  0.46900  0.45800  0.45800  0.45800
rm        6.57500  6.42100  7.18500  6.99800  7.14700  6.43000
age      65.20000 78.90000 61.10000 45.80000 54.20000 58.70000
dis       4.09000  4.96710  4.96710  6.06220  6.06220  6.06220
rad       1.00000  2.00000  2.00000  3.00000  3.00000  3.00000
tax     296.00000 242.00000 242.00000 222.00000 222.00000 222.00000
ptratio  15.30000 17.80000 17.80000 18.70000 18.70000 18.70000
black   396.90000 396.90000 392.83000 394.63000 396.90000 394.12000
lstat     4.98000  9.14000  4.03000  2.94000  5.33000  5.21000

                6        7        8        9    ...      496      497  \
crim      0.08829  0.14455  0.21124  0.17004  ...   0.2896   0.26838
zn       12.50000 12.50000 12.50000 12.50000  ...   0.0000   0.00000
indus     7.87000  7.87000  7.87000  7.87000  ...   9.6900   9.69000
chas      0.00000  0.00000  0.00000  0.00000  ...   0.0000   0.00000
nox       0.52400  0.52400  0.52400  0.52400  ...   0.5850   0.58500
rm        6.01200  6.17200  5.63100  6.00400  ...   5.3900   5.79400
age      66.60000 96.10000 100.00000 85.90000  ...  72.9000  70.60000
dis       5.56050  5.95050  6.08210  6.59210  ...   2.7986   2.89270
rad       5.00000  5.00000  5.00000  5.00000  ...   6.0000   6.00000
tax     311.00000 311.00000 311.00000 311.00000  ...  391.0000 391.00000
ptratio  15.20000 15.20000 15.20000 15.20000  ...  19.2000  19.20000
black   395.60000 396.90000 386.63000 386.71000  ...  396.9000 396.90000
lstat    12.43000 19.15000 29.93000 17.10000  ...  21.1400  14.10000

                498      499      500      501      502      503  \
crim      0.23912  0.17783  0.22438  0.06263  0.04527  0.06076
zn        0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
```

```
#Dot product of XTX
XTX=np.dot(XT,X)
print(XTX)
```

```
[[4.39703436e+04 4.68702660e+02 3.24790952e+04 6.48084600e+01
  1.22612317e+03 1.08219511e+04 1.68514980e+05 3.46627456e+03
  4.11186651e+04 1.17307321e+06 3.64715518e+04 4.99455291e+05
  3.72684187e+04]
 [4.68702660e+02 3.40029000e+05 2.09030900e+04 2.70000000e+02
  2.48444175e+03 3.87184670e+04 2.05485400e+05 3.82992940e+04
  2.29180000e+04 1.72295450e+06 9.61321500e+04 2.23960499e+06
  3.80196800e+04]
 [3.24790952e+04 2.09030900e+04 8.65256299e+04 4.45170000e+02
  3.43239536e+03 3.44618165e+04 4.49313490e+05 1.62206733e+04
  7.17656500e+04 2.72134904e+06 1.06875320e+05 1.89702529e+06
  8.62407050e+04]
 [6.48084600e+01 2.70000000e+02 4.45170000e+02 3.50000000e+01
  2.07699000e+01 2.28186000e+02 2.71250000e+03 1.06039800e+02
  3.26000000e+02 1.35190000e+04 6.12200000e+02 1.30549100e+04
  3.93460000e+02]
 [1.22612317e+03 2.48444175e+03 3.43239536e+03 2.07699000e+01
  1.62470380e+02 1.75151941e+03 2.04522016e+04 9.70389866e+02
  2.99183590e+03 1.21170623e+05 5.20395546e+03 9.80793458e+04
  3.79832520e+03]
 [1.08219511e+04 3.87184670e+04 3.44618165e+04 2.28186000e+02
  1.75151941e+03 2.02345982e+04 2.15670176e+05 1.22216807e+04
  2.97190270e+04 1.28073971e+06 5.84159728e+04 1.13838092e+06
  3.86817880e+04]
 [1.68514980e+05 2.05485400e+05 4.49313490e+05 2.71250000e+03
  2.04522016e+04 2.15670176e+05 2.77961463e+06 1.09297470e+05
  3.87798200e+05 1.53787387e+07 6.48434860e+05 1.20212117e+07
```

```
#Inverse of XTX
XTXI = (np.linalg.inv(XTX))
print(XTXI)
# dot product of XTY
XTY = np.dot(XT,Y)
print(XTY)
#Find the value of Bita
Bt = np.dot(XTXI,XTY)
print(Bt)
```

```
[[-9.28965170e-02]
 [ 4.87149552e-02]
 [-4.05997958e-03]
 [ 2.85399882e+00]
 [-2.86843637e+00]
 [ 5.92814778e+00]
 [-7.26933458e-03]
 [-9.68514157e-01]
 [ 1.71151128e-01]
 [-9.39621540e-03]
 [-3.92190926e-01]
 [ 1.49056102e-02]
 [-4.16304471e-01]]
```

```
#Find the Bita till
BtT = np.insert(Bt,0,1)
print(BtT)
Xnew = np.array(xnew)
print(Xnew)
#X new Til
XNT = np.insert(Xnew,0,1)
print(XNT)

# Find the value of Y new
YN = np.dot(BtT,XNT)
print(YN)
```

24.04231943205724

3.5 Ridge Regression

```
#create IDENTTY matrix
I = np.eye(13,13)
print(I)
print(I.shape)
```

```
[[1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]]
(13, 13)
```

```
# take the lamda value =0.5
# l= 0.4
l=0.9
# matrix multiplication
IL = np.dot(l,I)
print(IL)
print(IL.shape)

#Find the sum of product
XTXIL = np.add(XTX, IL)
print(XTXIL)
print(XTXIL.shape)
#find the inverse of lamda& I

XTXILI = (np.linalg.inv(XTXIL))
print(XTXILI)
print(XTXILI.shape)
```

```
[[4.39712436e+04 4.68702660e+02 3.24790952e+04 6.48084600e+01
  1.22612317e+03 1.08219511e+04 1.68514980e+05 3.46627456e+03
  4.11186651e+04 1.17307321e+06 3.64715518e+04 4.99455291e+05
  3.72684187e+04]
 [4.68702660e+02 3.40029900e+05 2.09030900e+04 2.70000000e+02
  2.48444175e+03 3.87184670e+04 2.05485400e+05 3.82992940e+04
  2.29180000e+04 1.72295450e+06 9.61321500e+04 2.23960499e+06
  3.80196800e+04]
 [3.24790952e+04 2.09030900e+04 8.65265299e+04 4.45170000e+02
  3.43239536e+03 3.44618165e+04 4.49313490e+05 1.62206733e+04
  7.17656500e+04 2.72134904e+06 1.06875320e+05 1.89702529e+06
  8.62407050e+04]
 [6.48084600e+01 2.70000000e+02 4.45170000e+02 3.59000000e+01
  2.07699000e+01 2.28186000e+02 2.71250000e+03 1.06039800e+02
  3.26000000e+02 1.35190000e+04 6.12200000e+02 1.30549100e+04
  3.93460000e+02]
 [1.22612317e+03 2.48444175e+03 3.43239536e+03 2.07699000e+01
  1.63370380e+02 1.75151941e+03 2.04522016e+04 9.70389866e+02
  2.99183590e+03 1.21170623e+05 5.20395546e+03 9.80793458e+04
```

```
BR = np.dot(XTXILI,XTY)
print(BR.shape)
print(BR)
#find the bita mean
min =np.mean(Y, axis=0)
# sumY = Y.sum()
# print(sumY)
# min = np.divide(sumY,506)
print(min)
```

medv 22.532806
dtype: float64

