PROJECT REPORT

Generalizable Hate Speech Detection

Submitted in partial fulfillment of the requirements
for the award of the degree of

Master of Computer Application
(2024-26, Ist Sem, Batch-1)

Under the supervision of:
Dr. Bhawana
Assistant Professor (CSET)

Submitted By:
Pawan Kumar Pandey (S24MCAG0015)
Bhanu Pundir(S24MCAG0049)

Plot Nos 8-11, TechZone 2, Greater Noida, Uttar Pradesh 201310

2024-26

ABSTRACT

The main challenge in automatic hate speech detection is distinguishing between hate speech and offensive language. Current word-based detection methods often label any message with certain words as hate speech, making them less accurate.
Previous supervised research has also faced difficulty in distinguishing between hate speech and offensive language. So we created a crowd-sourced hate speech dictionary to collect tweets that use hate speech keywords.

Then, we used crowdsourcing to categorize these tweets into three groups: hate speech, offensive language, or neutral. We trained multi-class classifier to tell these categories apart.
By observing and analyzing predictions and errors, we found some cases where hate speech and offensive language can be clearly separated, as well as situations where this separation is harder.

Racist language is more likely to be detected accurately, but some types are still challenging to classify.
Hate speech as language that expresses hatred towards a targeted group in a harmful or derogatory manner, while offensive language is broader and can include curse words and slurs without necessarily targeting specific groups with hateful intent.

TABLE OF CONTENTS

INTRODUCTION

Distinguishing between hate speech and offensive languages is a harder task. There is not any hard rule or definition, but hate speeches are generally such language that hurts sentiments or may targets specific groups of peoples, especially those who are already at a disadvantage. In the United State, hate speech is allowed under the First Amendment, which gives people the right to free speech. But in many countries like the UK, Canada, and France, there are laws against hate speech. These laws say hate speech is language that could cause violence or problems in society, especially if it targets minority groups. People who use hate speech can be fined or even sent to jail. This applies to social media too, and that's why websites like Facebook and Twitter have rules against hate speech. They don't allow attacks based on things like race, gender, or sexual orientation, or any threats of violence.

We define hate speech as words that are meant to express hate towards a group of people or insult them. In some extreme cases, it can also threaten or cause violence, but not all hate speech is that extreme. Some offensive language may not count as hate speech, especially if the intent is different. For example, some African Americans use the word "ngga" in a friendly way, some people quote rap lyrics with words like "he" and "b*tch," and teenagers might use hurtful words while playing video games. This kind of language is common online, so it's important to have a clear way of detecting hate speech.

Past studies have tried to figure out the difference, but many still mix up hate speech and offensive language. In this paper, we labeled tweets as one of three things: hate speech, offensive language, or neither. We trained a computer model to tell them apart, and then looked at the results to see how well it worked. Our results showed that having more specific labels helps, but there are still challenges. We concluded that future research should focus more on understanding the context and different ways hate speech is used.

Earlier methods used simple word lists to find hate speech, but they often made mistakes and labeled offensive tweets as hate speech. For example, tweets that use the word "ngger" are more likely to be labeled as hate speech, while "ngga" is usually seen as just offensive. Some words can be tricky, like "gay," which can be used in a bad way or in other ways that are not related to hate.

Researchers have also tried to use grammar and sentence structure to better identify hate speech. For example, sentences like "kill the Jews" are a clear sign of hate. Other methods have tried using patterns in sentences, like the structure "I hate <group>." However, some studies still confuse hate speech with just offensive language, which makes it harder to know if they are really detecting hate speech. Newer techniques like neural language models show promise, but they often still use broad definitions of hate speech. Non-linguistic features, like knowing the gender or ethnicity of the person writing, could help, but that information is not always available or reliable on social media.

PROBLEM STATEMENT

Social media websites like Facebook and Twitter are growing very fast, and because of that, there is more hate speech and hurtful language online. This makes it really important to create computer programs that can automatically find and sort this bad content. But, finding hate speech is hard because it is sometimes difficult to tell it apart from other types of hurtful language. Some computer programs make mistakes and think that any bad language is hate speech, even if it's not. Also, past methods have not been good at telling the difference between hate speech (which hurts certain groups of people on purpose) and just bad language (that's not meant to hurt anyone badly).

