Hate Speech Recognizer

Yashita Vajpayee
Masters in Data science
Stevens Institute of Technology
Hoboken, USA
yvajpaye@stevens.edu

Sharven Subhash Rane
Masters in Data science
Stevens Institute of Technology
Hoboken, USA
srane8@stevens.edu

Kalyan Varma Patchamatla
Masters in Data science
Stevens Institute of Technology
Hoboken, USA
kpatcham@stevens.edu

***Abstract*—The aim of this project is to detect hate speech from twitter data using logistic regression, Random forest, Naive Bayes, SVM and XGBoost methods and report a comparison between results.**

I. Introduction

Online communication has evolved as a result of social networking. While the capacity of regular individuals to instantly reach thousands of others clearly has benefits, drawbacks like polarization due to echo chambers have emerged. Due of people's interconnectedness, malevolent actors are able to spread hate speech and other offensive materials and sway public opinion. There are different definitions of hate speech. Despite the fact that there have always been prohibitions against harmful speech, the word was first used in the US in 1989 to address the issue of "destructive racist speech" that was still permitted there. Texts that "spread, encourage, advocate or excuse racial hatred, xenophobia, antisemitism or other forms of hatred based on intolerance" are considered "hate speech," according to the European Union's definition of the term from 1997.

Although it is not easy to distinguish between hate speech and merely offensive or upsetting content, it is possible. Hate speech detection is frequently framed as a categorization issue. In order to create a discriminative function that can distinguish between hate speech and non-hate speech, common machine learning algorithms are applied. To the best of our knowledge, despite the fact that a number of hate speech detection algorithms have been reported in the scientific literature, there hasn't yet been a thorough empirical evaluation of real implementations of suggested models and datasets. In five papers, five current model architectures are examined. We repeated the experiments while fine-tuning a pre-trained language model for the categorization tasks. Previous research has only considered so-called naive adversaries, who make no effort to evade detection. We compare the efficacy of various machine learning methods at identifying hate speech.

II. Related Works

. We chose an SVM strategy from paper [1] for our machine learning model because it offers performance that is almost at the cutting edge while being simpler and producing decisions that are easier to understand than neural methods. Despite mentioning research that cover both hate speech and other unpleasant words, paper [2] focuses on hate speech and hate speech datasets. Paper [3] demonstrates how to computerize the classification of hate speech components in text, which can then be used to speech recognition. utilizing the method of logistic regression.In paper [4], the researcher uses a dataset of tweets that identify hate speech and abusive language. This dataset is classified using the Random Forest approach, and the accuracy of its results vs those of AdaBoost and Neural Network are compared. The research [5] suggests using XGBoost classifier in conjunction with natural language processing to capture the context and semantics of hate speech. The research [6] examines the overall accuracy, precisions, and recall values of two supervised machine learning algorithms: K-Nearest Neighbour (K-NN) and Naive Bayes.By efficiently selecting the appropriate hyper-parameter configurations, this survey report [7] will assist industrial users, data analysts, and researchers in developing machine learning models more effectively. Two primary justifications for preparing data are I issues with the data and (ii) getting ready for data analysis, both of which are covered in length in the work [8]. 14 different strategies are covered in all. The end of the article provides two examples of data preparation applications from two of the most data-rich domains.

III. Our Solution

In this project, we are going to do sentiment analysis and hate speech recognition. The data from Twitter, a popular microblogging social media platform for sharing short digital content, was used for this project.

A. Description of Dataset

Dataset comprises of three columns : id, label and tweet, each with10,575 row entries. "Id" is of integer datatype which gives id assigned to a particular tweet and label. Given a training sample of tweets and labels, label '1' denotes the hate tweet and label '0' denotes the non-hate tweet. Label are of integer datatype and tweets are of object type. Full tweet texts are provided with their labels for training data. Mentioned users' username is replaced with @user.

B. Preprocessing

For better analysis, we originally began by processing the data using natural language processing. There has been developed a preprocessing function for data processing (tweet) that performs the following functions:

1. Lowercase letters for all uppercase letters.
2. Regex-based removal of URLs and hashtags
3. Eliminating all punctuation.
4. By tokenizing the tweets and subsequently filtering them, all stop words in the English language are eliminated.
5. Eliminating duplicate records

Lemmatization : We then proceeded to perform lemmatization which switches and groups any word to its base root mode. We used WordNetLemmatizer() from nltk library to perform this task. After all the above pre-processing, the number of rows is reduced to 9,949 from 10,575.