```
# find bita til
BOTil = np.insert(BR,0,min,axis = 0)
print(BOTil)
# find bita transpose
BOT = BOTil.transpose()
print(BOT)
print(XNT)
YN = np.dot(BOT,XNT)
print(YN)
```

[45.63864252]

3.6 LASSO

```
LL = np.divide(l,2)
print(LL)
```

0.45

```
ILL = np.dot(LL,I)
print(ILL)
print(IL.shape)

#Find the sum of product
XTXILL = np.add(XTX, ILL)
print(XTXILL)
print(XTXILL.shape)
#find the inverse of lamda& I

XTXILLI = (np.linalg.inv(XTXILL))
print(XTXILLI)
print(XTXILLI.shape)

BRL = np.dot(XTXILLI,XTY)
print(BRL.shape)
print(BRL)
#find the bita mean
min =np.mean(Y, axis=0)
# sumY = Y.sum()
# print(sumY)
# min = np.divide(sumY,506)
print(min)
```

medv 22.532806
dtype: float64

```
# find bita til
BOTilL = np.insert(BRL,0,min,axis = 0)
print(BOTilL)
# find bita transpose

BOTL = BOTilL.transpose()
print(BOTL)
```

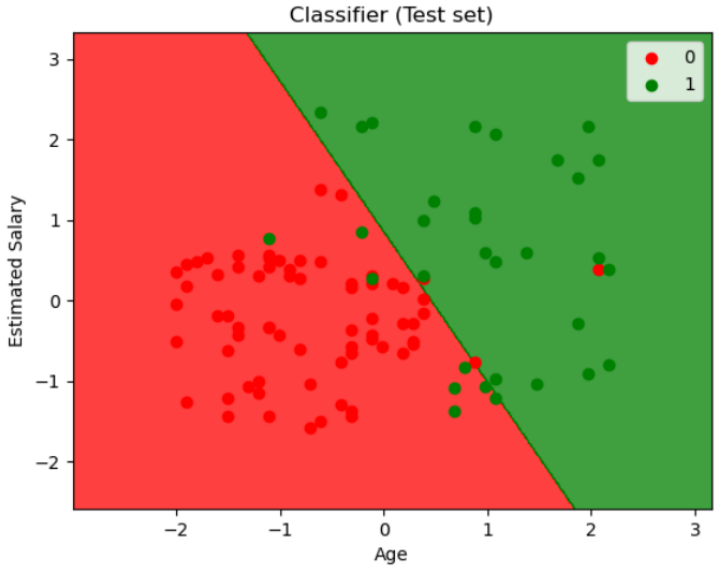
```
YNL = np.dot(BOTL,XNT)
print(YNL)
```

[45.61051951]

3.7 Logistic Regression

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

dataset = pd.read_csv('classificationdata.csv')

# input
x = dataset.iloc[:, [2, 3]].values

# output
y = dataset.iloc[:, 4].values

from sklearn.model_selection import train_test_split

xtrain, xtest, ytrain, ytest = train_test_split(
    x, y, test_size=0.25, random_state=0)

from sklearn.preprocessing import StandardScaler

sc_x = StandardScaler()
xtrain = sc_x.fit_transform(xtrain)
xtest = sc_x.transform(xtest)

print (xtrain[0:10, :])

from sklearn.linear_model import LogisticRegression

classifier = LogisticRegression(random_state = 0)
classifier.fit(xtrain, ytrain)

y_pred = classifier.predict(xtest)

from sklearn.metrics import confusion_matrix

cm = confusion_matrix(ytest, y_pred)
```

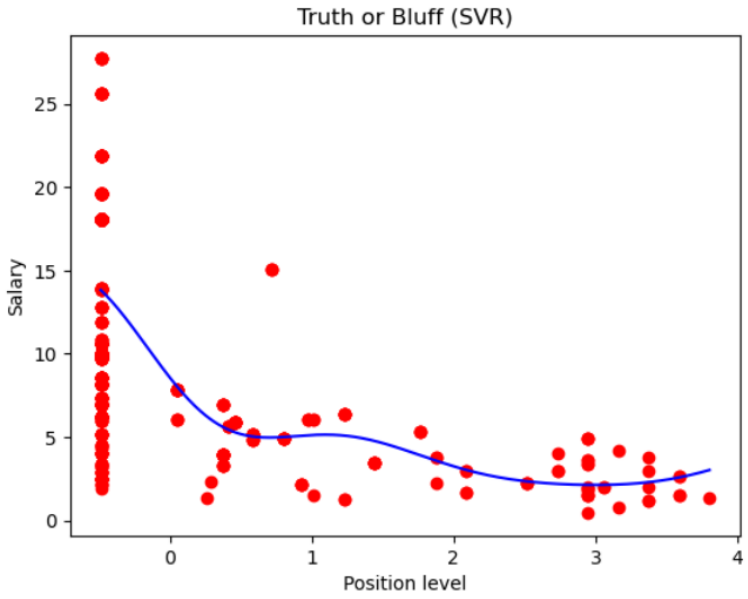
```
print ("Confusion Matrix : \n", cm)

from sklearn.metrics import accuracy_score

print ("Accuracy : ", accuracy_score(ytest, y_pred))

from matplotlib.colors import ListedColormap

X_set, y_set = xtest, ytest
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1,
                               stop = X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1,
                               stop = X_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(
    np.array([X1.ravel(), X2.ravel()]).T).reshape(
    X1.shape), alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)

plt.title('Classifier (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```



3.8 SVM

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

#Feature Scaling
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
sc_Y = StandardScaler()
X = sc_X.fit_transform(X)
Y = sc_Y.fit_transform(Y)

#Fitting SVR to the dataset
from sklearn.svm import SVR
regressor = SVR(kernel='rbf')
regressor.fit(X,Y)

Y_pred = regressor.predict(:)
Y_pred = sc_Y.inverse_transform(Y_pred)

dataset = pd.read_csv('Boston.csv')
X = dataset.iloc[:, 1:2].values
y = dataset.iloc[:, 2].values
```

```
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
sc_y = StandardScaler()
X = sc_X.fit_transform(X)
y = sc_y.fit_transform(y)

from sklearn.svm import SVR
regressor = SVR(kernel = 'rbf')
regressor.fit(X, y)

y_pred = regressor.predict(6.5)
y_pred = sc_y.inverse_transform(y_pred)

X_grid = np.arange(min(X), max(X), 0.01) #this step required because data is featur
X_grid = X_grid.reshape((len(X_grid), 1))
plt.scatter(X, y, color = 'red')
plt.plot(X_grid, regressor.predict(X_grid), color = 'blue')
plt.title('Truth or Bluff (SVR)')
plt.xlabel('Position level')
plt.ylabel('Salary')
plt.show()
```



Chapter 4

Machine Learning for Hate Speech Recognition in Twitter Text Data.

Hate Speech Recognition in Twitter Data using Machine Learning is our proposed model for this project. We will use the Naive Bayes and Logistic Regression Model for this project.

4.1 Overview

The purpose of this work is to solve the challenges faced by us in the field of hate speech recognition on various social media platforms, that is to get a better machine learning model that can detect hate speech with greater accuracy. As the reach of the internet and mobile phones has extended abruptly in the past few years, everyone has the power to share their opinions, but some use it as an opportunity to spread hate among one another. In this paper, we used the Davidson [10] dataset which is the most popular Twitter dataset for hate speech detection, and further we implemented various machine learning-based algorithms and compared them on the basis of various parameters such as accuracy, precision score, recall and F1 scores. After the study, we found out that XGBoost when used with TF-IDF transformer embedding gave us an accuracy of 94.43is the maximum among these three models for the given benchmark dataset.