This is a big challenge because people might use different words or phrases to express anger or frustration, and it's important for a computer to understand the context behind those words. Sometimes, a comment might sound rude, but it may not be intended to harm anyone. Other times, a person might use words that are not clearly hateful but are still meant to target and hurt others. The goal of this research is to build a better system that can make these important distinctions.

By creating a special program that can clearly separate hate speech, offensive language, and neutral content, we can help make social media platforms safer and friendlier for everyone. The program will not only help find bad content but also reduce mistakes that might unfairly flag harmless comments. This will help online communities stay respectful while preventing harmful content from spreading.

LITERATURE REVIEW

The Introduction to Speech Segregation and Offensive Language Research The area of research that involves hate speech and offensive language detection in the field of NLP is very interesting, especially due to the explosion of social media like Twitter, Facebook, and Instagram, which increasingly provide avenues for disseminating dangerous content, thereby rendering relevant the quest for automatically propounding systems that detect this kind of content: such a quest is extremely rewarding. Whilst hate speech promotes a kind of violence- or segregation-based behavior directed toward race, religion, ethnicity, or gender, offensive speech does not necessarily bring harm yet still violates social conventions.

The Challenges in Detecting Hate and Offensive Speech The challenge of detecting hate and offensive insults is quite intricate in view of the fact that most social mediaists traverse informality in their posts while being shoptalk heavy and, at times, multilingual; and this latter difficulty is compounded in a still worse way when it comes to Hinglish-a hybrid lingo spanning Hindi and English-ij formed of unfinished phrases, offhand-word scripts, and culturally bound references that are nearly impossible for conventional bastions to interpret accurately. Most hate speech detection systems create their model based on the labeled data, but mixing languages such as Hinglish tends to raise yet another layer of complexity.
Key findings:

Hate Speech vs. Offensive Language: hate speech as language that expresses hatred towards a targeted group in a harmful or derogatory manner, while offensive language is broader and can include curse words and slurs without necessarily targeting specific groups with hateful intent.

Data and Classification: Using a crowd-sourced lexicon of hate speech and offensive terms, they collected a sample of tweets, which were manually labeled into three categories: hate speech, offensive but not hate speech, and neither. They trained a multi-class classifier to distinguish between these categories. The results showed that while tweets containing explicit slurs (like "ngger" or "fggot") are more likely to be classified as hate speech, others (like sexist language) are typically classified as offensive rather than hateful.

Challenges in Classification: The model showed that certain types of hate speech, especially less frequent ones (e.g., anti-Chinese hate), were misclassified more frequently. Tweets containing multiple slurs were often correctly labeled, but tweets that were borderline or lacked explicit hate speech terms were more challenging to categorize.

Error Analysis: Misclassifications often stem from the context or lack of strong keywords associated with hate speech. Tweets containing words like "btch" or "pssy" were labeled as offensive, but the model still struggled with cases where hate speech was implied rather than explicit.

Future Directions: future work should consider the social context in which hate speech occurs, and avoid over-relying on lexicons, as they may include terms that are offensive in some contexts but not necessarily hateful. They also suggest that better training data, with nuanced distinctions and greater
This study contributes to the challenge of making automated systems for detecting hate speech more precise by providing a more granular approach to classification and addressing the biases inherent in previous work.

PROPOSED MODEL

In order to carry out sentiment identification, every tweet was converted into lowercase letters and processed using the Porter Stemmer to remove any extra endings from the words. Feature generation was based on unigrams, bigrams, and trigrams, each with some weighting according to its TF-IDF score. Finally, to understand the sentences, we used NLTK to create unigrams, bigrams, and trigrams of Part-of-Speech (POS) tags.

The quality of the tweet was analyzed taking the Flesch-Kincaid Grade Level and Flesch Reading Ease scores with the number of sentences being set to one. The sentiment of the tweet was analyzed using a special social media sentiment dictionary (Hutto and Gilbert 2014) and a sentiment score was assigned to it.

Additionally, we added features like whether a tweet has hashtags, mentions, retweets, or URLs, and summarized with the counts of characters, words, and syllables in each tweet.