C. *EDA*

To explore the data, we observed the distribution of sentiments by plotting a pie-chart showing the percentage of both the labels (hate tweet and non-hate tweet).

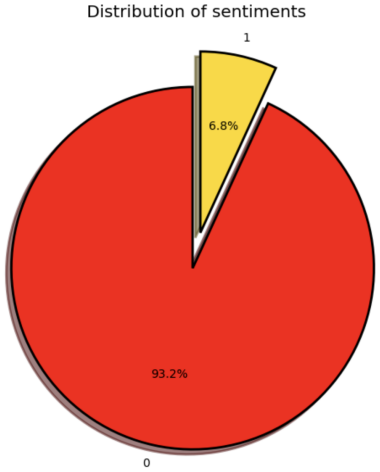

Fig. 1: Piechart showing distribution of labels

As seen from the chart, yellow color specifies Hate speech which is 6.8% while Non-hate speech in red color occupies 93.2%

1) ***WordCloud****:* A wordcloud visualizes the text data according to the frequency of each word in it. We visualized two different word clouds, each for hate tweets and non-hate tweets in order to take a look on which words are more frequently used in each type of tweets. Following is the Wordcloud of non-hate tweets and hate tweets respectively:

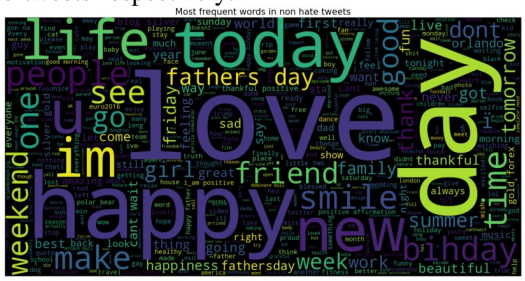

Fig. 2: Non-hate tweets word cloud

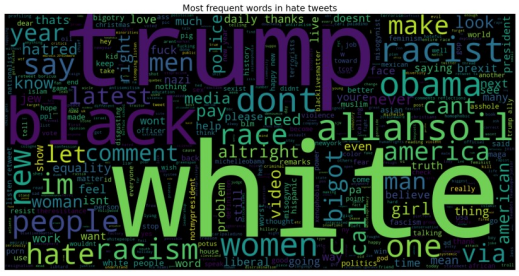

Fig. 3: Hate tweets word cloud

2) ***Vectorization:****:* We used Count vectorization(Tf-idf) that measures the frequency of a word in a text against its overall frequency in the corpus. It gives a higher relevance scores to words that occur in fewer documents within the corpus. In our code, we created a bi-gram and tri-gram language model and then used TfidfVectorizer() from sklearn.feature_extraction.text library to perform the vectorization.

D. *Model Building*

Preparing the data - We divided the data into X and Y where X is the vector transformed tweet and Y is corresponding label of that tweet. The data is then splitted into 20% test data and 80% training data using train_test_split() from sklearn.model_selection library.

Since our data is categorical and we working on a classification problem, We used Classification Machine learning models as follows:

1) ***Logistic Regression****:* Logistic regression is a classification model used in categorical data. Predictive analytics and categorization frequently make use of this kind of statistical model, also referred to as a logit model. Based on a given dataset of independent variables, logistic regression calculates the likelihood that an event will occur, such as voting or not voting. Given that the result is a probability, the dependent variable's range is 0 to 1. In logistic regression, the odds—that is, the probability of success divided by the probability of failure are transformed using the logit formula. The following formulas are used to represent this logistic function, which is sometimes referred to as the log odds or the natural logarithm of odds:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Since our data is categorical, we started by implementing Logistic regression on our data. In our code, We used LogisticRegression() from sklearn.linear_model library.