4.2 Background

The study of human activities is an active research area, and the fast growth of digitalization and social networking websites has increased interaction between people from different psychological and cultural backgrounds. Hence, people have come closer to one another, which has increased "digital" conflicts between them. Consequently, hate speech is now the hottest topic in the field of NLP for researchers and the government. Social media companies are investing a huge amount of money in filtering negative or hateful contents to keep the platform safe for users. Therefore, to find a solution and to come up with a better approach, There is no scale defined to differentiate between hateful and offensive speech, nor is there any formal

definition, but for the sake of simplicity, we can simply describe any group of words or sentences that hurt the feelings of any individual or group and cause them to harm in any form, be it emotionally or socially. Almost every country and community has defined certain rules and regulations in their beliefs, and everyone should act under those rules. And violating any of those rules may cause them huge fines or even send them to jail. Those rules and laws also extend to social media platforms and even those platforms, provide certain rules of action to prevent hate speech. Twitter along with Facebook has faced huge criticism for not doing enough to ensure the feelings of the

users and communities are not hurt by any means. They are even forced to face the approved laws related to common people and their differences, like race, colour, gender, sexual orientation, etc. Based on the above discussion, hate speech is defined as any language or gestures in physical form that can deliver hatred toward a targeted group of people to humiliate and insult its members. It may get even worse in extreme scenarios and can even lead to violence, but for the sake of this paper, we will be restricting our discussion only to hate speech. It should be noted that cultural diversity should also be taken care of, because something offensive to one community may not be considered offensive in another. Like the word teenagers use in the everyday conversation cannot be posted on Facebook and Twitter. This term was considered in most of the previous works as well but there was still a lot of confusion and miscommunication. Due to such vast networks, it is impossible to detect and classify each and every piece of content on the internet. That's why we need automated methods to detect hate speech.

We have used the Davidson dataset [10] because each tweet in this dataset is manually classified as "hate", "offensive" or "neither" by the author, so it is pretty much accurate. The main player and opponent in the way of automatic hate speech detection on social media and other communication related platforms are to separate hate speech from other families of impressions and wordings nearly belonging to the same categories and hierarchies. We implemented three algorithms named LSTM, Naïve Bayes, and XGBoost. The latter model outperforms the other two models and gives the best results among the models. In the next part of this paper, we will discuss the related work that has been done in this field by many authors, followed by the details of the dataset we have used in this paper. In section four, we will discuss the models we implemented, followed by the results obtained by these models and a comparative study of these models on different parameters. In the last part, we will conclude our work with some possible future scope in this field.

4.3 Importing Dataset for Naive Bayes Classification

```
import numpy as np
import pandas as pd
import json
import requests
import matplotlib.pyplot as plt
```

```
plt.style.use('ggplot')
import seaborn as sns
from sklearn import tree
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor

#Loading Labeled Dataset with hate tweets IDs
data =  pd.read_csv('twitter_data.csv')
data.head
```

Out[3]:

	Unnamed: 0	count	hate_speech	offensive_language	neither	class	tweet
0	0	3	0	0	3	2	!!! RT @mayasolovely: As a woman you shouldn't...
1	1	3	0	3	0	1	!!!!! RT @mleew17: boy dats cold...tyga dwn ba...
2	2	3	0	3	0	1	!!!!!!! RT @UrKindOfBrand Dawg!!!! RT @80sbaby...
3	3	3	0	2	1	1	!!!!!!!!! RT @C_G_Anderson: @viva_based she lo...
4	4	6	0	6	0	1	!!!!!!!!!!!! RT @ShenikaRoberts: The shit you...

4.4 ARRANGE DATA INTO FEATURES AND TARGET

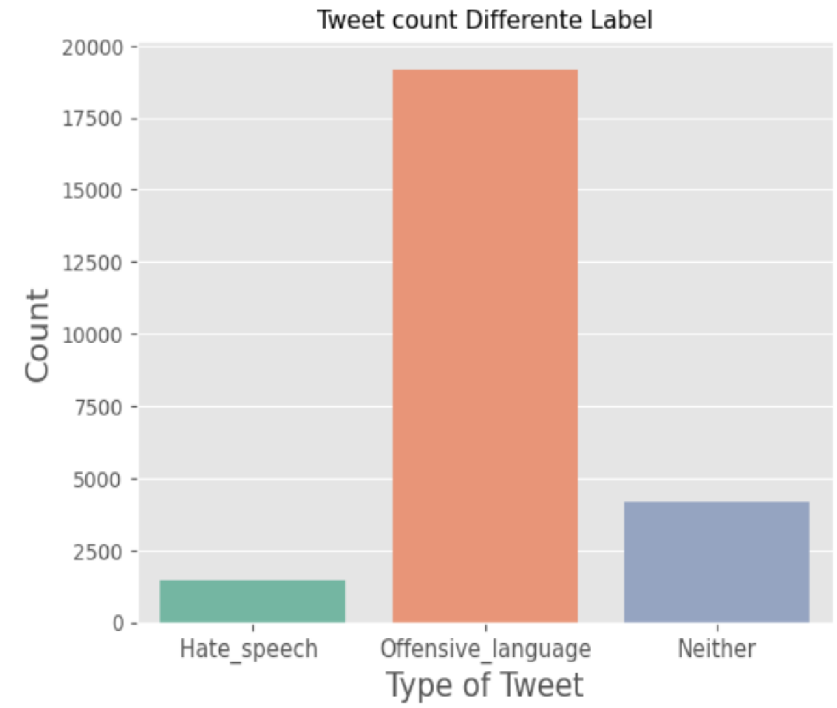
```
features = ['count','hate_speech','offensive_language','neither']
X = data.loc[:, features]
Y = data.loc[:, ['class']]
```

Out[8]:

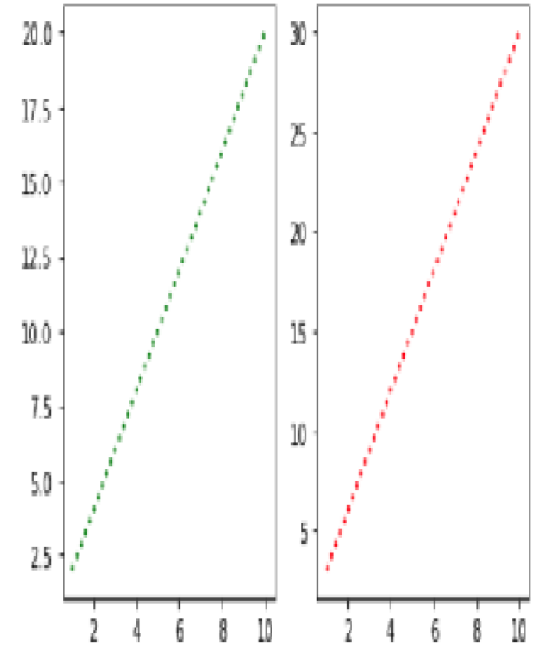
	count	hate_speech	offensive_language	neither
0	3	0	0	3
1	3	0	3	0
2	3	0	3	0
3	3	0	2	1
4	6	0	6	0
...	...	...	...	...
24778	3	0	2	1
24779	3	0	1	2
24780	3	0	3	0
24781	6	0	6	0
24782	3	0	0	3

