**Automated Hate Speech Detection using Logistic regression and Naïve Bayes**

**Introduction**

In today’s day and age, social media is used extensively in our everyday lives. People share their opinions and interact with one another on these social media platforms. At times people tend to leave hate comments on these social media platforms, the platforms have to be able to detect the hate comments so that they can remove these comments and flag people leaving such comments.

However, it has to be clearly defined what hate speech means to be able to classify comments as hate comments. Hate speech can be defined as *language that is used to express hatred towards a specific group of people to derogate and humiliate people on the basis of their race, colour, gender, ethnicity, caste or to threaten violence towards others*.

There can be certain comments that contain offensive words but are not hate speech. For example. “The weather is f\*cking up our game” or “Gay people should be allowed to live their lives freely”. These two comments contain offensive words like f\*ck and gay but they are not hate comments, the first comment is merely a show of dissatisfaction, it does not fit the definition of hate speech and the second comment is supporting a community of people hence it is not hate speech.

The social media platforms should be able to differentiate between hate comments and comments that contain offensive language but are not hate comments because they shouldn’t end up flagging or banning accounts that are not actually leaving hate comments.

**Related work**

Bag of words approach has been used for hate speech classification in the past. Bag of words model is a model in which text is represented as an unordered collection of words. This model often has high recall but has high rates of false positives because sentences having offensive words are misclassified as hate speech.

In the modern day, the amount of curse words or offensive language used is substantially high which makes the differentiation of hate speech and offensive language quite difficult.

Sometimes words can be ambiguous, for example the word gay can mean happy or can be used in some contexts unrelated to hate speech but it is also often used in hate speech.

The existing models simply use the hate base lexicons and test if the comments contain these lexicons, if it is present then it is classified as hate speech, if it is not present then it is not. This is the reason for false positives, there is no sentimental analysis.

**Problem Statement**

Creating a model for hate speech detection that has a high precision and recall with low false positives i.e. we should be able to distinguish between offensive speech and hate speech. The model should have 3 classes – hate speech (class 0), offensive language (class 1) and neither (class 2).

The dataset is a collection of tweets (around 24 thousand tweets). The output is the precision rate, recall rate, f1 score and confusion matrix of the model.

**Approach**

We have used data preprocessing, cleaning and regularization. The 2 models used are Logistic Regression model and Naïve Bayes model. The code has been written in python. We used 2 approaches, one including sentimental analysis and hate base lexicons, and one only using the dataset.

**Dataset**

We have used a dataset of about 25,000 tweets which have been manually classified into 3 classes- hate speech, offensive language and neither by CrowdFlower (CF) workers. They did the classification on the basis of the definition of hate speech. They also used their own sentimental analysis to see the context in which the tweets were made to ensure proper classification. The final class that was determined for the tweet was based on what the majority of the workers classified the tweet.

In one approach we also used the hate base lexicons and sentiment analyzer along with the labeled dataset. We calculated the hate score (based on lexicons), vader score and textblob\_score (sentimental analyzer scores).

**Preprocessing Data**

**Approach 1 (No hate base lexicons or Vader scores)**

We cleaned the dataset by first converting all text to lowercase, removing all URLs, mentions, hashtags, non-alphabetic characters and extra spaces.

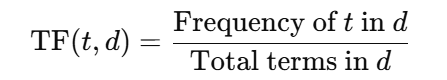
We then split the data into training sets and test sets, taking the test size as 0.2 which implies that 80% of the dataset is used to train data and 20% of the dataset is used to test data.

TF-IDF

Algorithms like Logistic Regression and Naïve Bayes work on numerical data. We used TF-IDF (Term Frequency- Inverse Document Frequency) to represent text in a vectorized format by assigning numerical weights to words based on their importance.

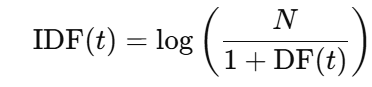
This is done by first performing tokenization i.e. splitting the text into smaller units like words or phrases. We have used unigrams (single words) and bigrams (two-word sequences) and trigrams (three-word sequences).

The Term Frequency is the relative frequency of a word in a tweet, i.e. it measures how often a term occurs in a tweet compared to the total number of terms in the tweet.



We identify the top 5000 terms based on importance and frequency to save memory and computation time.

The Inverse Document Frequency is a measure of how rare a term is across the dataset. Here t is the term, DF(t) is the number of documents containing term t and N is the total number of documents. It assigns lower weight to terms that occur frequently across all documents (common words).



We then convert the tweets in X\_train to a numerical vector by combining the TF and IDF scores of each term

TF-IDF (t, d) =TF (t, d) ×IDF (t)

Each vector has the length of 5000, then the test data is also converted into TF-IDF vectors.

**Approach 2 (Using Vader score and hate score)**

All steps are same as Approach 1 but after the TF-IDF vectorization, the TF-IDF features are combined with sentiment and hate scores, 3 additional columns are appended to the vector (hate score, Vader score, textblob\_score).