2) ***Naive-Bayes Classification****:* Naive-bayes models works on the principle of Bayes' theorem with an assumption of independence among the predictors i.e. presence of a particular feature in a class is unrelated to the presence of any other feature. Naive- Bayes creates a likelihood table of the probabilities and then use Naive-Bayesian equation to calculate posterior probability of each class. The class with highest posterior probability is the outcome of prediction. Naive Bayes theorem can be given by:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

In this equation, P(A—B) represents the conditional probability of event A occurring given that event B has occurred. P(B—A) represents the conditional probability of event B occurring given that event A has occurred. P(A) represents the probability of event A occurring, and P(B) represents the probability of event B occurring.] Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods. Advantages of Naive Bayes are : Predicting the class

of test data set is simple and quick. Additionally, it excels at multi-class prediction. A Naive Bayes classifier performs better when the assumption of independence is true than other models, such as logistic regression, and requires fewer training data. Compared to a numerical variable, it performs well with categorical input variables (s). It is assumed that numerical variables have a normal distribution (bell curve, which is a strong assumption).

*3) **SVM**:* In SVM, each data point is plotted in n-dimensional space (n = number of features) and then we perform classification by identifying a hyperplane which separates the two classes best. Support Vectors are the data points of both the classes lying closest to the hyperplane. We try to maximize the marginal distance between the Support vectors of both the classes in order to maximize our accuracy. Types of Support Vector Machine: Linear SVM When the data is perfectly linearly separable only then we can use Linear SVM. Perfectly linearly separable means that the data points can be classified into 2 classes by using a single straight line(if 2D). Non-Linear SVM When the data is not linearly separable then we can use Non-Linear SVM, which means when the data points cannot be separated into 2 classes by using a straight line (if 2D) then we use some advanced techniques like kernel tricks to classify them. In most real-world applications we do not find linearly separable datapoints hence we use kernel trick to solve them.In our model, there is only one feature "tweets" to classify into two classes. We used LinearSVC() from sklearn.svm to model the SVM. Support Vectors: These are the points that are closest to the hyperplane. A separating line will be defined with the help of these data points. Margin: it is the distance between the hyperplane and the observations closest to the hyperplane (support vectors). In SVM large margin is considered a good margin. There are two types of margins hard margin and soft margin. I will talk more about these two in the later section.

*4) **Random forest**:* Random forests or random decision forests is an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time. For classification tasks, the output of the random forest is the class selected by most trees. For regression tasks, the mean or average prediction of the individual trees is returned.Random decision forests correct for decision trees' habit of overfitting to their training set.Random forests generally outperform decision trees, but their accuracy is lower than gradient boosted trees.However, data characteristics can affect their performance. In random forest n number of random samples are taken from the dataset having k number of records and then individual decision trees are constructed for each sample. Each decision tree gives an output and then the final output is considered by majority voting for classification. Since random forests are created from subsets of data and final output is result of majority ranking, Random forest doesn't experience the problem of overfitting. We used RandomForestClassifier() from sklearn.ensemble library.

*5) **XGBoost Classifier**:* XGBoost, which stands for Extreme Gradient Boosting, is a scalable, distributed gradient-boosted decision tree (GBDT) machine learning library. It provides parallel tree boosting and is the leading machine learning library for regression, classification, and ranking problems.Like other gradient boosting methods, XGBoost works by building a model in the form of an ensemble of weak learners, where each weak learner is a decision tree. However, XGBoost introduces several innovations to the gradient boosting algorithm that make it more efficient and effective. For example, XGBoost uses a more sophisticated tree splitting algorithm and adds a regularization term to the loss function to prevent overfitting. XGBoost is widely used in practice because it often outperforms other machine learning algorithms on a variety of tasks, and it is relatively fast to train. It is also resistant to overfitting, which makes it a good choice for working with large datasets. The following are only a few of the many advantages and qualities of XGBoost: There is a sizable and expanding list of data scientists working on the XGBoost open source project from across the world. Use in a variety of applications, including as resolving issues with categorization, ranking, regression, and user-defined prediction challenges a highly portable library that is now available for use with OS X, Windows, and Linux Support for AWS, Azure, Yarn clusters, and other ecosystems through cloud integration active production utilization across a number of companies and vertical market segments A library that is effective, adaptable, and portable since it was created from the ground up