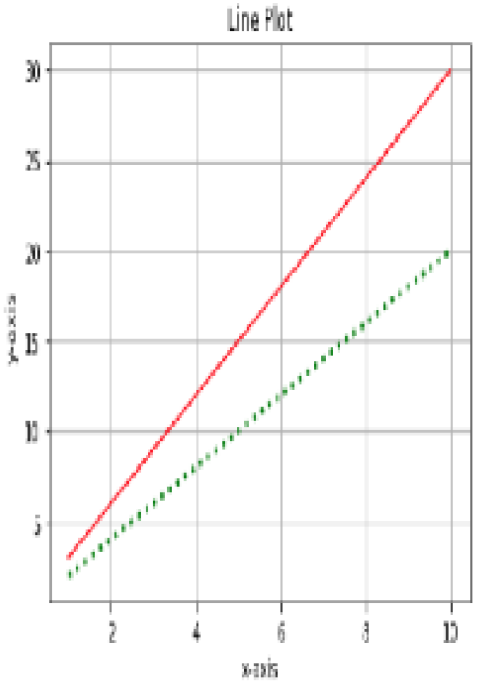
24783 rows × 4 columns

```
# creating feature bar plot

ax = sns.countplot(data['class'], palette='Set2')

ax.set_title('Tweet count Differente Label',fontsize = 12)
ax.set_xlabel('Type of Tweet',fontsize = 15)
ax.set_ylabel('Count',fontsize = 15)

ax.set_xticklabels(['Hate_speech ','Offensive_language', 'Neither'],fontsize = 11)

total = float(len(data)) # one person per row
```

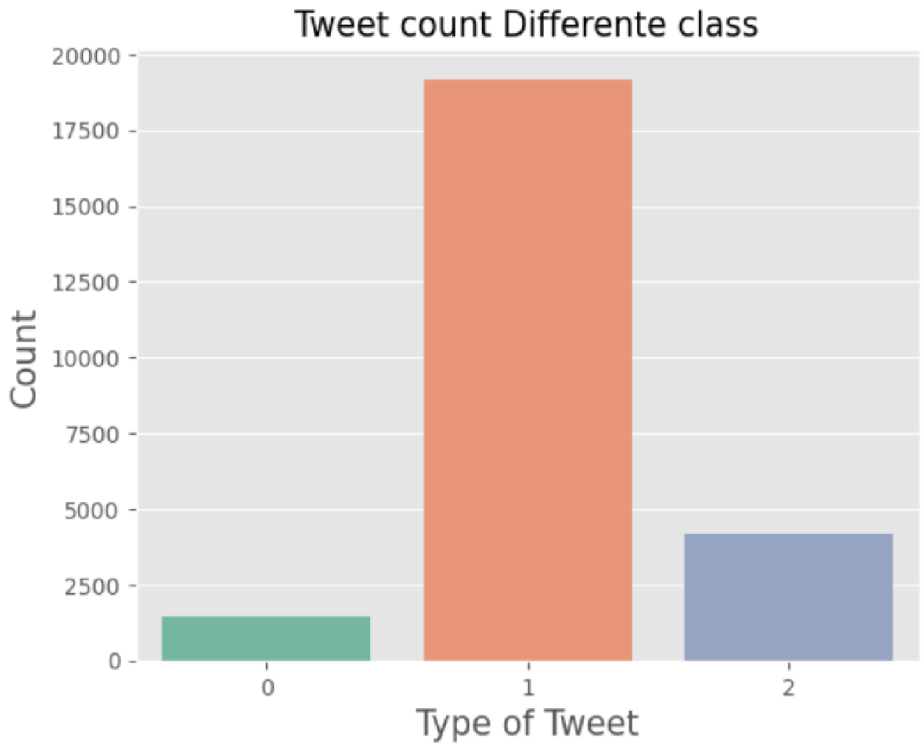


```
# from matplotlib import pyplot as plt

xaxis = ['hate_speech']
yaxis=['offensive_language']
zaxis=['neither']

plt.plot(xaxis,yaxis,color='g',linestyle=':',linewidth=2.5)
plt.plot(xaxis,zaxis,color='r',linestyle='-',linewidth=1.3)
plt.title("Line Plot")
plt.xlabel("x-axis")
plt.ylabel("y-axis")
plt.grid(True)
plt.show()
plt.plot(xaxis, yaxis)
plt.show
# plt.subplot(1,2,1)
# plt.plot(xaxis,yaxis,color='g',linestyle=':',linewidth=2)
# plt.subplot(1,2,2)
# plt.plot(xaxis,zaxis,color='r',linestyle=':',linewidth=2)
# plt.show()
```



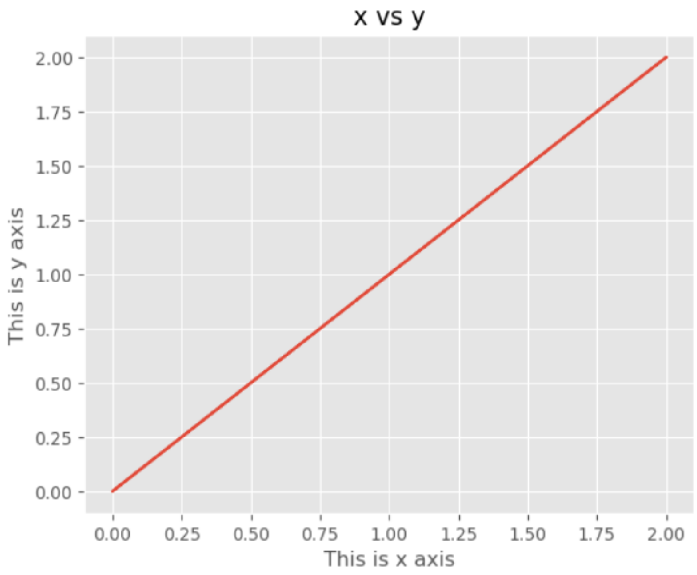


```
plt.plot(Y, 'o:r')
plt.show()

# Class Imbalance

# fig, ax = plt.subplots(figsize=(10,6))
ax = sns.countplot(data['class'], palette='Set2')

ax.set_title('Tweet count Differente class',fontsize = 15)
ax.set_xlabel('Type of Tweet',fontsize = 15)
ax.set_ylabel('Count',fontsize = 15)
# ax.set_xticklabels(['Hate_speech (0)','Offensive_language(1)', 'Neither(2)'],font

total = float(len(data)) # one person per row
```