**Logistic Regression**

Logistic regression is a linear model used for classification. Since there are 3 classes in the given problem, we need to use multiclass classification, which is done using the softmax function.

Weights are initialized as 0 and the labels (y) are converted to one hot encoding. We then compute the raw scores (logits) using the following dot product: -



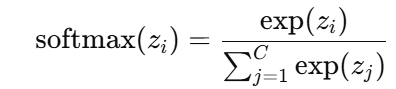
These logits are passed to the softmax function. Epochs refers to the number of iterations.

**Softmax function**

The softmax function converts logits (Raw scores zi) into a probability distribution that lies between 0 and 1.

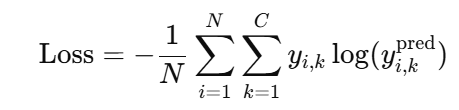
The raw scores can be any real numbers, the exponential in the softmax function ensures that the value is positive. The probabilities sum to 1 due to the denominator which normalizes the value and also ensures that the value lies between 0 and 1.

For numerical stability i.e. to prevent overflow or underflow, z is adjusted to z−max(z), this does not change the probabilities as the same amount is subtracted from each z.



**Cross entropy loss**

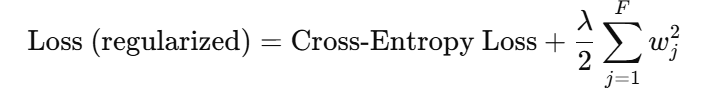
The cross-entropy loss measures how well the predicted probability distribution ypred matches the true labels ytrue. It has low loss values for correct predictions and high loss values for incorrect predictions. yi,k is 1 only for the correct class and 0 otherwise due to one hot encoding.



**Regularization**

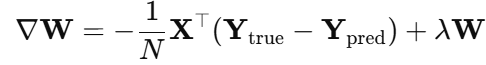
Regularization is a technique used to prevent overfitting i.e. lack of generalization. It also controls the complexity of the model by penalizing large weights hence avoiding over complex decision boundaries.

We have used L2 regularization (Ridge regularization) for the loss function. This penalizes excessively large weights to ensure generalization. Where λ is the regularization strength.



**Gradient Descent**

We find the gradient of the loss with respect to the weights where λW is the regularization gradient.



The weights are then updated using gradient descent, where η is the learning rate.

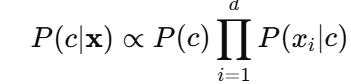


Finally, the prediction is made in which the class with the highest probability from softmax function is selected.

**Naïve Bayes Model**

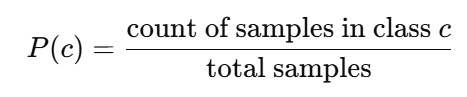
The naïve Bayes model is a probabilistic classification model based on the Bayes theorem. It assumes independence of the features i.e. bag of words assumption. This is a naïve assumption hence the name Naïve Bayes model.

P(c) is the prior probability of the class, P(xi∣c) is the probability of feature xi given class c, this is called the likelihood. This means given a class c, what is the probability of feature xi being true.



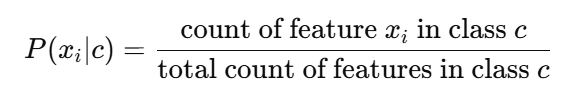
**Prior probability**

All the class prior probabilities are computed as per the following formula: -

****

**Likelihoods**

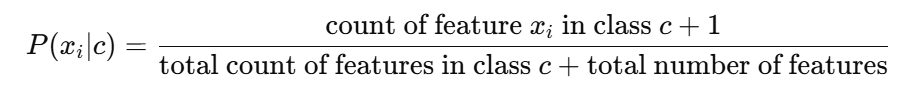
Typically, the likelihoods of features given class c are calculated using



However, a problem occurs when the feature doesn’t occur in class c, this would make the probability of the likelihood of the feature to be 0 and the probability of the entire class 0 because we are using the product of probabilities, this problem is called the zero-frequency problem.

**Laplace smoothing**

Laplace smoothing is a method used to solve the zero-frequency problem; it ensures that every feature has non-zero probability even if the frequency is 0. We add 1 to each feature in every class, this does not change the probabilities as the denominator also increases by the number of features but ensures non zero probability.



**Log posterior**

To avoid numerical underflow, we take the log of the class probabilities and the logarithm of the likelihoods. The log posterior is computed using: -



We predict the class having the largest posterior probability.

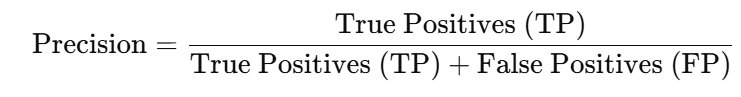
**Evaluation metrices**

We have used 3 evaluation metrices, precision, recall and F1-score.

**Precision**

Precision is a measure of the number of instances the model classified as positive that were actually positive.