After this, we performed logistic regression with L1 penalties for culling irrelevant features from the features in the dataset. Then, we validated several models previously used by other researchers: logistic regression, naive Bayes, decision trees, random forests, and linear support vector machines (SVMs). Each single model was cross-validated using 5-fold cross-validation, with 10% of the data used for testing so that overfitting might be avoided. After experimenting with numerous classifiers and combinations of tuning parameters, we found the two highest-performing classifiers were logistic regressions and linear SVMs.

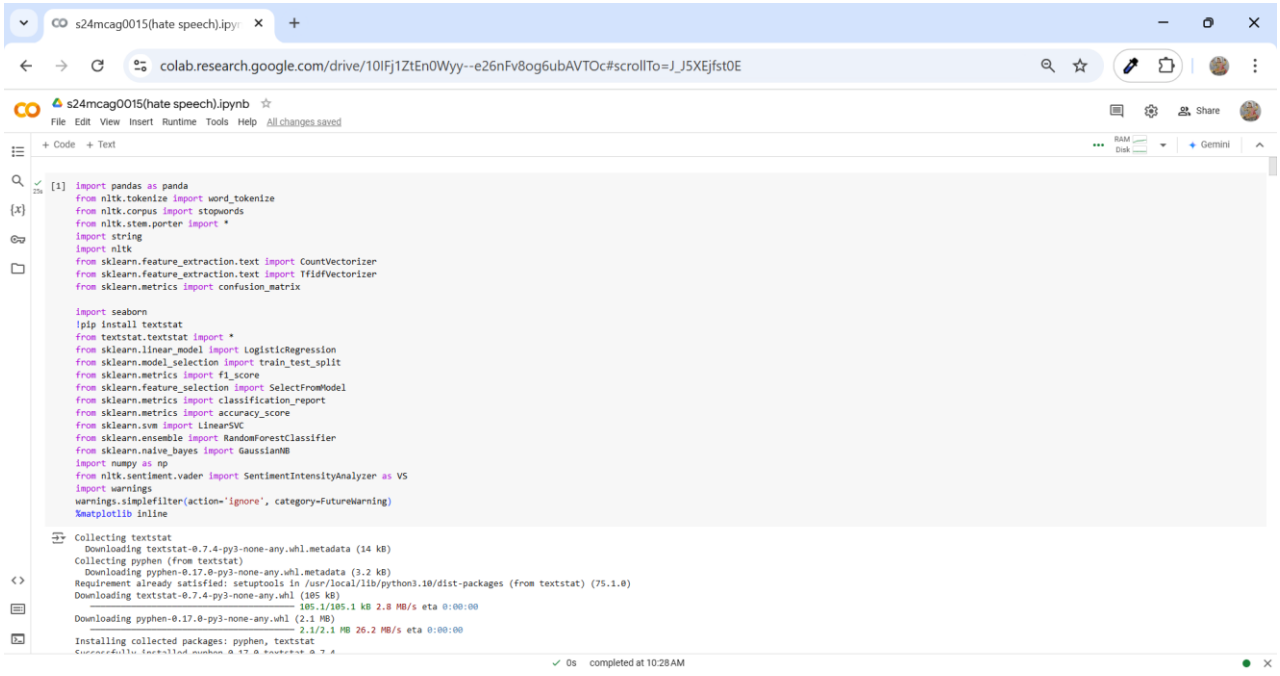
In our final**, we chose a logistic regression model with L2 penalty** since it allows us to interpret predicted probabilities for each class and has worked successfully in previous studies (Burnap and Williams 2015; Waseem and Hovy 2016). We fitted the final model on the complete dataset and used it to predict labels for tweets. One-vs-rest was employed, where each representation of itself.