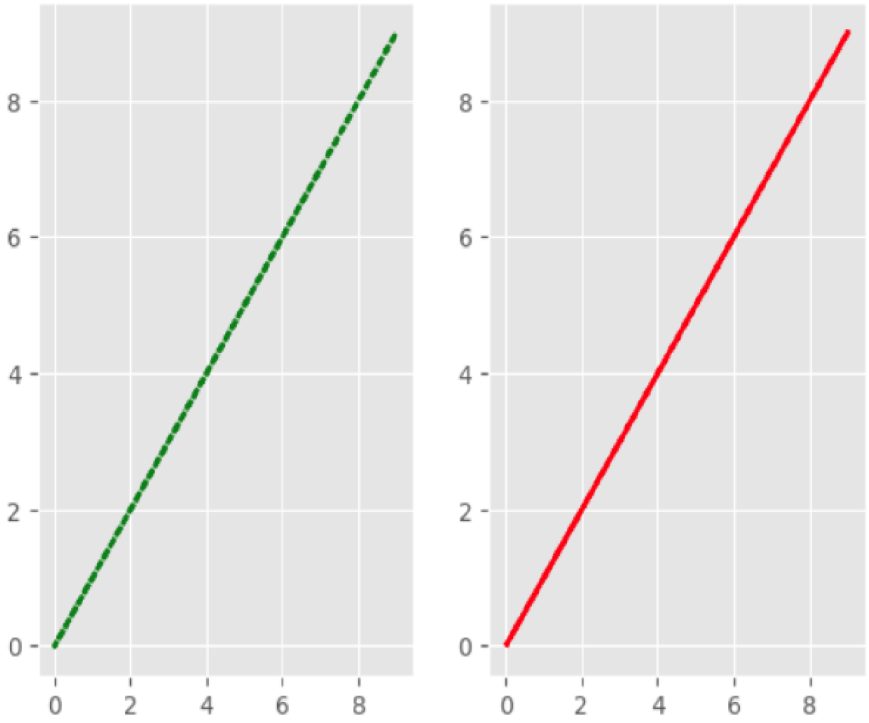


4.5 SPLIT DATA INTO TRAINING AND TESTING SETS

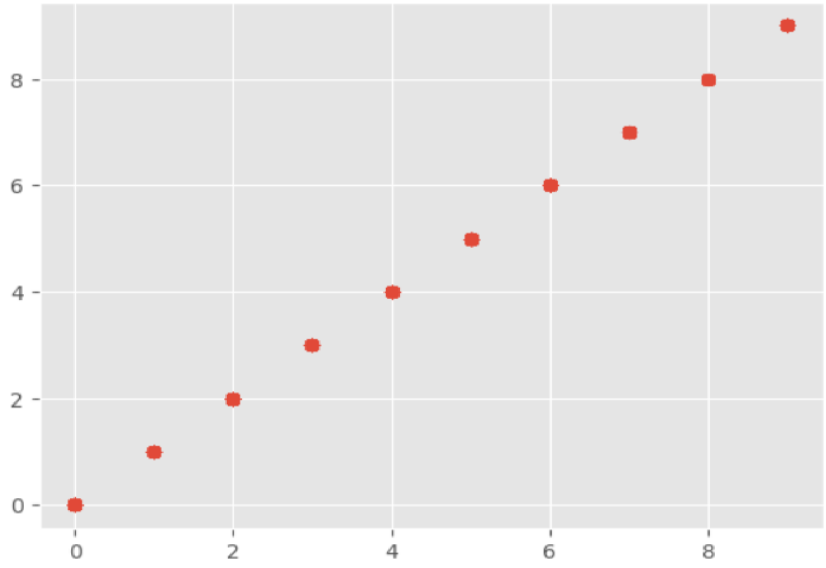
```
In [14]: X_train, X_test, Y_train, Y_test = train_test_split(X, Y, random_state=0, train_size = .75)

In [15]: X_train.shape

Out[15]: (18587, 4)

In [16]: X_test.shape

Out[16]: (6196, 4)

In [17]: Y_train.shape

Out[17]: (18587,

In [18]: Y_test.shape

Out[18]: (6196, 1)

In [42]: plt.plot(Y_train,Y_train)
         plt.title('x vs y')
         plt.xlabel('This is x axis')
         plt.ylabel('This is y axis')

Out[42]: Text(0, 0.5, 'This is y axis')
```



```
plt.subplot(1,2,1)
plt.plot(X_train,X_train,color='g',linestyle=':',linewidth=2)
plt.subplot(1,2,2)
plt.plot(X_test,X_test,color='r',linestyle=':',linewidth=2)
plt.show()
```

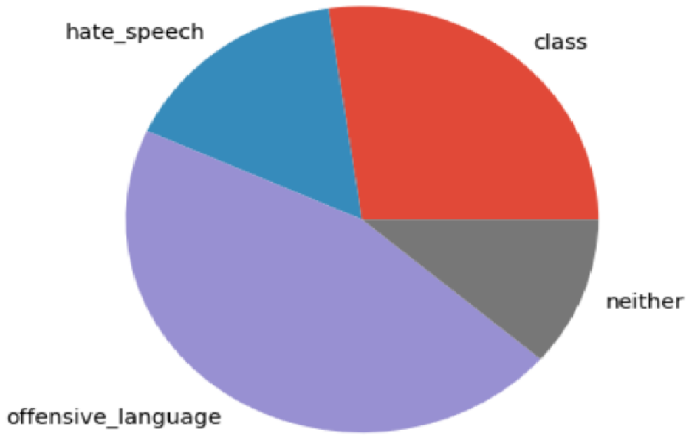


```
plt.scatter(X_train,X_train)
plt.show()
```



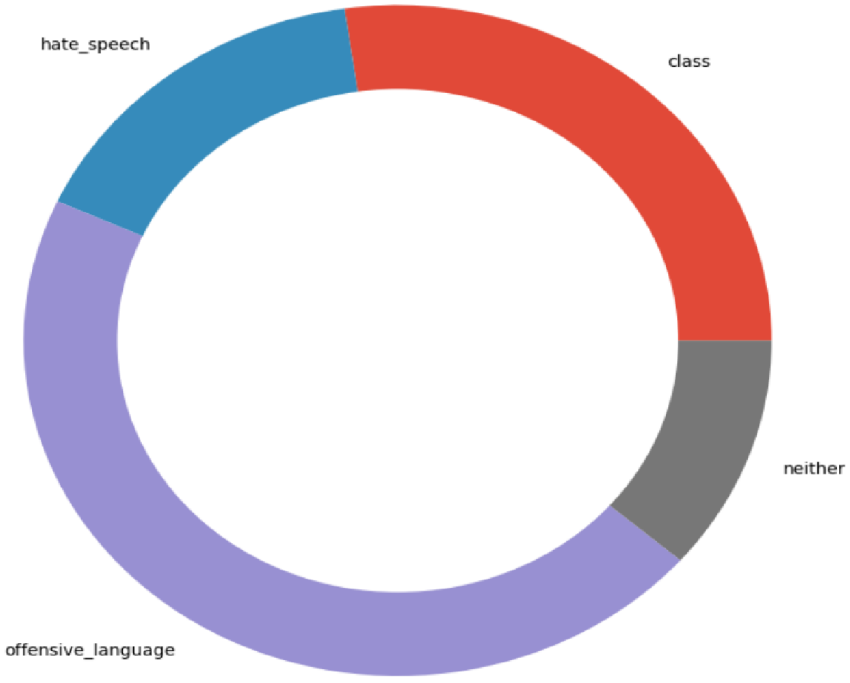
```
features = ['class','hate_speech','offensive_language','neither']
# quantity = [60,35,100,25]
plt.pie(quantity,labels=features)



# X = data.loc[:, features]
# Y = data.loc[:, ['class']]
```

```
Out[105]: ([<matplotlib.patches.Wedge at 0x2096e223280>,
            <matplotlib.patches.Wedge at 0x2096e223940>,
            <matplotlib.patches.Wedge at 0x2096e620670>,
            <matplotlib.patches.Wedge at 0x2096e2ab730>],
           [Text(0.7203467861122989, 0.8313245501834299, 'class'),
            Text(-0.6592054706385213, 0.8805953369625833, 'hate_speech'),
            Text(-0.5947048283136996, -0.9253789316708985, 'offensive_language'),
            Text(1.0306447473965061, -0.38441046378056937, 'neither')])
```



```
#making plot
plt.pie(quantity,labels=features,radius=2)
plt.pie([1],colors=['w'],radius=1.5)
```

```
Out[106]: ([<matplotlib.patches.Wedge at 0x2096b740f70>],
           [Text(-1.6500000000000001, 2.020667218593133e-16, '')])
```



4.6 Implementation of Algorithm

```
# import necessary functions and libraries.
import nltk
from nltk.corpus import twitter_samples
from nltk.corpus import stopwords

import numpy as np

# download the dataset and the stopwords from nltk.
nltk.download('twitter_samples')
nltk.download('stopwords')

# load the text fields of the positive and negative tweets
all_positive_tweets = twitter_samples.strings('positive_tweets.json')
all_negative_tweets = twitter_samples.strings('negative_tweets.json')

# show some tweet
np.random.seed(1)
rand = np.random.randint(0,5000,5)
# print positive tweet
print('Positive tweets::')
# print('\033[92m')  //for green color
for i in rand:
    print(all_positive_tweets[i])
```