IV. Hyperparameter Tuning for optimization

Hyperparameter tuning is the process of finding the optimal values for the hyperparameters of a machine learning model. Hyperparameters are the parameters of a model that are not learned from the data during training, but rather are set by the practitioner. They include things like the learning rate, the regularization strength, and the number of hidden units in a neural network. The goal of hyperparameter tuning is to find the values for the hyperparameters that result in the best performance of the model on a given task. This can be done through a process called model selection, which involves training multiple models with different combinations of hyperparameters and selecting the one that performs the best. There are several approaches you can take to perform hyperparameter tuning, including manual tuning, grid search, and random search. The specific approach you choose will depend on your specific needs and the characteristics of your data. Hyperparameter tuning is an important step in the machine learning process, as it can significantly improve the performance of a model and help you get the most out of your data. We used Grid Search for all the models used. Hyperparameter tuning first involves loading the features and labels for the data and splits the data into training and test sets. We then set up a parameter grid containing the hyperparameters to be tuned and the values to try for each parameter. We then create a GridSearchCV object, which performs cross-validation over the parameter grid to find the best combination of hyperparameters. Finally, we fit the grid search object to the training data, print the best

hyperparameters and the best score, and make predictions on the test set using the best model. Then evaluates the model's performance on the test set using the accuracy_score function.

A. *Logistic Regression*

In a logistic regression model, the main parameter that we tuned is the regularization strength, which is represented by the hyperparameter "C". The regularization strength determines how much the model should penalize large coefficients, which can help prevent overfitting. It is a good idea to use cross-validation to tune this parameter and find the best combination for your specific dataset. This will help you find the optimal balance between model accuracy and training time.

B. *Naive-Bayes Classification:*

In a Naive Bayes classifier, there are typically two main parameters that we tuned:
1.The smoothing parameter: This parameter is used to smooth the probabilities estimated from the training data. It is used to prevent the model from overfitting by "smoothing" the probabilities of the class labels.
2.The prior probabilities of the class labels: These are the probabilities of each class label in the training data. You can specify your own prior probabilities if you have some prior knowledge about the distribution of the class labels in your data, or you can let the model estimate the prior probabilities from the training data.
There are also other parameters that you can tune in a Naive Bayes classifier, such as the type of distribution used to estimate the probabilities of the features (e.g. Gaussian, multinomial, etc.), but the smoothing parameter and the prior probabilities are the most important ones.

C. *SVM*

Linear Support Vector Classification (LinearSVC) is a linear model for classification tasks that is similar to SVC with a linear kernel. The main hyperparameters that you can tune in a LinearSVC model are:
1.Penalty parameter C: This controls the trade-off between the smooth decision boundary and the number of support vectors. A large value of C means that the model is more prone to overfitting, while a small value of C means that the model is more prone to underfitting.
2.Loss function: This determines the penalty for misclassifying a data point. The most common loss functions for LinearSVC are the hinge loss and the squared hinge loss.
3.Multi-class strategy: This determines how the model handles multiple classes. The most common strategies are "ovr" (one-versus-rest) and "crammer-singer" (multinomial).

D. *Random Forest*

There are several parameters that you can tune in a random forest model to try to improve its performance. Some of the main parameters that we considered tuning include the following:
1. Number of trees: The number of trees in the random forest determines the complexity of the model and can affect the model's ability to generalize to new data. A larger number of trees can result in a more accurate model, but it may also increase the training time and risk overfitting.
2. Maximum depth of trees: The maximum depth of each tree in the random forest determines how deep the tree can grow before it is split. A deeper tree can capture more information about the data, but it may also increase the risk of overfitting.
3.Minimum samples per leaf: The minimum number of samples that are required to be at a leaf node determines the number of splits that are made in the tree. A larger minimum number of samples per leaf can reduce the risk of overfitting, but it may also reduce the model's ability to capture subtle patterns in the data.

E. *XGBoost Classifier*

In the XGBoost XGBClassifier model, we included the below parameters for tuning:
1.eta (learning rate): Controls the step size at which the algorithm moves in the direction of the gradient, in an attempt to minimize the loss function.
2.max_depth: Controls the maximum depth of the trees in the model. Deeper trees can capture more complex patterns in the data, but can also be more prone to overfitting.
3.subsample: Controls the fraction of observations to be used for each tree. Using a smaller subsample can help prevent overfitting, but may also result in a less accurate model.
4.colsample_bytree: Controls the fraction of features to be used for each tree. Using a smaller subset of features can help prevent overfitting, but may also result in a less accurate model.
In general, it is a good idea to use cross-validation to tune these parameters and find the best combination for a specific dataset. This will help you find the optimal balance between model accuracy and training time.