IMPLEMENTATION AND RESULTS

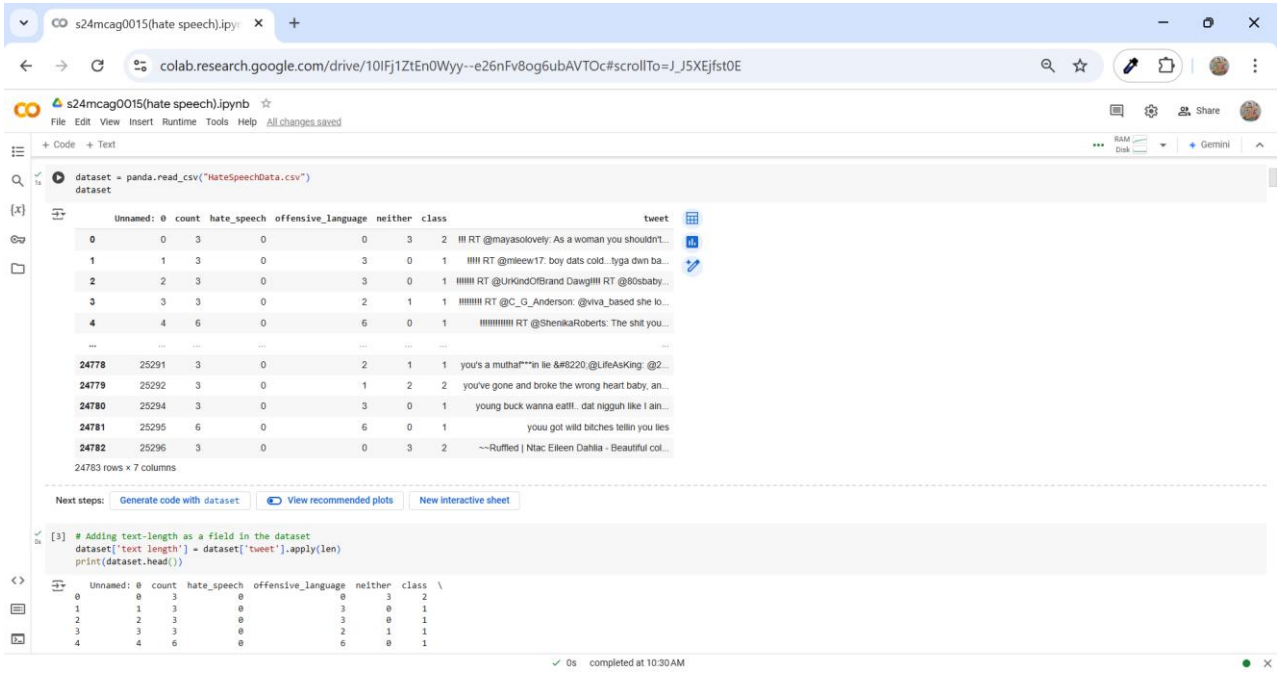
Data Collection and Labelling:

The researchers used a hate-speech lexicon from Hatebase.org, which labels specific words and phrases identified as hate speech; the researchers then used the Twitter API to query all tweets containing these terms. This resulted in tweets corresponding to 33,458 users providing a total of 85.4 million tweets. Out of these, they randomly selected 25,000 tweets that contained dictionary-referenced words and had them manually labelled by CrowdFlower workers. Each tweet was then classified into one of the following categories: hate speech, offensive but not hate speech, or neither. Workers were told to view the words' usage in context rather than at face value. A total of 24,802 tweets were labelled: 5% as hate speech, 76% as offensive, and 16.6% as non-offensive.