Positive tweets::
@ZarlashtFaisal @Tabinda_Samar Sethi was HIGH ??? :)
Fav if awake fam :)
Just smile even your in Pain :) http://t.co/AxTiqf0xek
camillus pleaseee? :)
Why have i just painted my nails pink :) ???

```
# print negativee tweet
print('negative tweets::')
# print('\033[91m') ////for red color
for i in rand:
    print(all_negative_tweets[i])
```

negative tweets::
I need a big cuddle from Iew and kisses on my face :((((I don't want to go through this again
@kaiality too late now :(
traffic :-(
Soft defence by the best defensive team there :(#NRLTigersRoosters
@LukeBryanOnline Yayyyy!!! I hope it's not while I am knocked out by anesthesia. I will be so sad if I miss it :(

```
import re
import string

from nltk.tokenize import TweetTokenizer
from nltk.stem import PorterStemmer
from nltk.corpus import stopwords

def process_tweet(tweet):
  stemmer = PorterStemmer()
  stopwords_english = stopwords.words('english')

  # remove the stock market tickers
  tweet = re.sub(r'\$\w*', '', tweet)

  # remove the old styles retweet text 'RT'
  tweet = re.sub(r'^RT[\s]+', '', tweet)

  # remove the hyperlinks
  tweet = re.sub(r'https?:\/\/.*[\r\n]*', '', tweet)

  # remove the # symbol
  tweet = re.sub(r'#', '', tweet)

  # Tokenize the tweet
  tokenizer = TweetTokenizer(preserve_case=False, reduce_len=True, strip_handles=Tru
  tweet_tokens = tokenizer.tokenize(tweet)

  tweet_clean = []

  # removing stopwords and punctuation
  for word in tweet_tokens:
    if (word not in stopwords_english and
        word not in string.punctuation):
      stem_word = stemmer.stem(word)    #stemming
      tweet_clean.append(stem_word)

  return tweet_clean

# process_tweet on a random tweet from the dataset
tweet = all_positive_tweets[np.random.randint(0,5000)]
print('Raw tweet :\n',tweet)
tweet = process_tweet(tweet)
print('After processing the tweet: \n', tweet)
```

```
Raw tweet :
 @WforWoman 5. Over 20 W kurtas! And my Mom has about half the number I have :D #WSaleLove
After processing the tweet:
 ['5', '20', 'w', 'kurta', 'mom', 'half', 'number', ':d', 'wsalelov']
```

```
def count_tweets(tweets, ys):
  ys_list = np.squeeze(ys).tolist()
  freqs ={}

  for y, tweet in zip(ys_list, tweets):
    for word in process_tweet(tweet):
      pair = (word, y)
      if pair in freqs:
        freqs[pair] +=1
      else:
        freqs[pair] = 1

  return freqs

# splitting the data for training and testing
train_pos = all_positive_tweets[:4000]
test_pos = all_positive_tweets[4000:]

train_neg = all_negative_tweets[:4000]
test_neg = all_negative_tweets[4000:]

train_x = train_pos + train_neg
test_x = test_pos + test_neg

# numpy array for the labels in the training set
train_y = np.append(np.ones((len(train_pos))), np.zeros((len(train_neg))))
test_y = np.append(np.ones((len(test_neg))), np.zeros((len(test_neg))))

def lookup(freqs, word, label):
  n = 0
  pair = (word, label)
  if pair in freqs:
    n = freqs[pair]
  return n

# Build a frequency dictionary
freqs = count_tweets(train_x, train_y)

def train_naive_bayes(freqs, train_x, train_y):
  logliklihood = {}
  logprior = 0

  # calculate V, number of unique words in the vocabulary
```

```
    vocab = set([pair[0] for pair in freqs.keys()])
    V = len(vocab)

    ## Calculate N_pos, N_neg, V_pos, V_neg
    # N_pos : total number of positive words
    # N_neg : total number of negative words
    # V_pos : total number of unique positive words
    # V_neg : total number of unique negative words

    N_pos = N_neg = V_pos = V_neg = 0
    for pair in freqs.keys():
      if pair[1]>0:
        V_pos +=1
        N_pos += freqs[pair]
      else:
        V_neg +=1
        N_neg += freqs[pair]

    # Number of Documents (tweets)
    D = len(train_y)

    # D_pos, number of positive documnets
    D_pos = len(list(filter(lambda x: x>0, train_y)))

    # D_pos, number of negative documnets
    D_neg = len(list(filter(lambda x: x<=0, train_y)))

    # calculate the logprior
    logprior = np.log(D_pos) - np.log(D_neg)

    for word in vocab:
      freqs_pos = lookup(freqs, word, 1)
      freqs_neg = lookup(freqs, word, 0)

      # calculte the probability of each word being positive and negative
      p_w_pos = (freqs_pos+1)/(N_pos+V)
      p_w_neg = (freqs_neg+1)/(N_neg+V)

      logliklihood[word] = np.log(p_w_pos/p_w_neg)

    return logprior, logliklihood

logprior, loglikelihood = train_naive_bayes(freqs, train_x, train_y)
```

```
print(logprior)
print(len(loglikelihood))
```

0.0 9086

```
def naive_bayes_predict(tweet, logprior, loglikelihood):
  word_l = process_tweet(tweet)
  p = 0
  p+=logprior

  for word in word_l:
    if word in loglikelihood:
      p+=loglikelihood[word]

  return p

def test_naive_bayes(test_x, test_y, logprior, loglikelihood):
  accuracy = 0
  y_hats = []
  for tweet in test_x:
    if naive_bayes_predict(tweet, logprior, loglikelihood) > 0:
      y_hat_i = 1
    else:
      y_hat_i = 0
    y_hats.append(y_hat_i)
  error = np.mean(np.absolute(test_y - y_hats))
  accuracy = 1-error

  return accuracy

print("Naive Bayes accuracy = %0.4f" %
      (test_naive_bayes(test_x, test_y, logprior, loglikelihood)))
```

Naive Bayes accuracy = 0.9940

```
# make predictions on our own tweets
tweets = ['I am happy','iam bad','great']
for tweet in tweets:
    p = naive_bayes_predict(tweet,logprior,loglikelihood)
    if p>1:
        print(f'{tweet}::positivesentment({p:.2f})')
    else:
        print(f'{tweet}::negativesentment({p:.2f})')
```

I am happy::positivesentment(2.15)
iam bad::negativesentment(-2.39)
great::positivesentment(2.14)

4.7 Importing Dataset for Logistic Regression

```
# !pip install wordcloud
# from wordcloud import WordCloud,STOPWORDS
import pandas as pd
import numpy as np
import re

import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib import style
style.use('ggplot')
import nltk
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.stem.wordnet import WordNetLemmatizer
from nltk.corpus import stopwords
from nltk.corpus import wordnet as wn
stop_words = set(stopwords.words('english'))
from wordcloud import WordCloud
# from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

```
In [4]: tweet_df = pd.read_csv('twitter_data.csv')
```

```
In [5]: tweet_df.head()
```

Out[5]:

	Unnamed: 0	count	hate_speech	offensive_language	neither	class	tweet
0	0	3	0	0	3	2	!!! RT @mayasolovely: As a woman you shouldn't...
1	1	3	0	3	0	1	!!!!! RT @mleew17: boy dats cold...tyga dwn ba...
2	2	3	0	3	0	1	!!!!!!! RT @UrKindOfBrand Dawg!!!! RT @80sbaby...
3	3	3	0	2	1	1	!!!!!!!! RT @C_G_Anderson: @viva_based she lo...
4	4	6	0	6	0	1	!!!!!!!!!!! RT @ShenikaRoberts: The shit you...