It is the ratio of true positive predictions to the total number of predictions that were positive (true positives +false positives)



True positives are instances that are correctly classified as positive and false positives are instances that are classified as positive but are actually negative

Precision is important in our given problem because the cost of false positives is high i.e. we don’t want an innocent tweet to be classified as hate leading to the account being flagged or restricted.

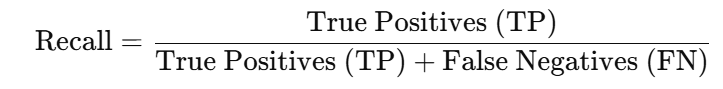
High precision means that when the model predicts positive, then it is correct most of the time.

.

**Recall**

Recall is a measure of how many of the actual positives the model correctly identified. It is also called sensitivity or true positive rate.

Recall is the ratio of true positive predictions to the total number of actual positives in the data.

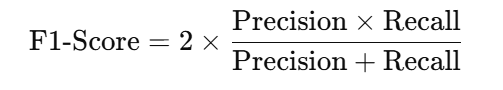


False negatives are instances that should be positive but are classified as negative.

High recall means the model is good at identifying positive instances i.e. hate speech.

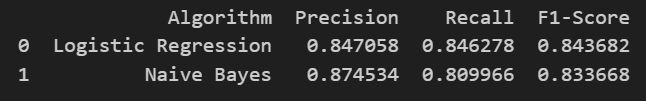
**F1-score**

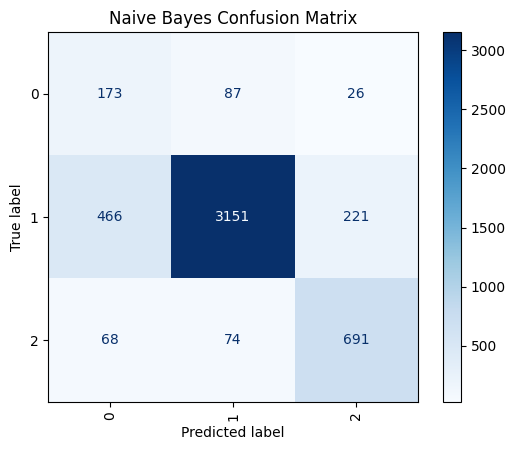
The F1-score is the harmonic mean of the precision and recall. It is a single metric that combines the previous two metrics. F1-score is important when we want to balance the precision and recall scores.

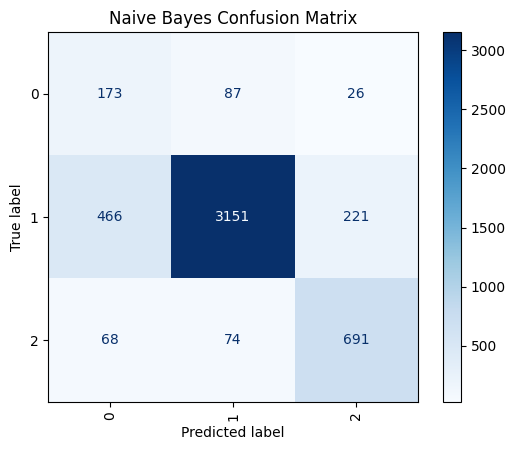


**Results**

The logistic regression model performs Slightly better than the Naïve Bayes model due to the use of regularization, hyperparameter tuning and the probabilistic output. The Output has been shown in the below screenshots, the confusion matrix shows us the true positives, true negatives, false positives and false negatives of the model.







**Hyperparameter tuning**

We have three hyperparameters in the logistic regression model which are- learning rate, regularization strength and epochs. By varying these hyperparameters, the output of our model varies, after multiple experiments we arrived at the following hyperparameters->

learning\_rate=0.05, reg\_lambda=0.001, epochs=500

Other values were tried like learning\_rate=0.1, reg\_lambda=0.01, epochs=300, this gave rise to Precision=0.76, Recall=0.78, F1-Score=0.69 output which is significantly worse than the output that we have obtained at our set hyperparameters.

**Conclusion and future direction**

We conclude that using logistic regression works better than naïve bayes for hate speech classification. Logistic regression is a good method to ensure that we don’t misclassify curse words as hate speech due to the use of techniques like L2 regularization, hyperparameter tuning and probabilistic output.

Future work that can be done on this project is that hyperparameter tuning can be automated to get the optimal values of the parameters, in this project we manually tested values to perform hyperparameter tuning.

We have used 0.8 for the training set and 0.2 for the test set, varying these and tuning this to optimal values can further enhance the results.

We can also incorporate advanced machine learning concepts to create an agent for performing hate speech classification using other models like neural networks, genetic engineering etc.

**References**

1. Davidson, T., Warmsley, D., Macy, M., and Weber, I., "Automated hate speech detection and the problem of offensive language," *Proc. 11th Int. Conf. Web Social Media (ICWSM 2017)*, 2017
2. El-Sayed, T., Elrashidy, M., Mustafa, A., and El-Sayed, A., "Hate speech detection by classic machine learning," *Proc. 3rd Int. Conf. Electronic Eng. (ICEEM)*, 2023