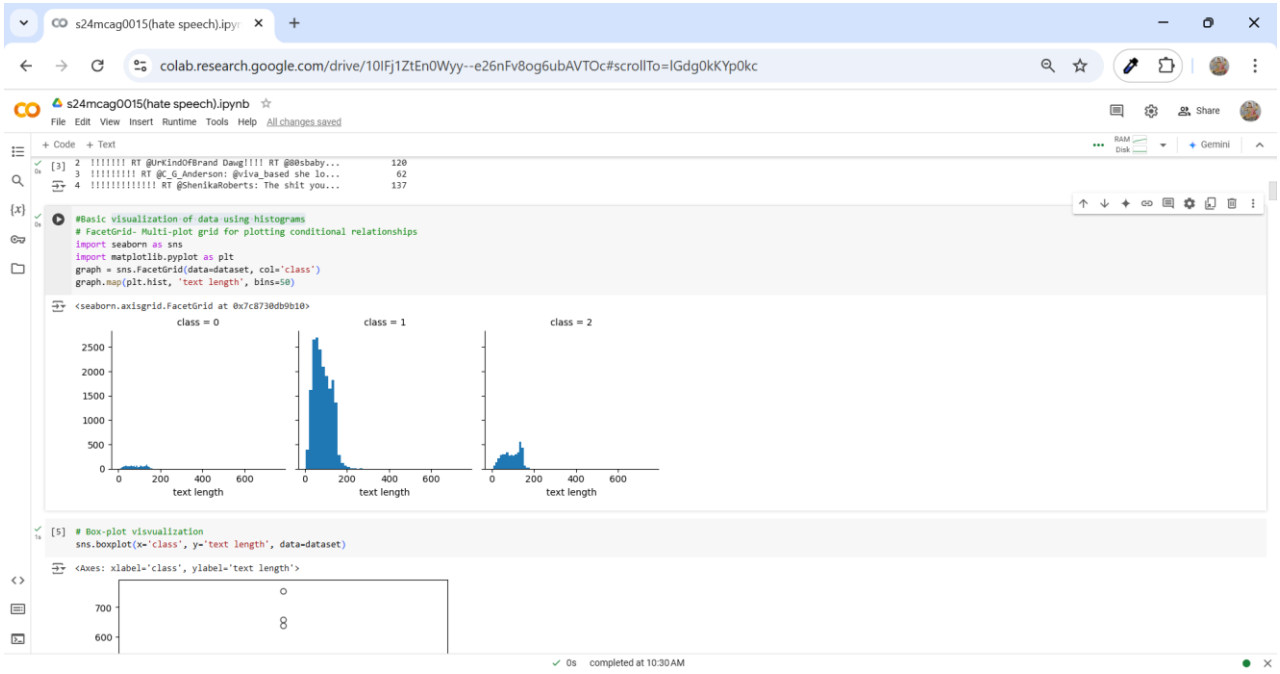
Preprocessor Files



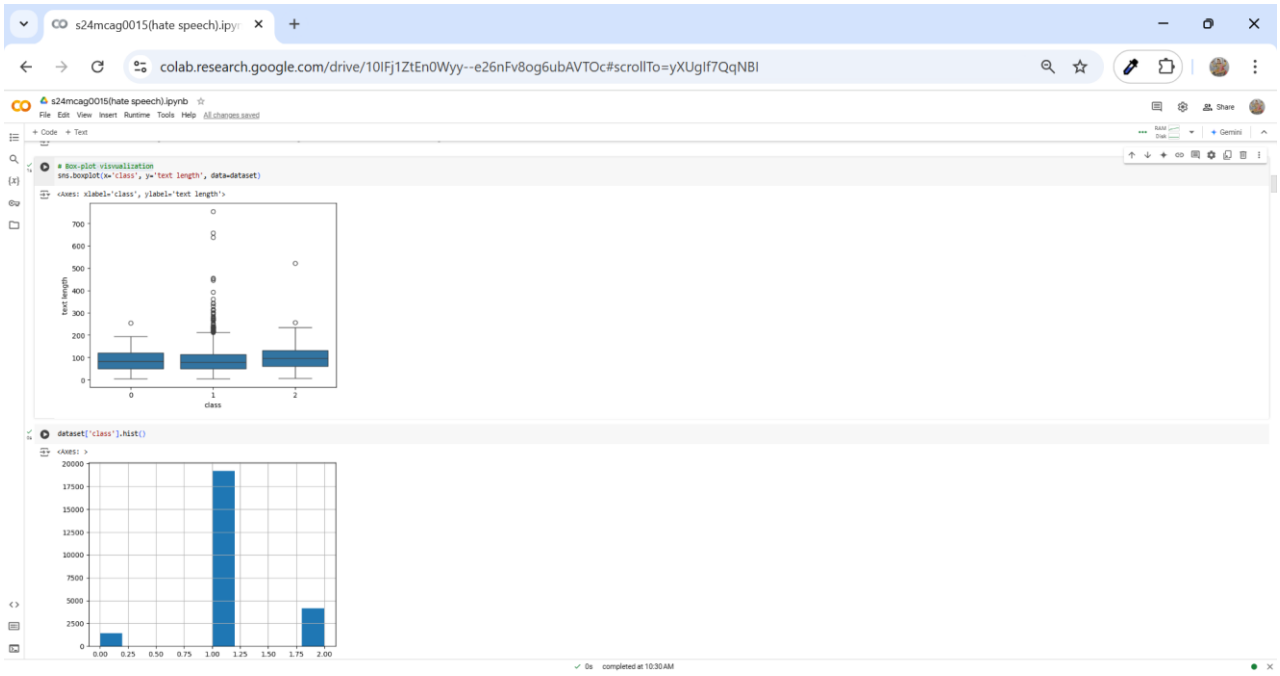
Importing CSV file and Fining of Data