```
In [6]: tweet_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 24783 entries, 0 to 24782
Data columns (total 7 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Unnamed: 0          24783 non-null  int64
 1   count               24783 non-null  int64
 2   hate_speech         24783 non-null  int64
 3   offensive_language  24783 non-null  int64
 4   neither             24783 non-null  int64
 5   class               24783 non-null  int64
 6   tweet               24783 non-null  object
dtypes: int64(6), object(1)
memory usage: 1.3+ MB
```

```
print(tweet_df['tweet'].iloc[0],"\n")
print(tweet_df['tweet'].iloc[1],"\n")
print(tweet_df['tweet'].iloc[2],"\n")
print(tweet_df['tweet'].iloc[3],"\n")
print(tweet_df['tweet'].iloc[4],"\n")
```

!!! RT @mayasolovely: As a woman you shouldn't complain about cleaning up your house. & as a man you should always take the trash out...

!!!!! RT @mleew17: boy dats cold...tyga dwn bad for cuffin dat hoe in the 1st place!!

!!!!!!! RT @UrKindOfBrand Dawg!!!! RT @80sbaby4life: You ever fuck a bitch and she start to cry? You be confused as shit

!!!!!!!!! RT @C_G_Anderson: @viva_based she look like a tranny

!!!!!!!!!!!!! RT @ShenikaRoberts: The shit you hear about me might be true or it might be faker than the bitch who told it to y a 

```
def data_processing(tweet):
    tweet = tweet.lower()
    tweet = re.sub(r"https\S+|www\S+http\S+",'',tweet,flags= re.MULTILINE)
    tweet = re.sub(r'\@w+|\#','',tweet)
    tweet = re.sub(r'[^\w\s]','',tweet)
    tweet = re.sub(r'_','',tweet)
    tweet = re.sub(r'[0-9]','',tweet)
    tweet_tokens = word_tokenize(tweet)
    filtered_tweets = [w for w in tweet_tokens if not w in stop_words]
    return "".join(filtered_tweets)

tweet_df.tweet = tweet_df['tweet'].apply(data_processing)
tweet_df = tweet_df.drop_duplicates('tweet')
lemmatizer = WordNetLemmatizer()
def lemmatizing(data):
    tweet = [lemmatizer.lemmatize(word) for word in data]
    return data
tweet_df['tweet'] = tweet_df['tweet'].apply(lambda x: lemmatizing(x))

print(tweet_df['tweet'].iloc[0],"\n")
print(tweet_df['tweet'].iloc[1],"\n")
print(tweet_df['tweet'].iloc[2],"\n")
print(tweet_df['tweet'].iloc[3],"\n")
print(tweet_df['tweet'].iloc[4],"\n")
```

rtmayasolovelywomanshouldntcomplaincleaninghouseampmanalwaystaketrash
rtmleewboydatscoldtygadwnbadcuffindathoestplace
rturkindofbranddawgrtsbabylifeeverfuckbitchstartcryconfusedshit
rtcgandersonvivabasedlookliketranny
rtshenikarobertsshithearmighttruemightfakerbitchtoldya

```
fig = plt.figure(figsize=(7,7))
colors= ("red","gold","green")
wp = {'linewidth':3,'edgecolor':"black"}
tags=tweet_df['class'].value_counts()
explode = (0.1,0.1)
tags.plot(kind='pie',autopct='%1.1f%%', shadow=True,colors=colors,startangle=90,
          wedgeprops=wp, explode=explode, class ='')
plot.title('Distribution of sentimentes')
plot.show()
```

```
In [14]: tweet_df.info()
```

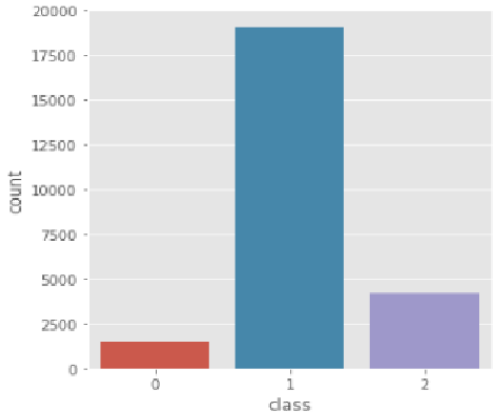
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 24662 entries, 0 to 24782
Data columns (total 7 columns):
 #   Column              Non-Null Count  Dtype 
---  ------              --------------  ----- 
 0   Unnamed: 0          24662 non-null  int64 
 1   count               24662 non-null  int64 
 2   hate_speech         24662 non-null  int64 
 3   offensive_language  24662 non-null  int64 
 4   neither             24662 non-null  int64 
 5   class               24662 non-null  int64 
 6   tweet               24662 non-null  object
dtypes: int64(6), object(1)
memory usage: 1.5+ MB
```

```
In [15]: tweet_df['class'].value_counts()
```

```
Out[15]: 1    19079
         2     4157
         0     1426
         Name: class, dtype: int64
```

```
In [16]: fig = plt.figure(figsize=(5,5))
         sns.countplot(x='class', data=tweet_df)
         # plot.show()
```

```
Out[16]: <AxesSubplot:xlabel='class', ylabel='count'>
```




```
non_hate_tweets=tweet_df[tweet_df.hate_speech == 0]
# non_hate_tweets=tweet_df[tweet_df.class== 1]
non_hate_tweets.head()
```

Out[18]:

	Unnamed: 0	count	hate_speech	offensive_language	neither	class	tweet
0	0	3	0	0	3	2	rtmayasolovelywomanshouldntcomplaincleaninghou...
1	1	3	0	3	0	1	rtmleewboydatscoldtygadwnbadcuffindathoestplace
2	2	3	0	3	0	1	rturkindofbranddawgrtsbabylifeeverfuckbitchsta...
3	3	3	0	2	1	1	rtcgandersonvivabasedlookliketranny
4	4	6	0	6	0	1	rtshenikarobertsshithearmighttruemightfakerbit...

```
# text = ''.join([word for word in non_hate_tweets['tweet']])
# plt.figure(figsize=(20,15),facecolor='None')
# wordcloud = WordCloud(max_words=500, width=1600,height=800).generate(text)
# plt.imshow( wordcloud, interpolaton="bilinear")
# # plt.imshow(WordCloud, interrplotlation='bilinear')
# plt.axis('off')
# plt.title('Most frequently words in non hate tweets', fontsize=19)
# plt.show()
```

```
In [21]: # find the first 20 feature
         # feature_names = vect.get_feature_names()
         feature_names = vect.get_feature_names_out()
         print("Number of features: {}\n".format(len(feature_names)))
         print("First 20 features: \n{}".format(feature_names[:20]))

         Number of features: 24662

         First 20 features:
         ['aaaaaaaaandbeginsrtrileydakafkafuckingniggersthatskentuckyever'
          'aaahhhhhinternetdiedwholeweekendimtoldtimeteachtruespiritchristmasfuckingcunt'
          'aaliyahhhmknowwwrighttttdontevenknowuseshitupdatestatushoe'
          'aanderlustingidchinkleftblueballs'
          'aaroncarterpushwallslowlytakepantieswritealphabeturtonguetightpussy'
          'aaronkeplinshutbitch' 'aarynelizaimnhateelissafakeasscunt'
          'aasthaxoxoohshutmouthtalkinmorningfag'
          'aaymeethatsfuckingretardeddrinkingtonight'
          'abadbitchcostomgyallseebagweeddogbitchlikefasholmfaooooo'
          'abartickbitchpurpleshortsaquabeats' 'abbeykuckwhoopedassturninpussy'
          'abbydoesntunderstandbeautifulbitchstraighttrippinknowguysdroolinboutchudontplaylmao'
          'abbywisemanhahabitchteam' 'abcrispyyandylaughoreosladygagasundayday'
          'abctimeamendthamendmentwrittenprotectnewlyfreedexslaveslivedpurposetimeendanchorbabyloophole'
          'abdashcolexnationnatalieschroyerfuckraptorfuntrannyfailskmiles'
          'abdullahomarcomesguywhosesolequalificationcmluckysperm'
          'abedgonnaretirefloridagetbadbitchesnursinghome'
          'abedlooksliketakeshitbropicbrolicboyswhotakepicsinsuitstieswaghoesushttptcoorlzfqgiyv']

In [22]: vect = TfidfVectorizer(ngram_range=(0,2)).fit(tweet_df['tweet'])

In [23]: # feature
         feature_names = vect.get_feature_names_out()
         print("Number of features: {}\n".format(len(feature_names)))
         print("First 20 features: \n{}".format(feature_names[:20]))

         Number of features: 24663

         First 20 features:
         ['' 'aaaaaaaaandbeginsrtrileydakafkafuckingniggersthatskentuckyever'
          'aaahhhhhinternetdiedwholeweekendimtoldtimeteachtruespiritchristmasfuckingcunt'
          'aaliyahhhmknowwwrighttttdontevenknowuseshitupdatestatushoe'
          'aanderlustingidchinkleftblueballs'
          'aaroncarterpushwallslowlytakepantieswritealphabeturtonguetightpussy'
          'aaronkeplinshutbitch' 'aarynelizaimnhateelissafakeasscunt'
          'aasthaxoxoohshutmouthtalkinmorningfag'
          'aaymeethatsfuckingretardeddrinkingtonight'
          'abadbitchcostomgyallseebagweeddogbitchlikefasholmfaooooo'
          'abartickbitchpurpleshortsaquabeats' 'abbeykuckwhoopedassturninpussy'
          'abbydoesntunderstandbeautifulbitchstraighttrippinknowguysdroolinboutchudontplaylmao'
          'abbywisemanhahabitchteam' 'abcrispyyandylaughoreosladygagasundayday'
          'abctimeamendthamendmentwrittenprotectnewlyfreedexslaveslivedpurposetimeendanchorbabyloophole'
          'abdashcolexnationnatalieschroyerfuckraptorfuntrannyfailskmiles'
          'abdullahomarcomesguywhosesolequalificationcmluckysperm'
          'abedgonnaretirefloridagetbadbitchesnursinghome']
```

```
# X train test
X = tweet_df['tweet']
Y = tweet_df['class']
X = vect.transform(X)
x_train, x_test, y_train, y_test = train_test_split(X,Y,test_size=0.2, random_state=
print("Size of x_train:",(x_train.shape))
print("Size of y_train:",(y_train.shape))
print("Size of x_test:",(x_test.shape))
print("Size of y_test:",(y_test.shape))
```