V. Results and Comparison

1) ***Logistic Regression****:* After fitting the x_train and y_train in the model, we get the accuracy for predicting y_test from x_test as **92.16%** but after hyperparameter tuning the result is **93.33%** The confusion matrix for the same can be seen as:

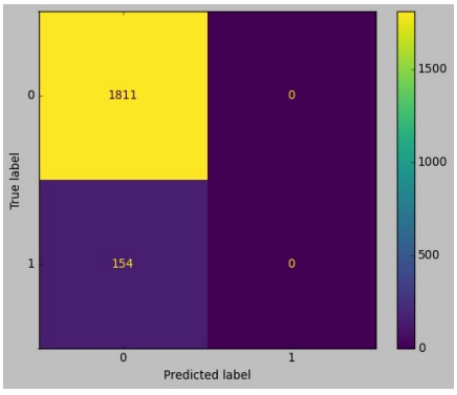

Fig. 4: Confusion matrix

2) ***Naive-Bayes Classification****:* After fitting the x_train and y_train in the model, we get the accuracy for predicting y_test from x_test as **90.63%** but after hyperparameter tuning the

result is **92.93%** The confusion matrix for the same can be seen as:

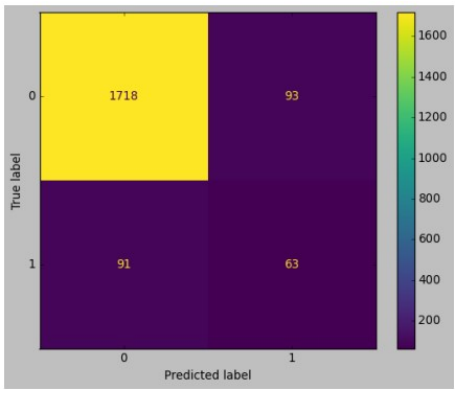

Fig. 5: Confusion matrix

*3) **SVM**:* After fitting the x_train and y_train in the model, we get the accuracy for predicting y_test from x_test as **93.23%** but after hyperparameter tuning the result is **93.93%** The confusion matrix for the same can be seen as:

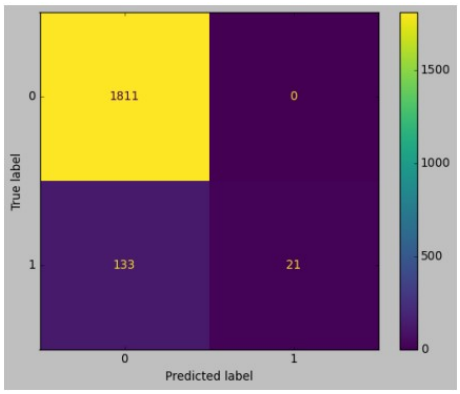

Fig. 6: Confusion matrix

*4) **Random Forest**:* After fitting the x_train and y_train in the model, we get the accuracy for predicting y_test from x_test as **93.33%** but after hyperparameter tuning the result is **94.4%** The confusion matrix for the same can be seen as:

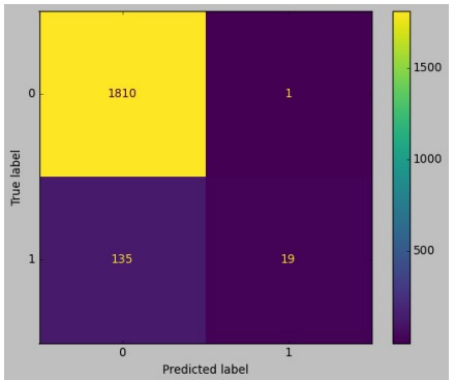

Fig. 7: Confusion matrix

*5) **XGBoost Classifier**:* After fitting the x_train and y_train in the model, we get the accuracy for predicting y_test from x_test as **93.87%** but after hyperparameter tuning the result is **94.12%** The confusion matrix for the same can be seen as:

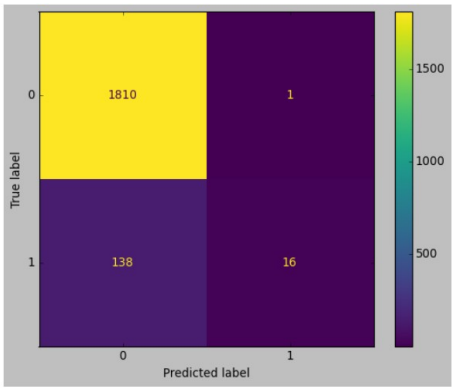

Fig. 8: Confusion matrix

The comparison of accuracy of all the five models before and after hyperparameter tuning can be seen in the following visualization:

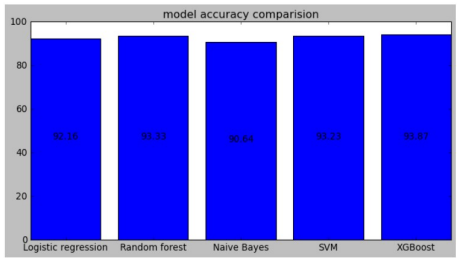

Fig. 9: General accuracies

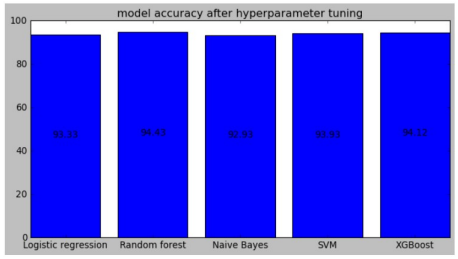

Fig. 10: Accuracies after hyperparameter tuning

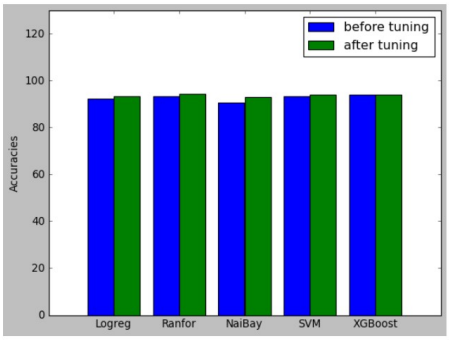

Fig. 11: Comparison of the accuracies

VI. Future Directions

Future plans call for the implementation of ML techniques such RNN, LSTM, BERT, Deep Neural Networks, etc. Additionally, we want to work with a bigger dataset that includes multiple languages. Additionally, we want to push ourselves by employing unsupervised learning algorithms to work with unlabeled data. In order to address the current problems with hate speech, we want to implement this project on any social media site. By counting the number of hateful terms in a tweet, we can rate it and automatically prevent hateful tweets from being published.

VII. Conclusion

The extensive experiments and analysis show that XGBoost and Random forest are good algorithms for hate speech detection. Also implementing hyperparameter tuning helped in improving the accuracies of all the models even though the computation time for implementing hyperparameter tuning is more. More efficient algorithms can be used alongside deep learning and neural networks techniques to further enhance the model. This is to provide important insights into their detection accuracy, computational efficiency, capability in using pre-trained models, and domain generalizability for their deployment in real-world applications.

References

[1] Sean MacAvaney ,Hao-Ren Yao,Eugene Yang,Katina Russell,Nazli Goharian,Ophir Frieder Hate speech detection: Challenges and solutions

[2] Yin W, Zubiaga A. 2021. Towards generalisable hate speech detection: a review on obstacles and solutions. PeerJ Computer Science 7:e598

[3] Purnama Sari Br Ginting; Budhi Irawan; Casi Setianingsih Hate Speech Detection on Twitter Using Multinomial Logistic Regression Classification Method.

[4] Kristiawan Nugroho Doctoral Program, Dian Nuswantoro University, Semarang, Indonesia; Edy Noersasongko; Purwanto; Muljono; Ahmad Zainul FananiImproving Random Forest Method to Detect Hatespeech and Offensive Word

[5] Shahadat, Ashraf Bin Rony, Md. Mizanur Rahman Anwar, Md. Adnanul Joy, Eialid Ahmed Hate speech detection from social networking posts using CNN and XGBoost

[6] Sentiment Analysis of Review Datasets Using Naive Bayes and K-NN Classifier Lopamudra Dey, Sanjay Chakraborty, Anuraag Biswas, Beepa Bose, Sweta Tiwari

[7] On hyperparameter optimization of machine learning algorithms: Theory and practice by Li Yang Abdallah Shami

[8] Famili, A. et al. 'Data Preprocessing and Intelligent Data Analysis'. 1 Jan. 1997

[9] https://www.kaggle.com/datasets/arkhoshghalb/twitter-sentiment-analysis-hatred-speech?select=train.csv

[10] https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc

[11] https://realpython.com/logistic-regression-python/