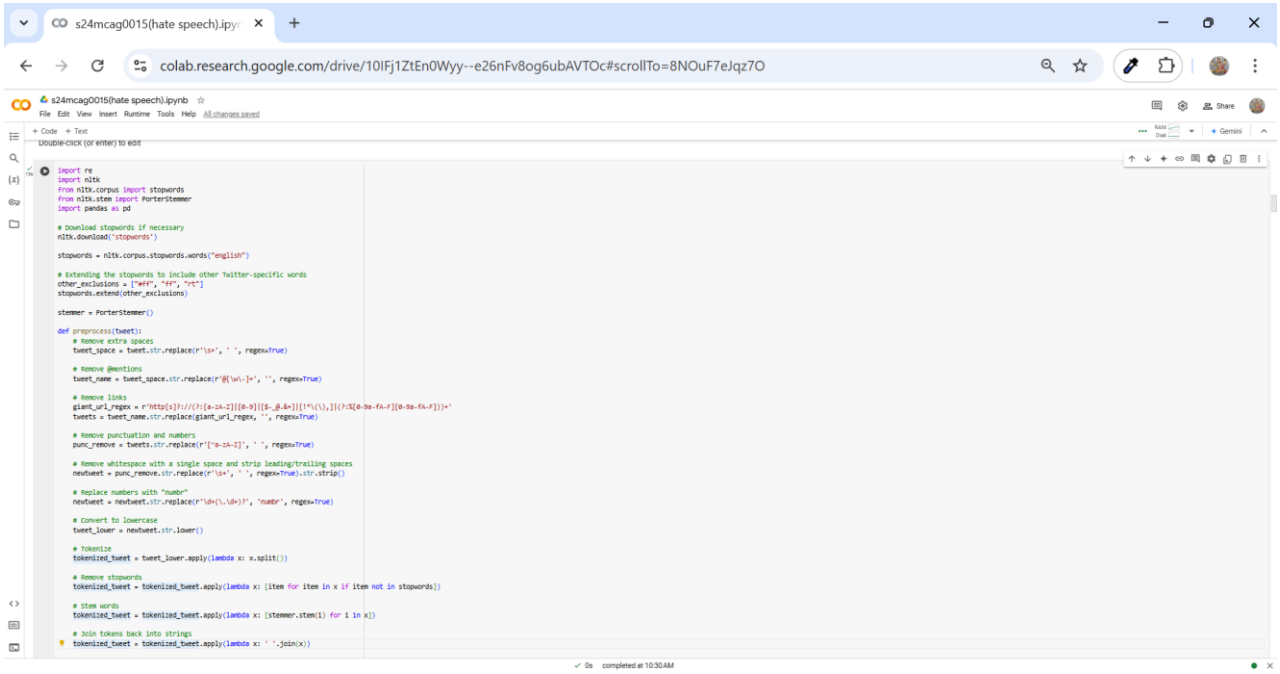
visualization of data using histograms



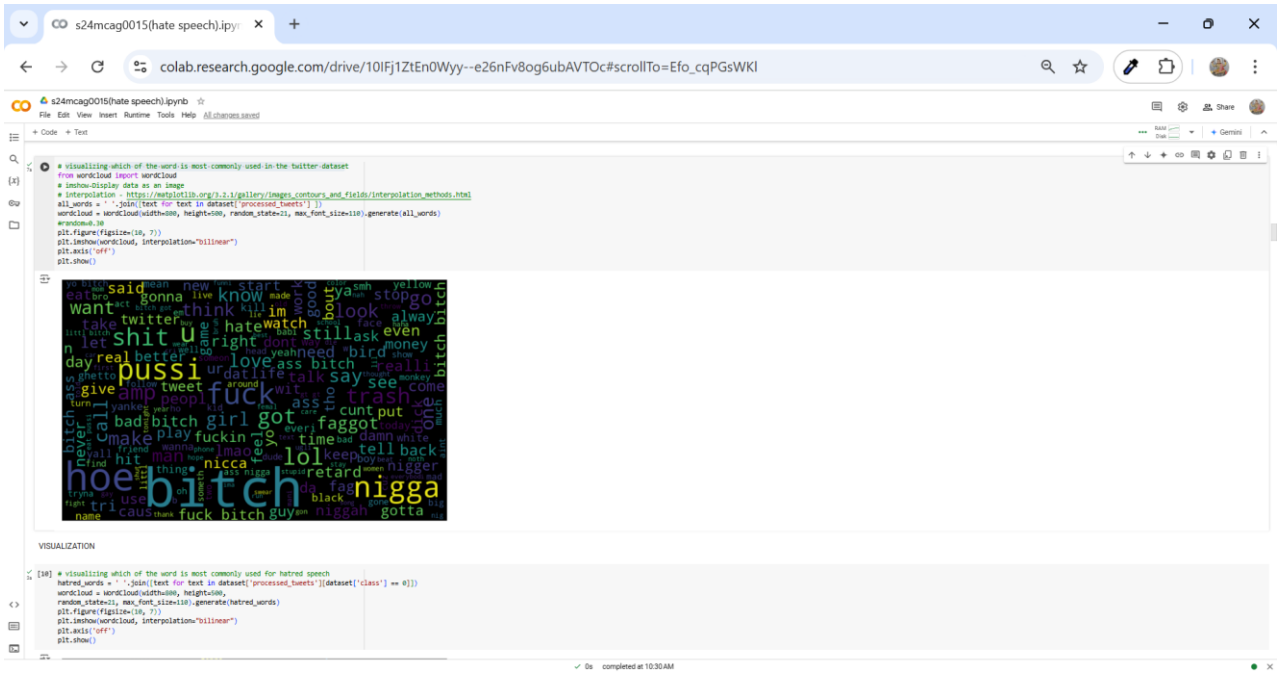