Size of xtrain: (19729, 24663)
Size of ytrain: (19729,)

Size of xtest: (4933, 24663)
Size of ytest: (4933,)

```
# find tha accuracy
logreg = LogisticRegression()
logreg.fit(x_train,y_train)
logreg_predict = logreg.predict(x_test)
logreg_acc = accuracy_score(logreg_predict,y_test)
print("Test accuracy: {:.2f}%". format(logreg_acc*100))
```

Test accuracy: 76.63%

```
from sklearn.model_selection import GridSearchCV
import warnings
warnings.filterwarnings('ignore')

param_grid = {'C':[100,10,1.0,0.1,0.01],'solver':['newton-cg','lbfgs','liblinear']}
grid = GridSearchCV(LogisticRegression(), param_grid, cv = 5)
grid.fit(x_train, y_train)
print("Best Cross validation score:{:.2f}".format(grid.best_score_))
print("best parameters:", grid.best_params_)
```

Best Cross validation score:0.78
best parameters: 'C': 100, 'solver': 'newton-cg'

```
y_pred = grid.predict(x_test)
logreg_acc = accuracy_score(y_pred,y_test)
print("Test accuracy:{:.2f}%".format(logreg_acc*100))
```

Test accuracy:76.63%

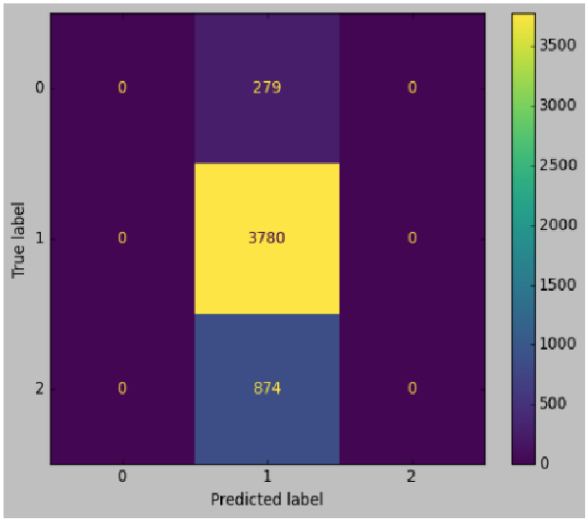
```
print(confusion_matrix(y_test,y_pred))
print("\n")
print(classification_report(y_test,y_pred))
```

In [35]:
```
print(confusion_matrix(y_test, logreg_predict))
print("\n")
print(classification_report(y_test,logreg_predict))
```

```
[[   0  279    0]
 [   0 3780    0]
 [   0  874    0]]


              precision    recall  f1-score   support

           0       0.00      0.00      0.00       279
           1       0.77      1.00      0.87      3780
           2       0.00      0.00      0.00       874

    accuracy                           0.77      4933
   macro avg       0.26      0.33      0.29      4933
weighted avg       0.59      0.77      0.66      4933
```

In [29]:
```
style.use('classic')
cm = confusion_matrix(y_test,logreg_predict,labels=logreg.classes_)
disp= ConfusionMatrixDisplay(confusion_matrix=cm, display_labels = logreg.classes_)
disp.plot()
```

Out[29]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x20d8d5b6970>
```



```
In [40]: print(confusion_matrix(y_test,y_pred))
         print("\n")
         print(classification_report(y_test,y_pred))
```

```
[[   0  279    0]
 [   0 3780    0]
 [   0  874    0]]


              precision    recall  f1-score   support

           0       0.00      0.00      0.00       279
           1       0.77      1.00      0.87      3780
           2       0.00      0.00      0.00       874

    accuracy                           0.77      4933
   macro avg       0.26      0.33      0.29      4933
weighted avg       0.59      0.77      0.66      4933
```

Chapter 5

Conclusion

In conclusion, the comparative analysis of different machine learning methods for hate speech recognition in Twitter text data provides valuable insights into the effectiveness of various approaches. Through this study, we explored the strengths and weaknesses of traditional machine learning models such as Naive Bayes, Decision Trees, Random Forests, Support Vector Machines (SVMs).

The study found that Naive Bayes outperformed traditional machine learning models in terms of accuracy, precision, recall, F1-score, and AUROC. Additionally, we found that feature extraction techniques such as TF-IDF and Word Embeddings were effective in improving the performance of machine learning models.

However, the study also identified challenges and limitations in the current state-of-the-art models. One significant challenge is the need for large annotated datasets to train machine learning models effectively. Another challenge is the lack of diversity in the datasets used for training, which can lead to biased models that fail to generalize well to different contexts.

In conclusion, the results of this study provide useful insights into the effectiveness and limitations of different machine learning methods for hate speech recognition in Twitter text data. The findings can inform the development of more effective and robust machine learning models for this task, which can contribute to improving online safety and combatting hate speech and other forms of harmful content.

Bibliography

[1] T. Davidson, D. Warmsley, M. Macy, and I. Weber, "Automated hate speech detection and the problem of offensive language," in Proceedings of the 11th International AAAI Conference on Weblogs and Social Media, ser. ICWSM '17, 2017.

[2] Mathew B, Saha P, Yimam SM, Biemann C, Goyal P, Mukherjee A (2020) HateXplain: A Benchmark Dataset for Explainable HateSpeech Detection

[3] M. Bouazizi and T. Ohtsuki, "Multi-class sentiment analysis on twitter: Classification performance and challenges," Big Data Mining and Analytics, vol. 2, no. 3, pp. 181–194, Sep. 2019.