Box-plot visvualization



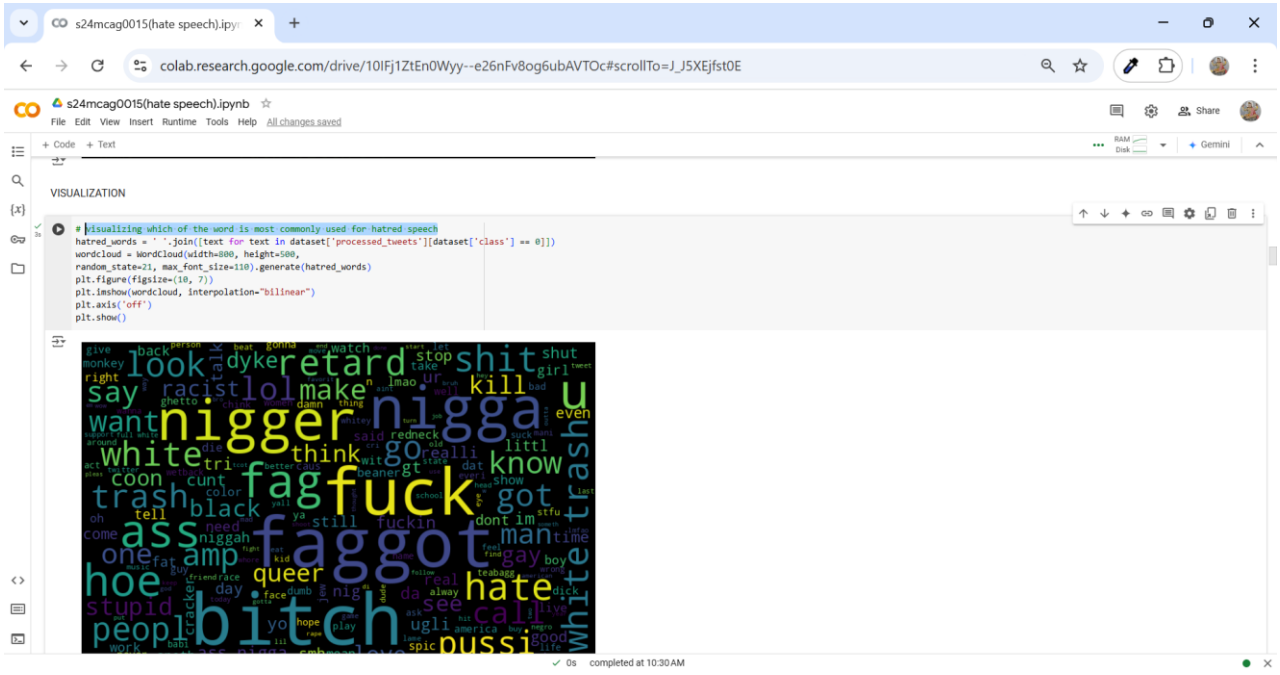
PREPROCESSING OF **TWEETS**



visualizing which of the word is most commonly used in the twitter dataset



visualizing which of the word is most commonly used for hatred speech

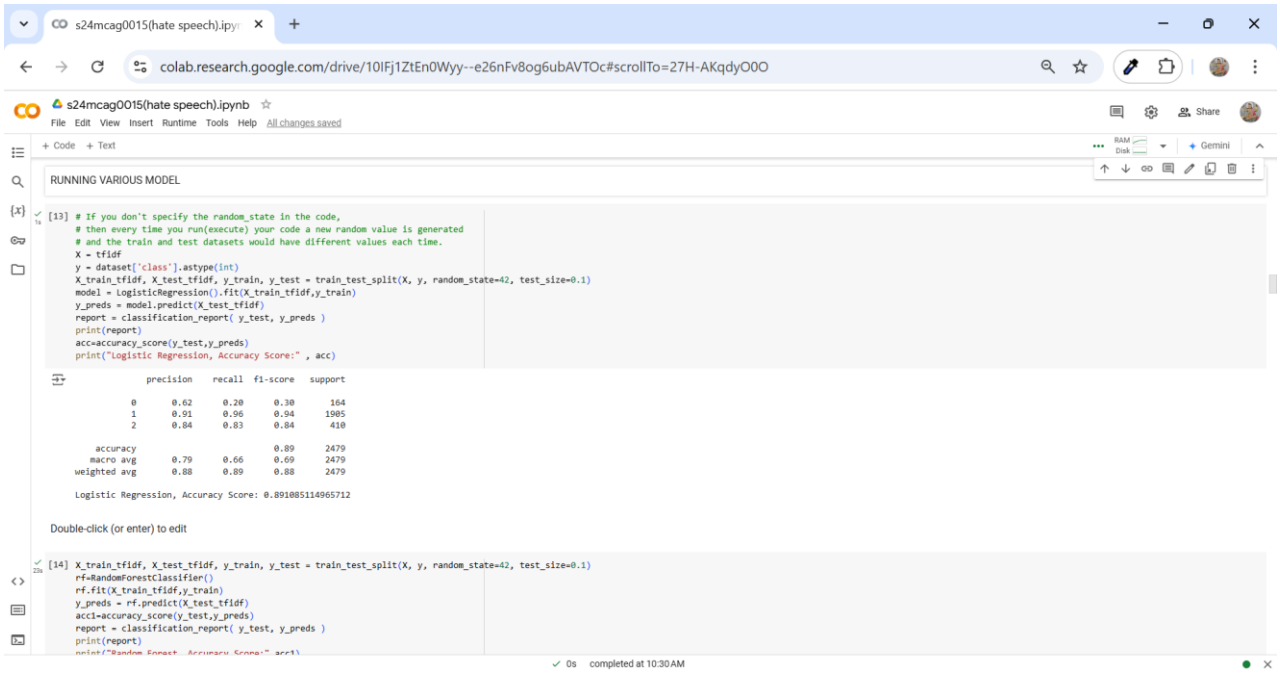


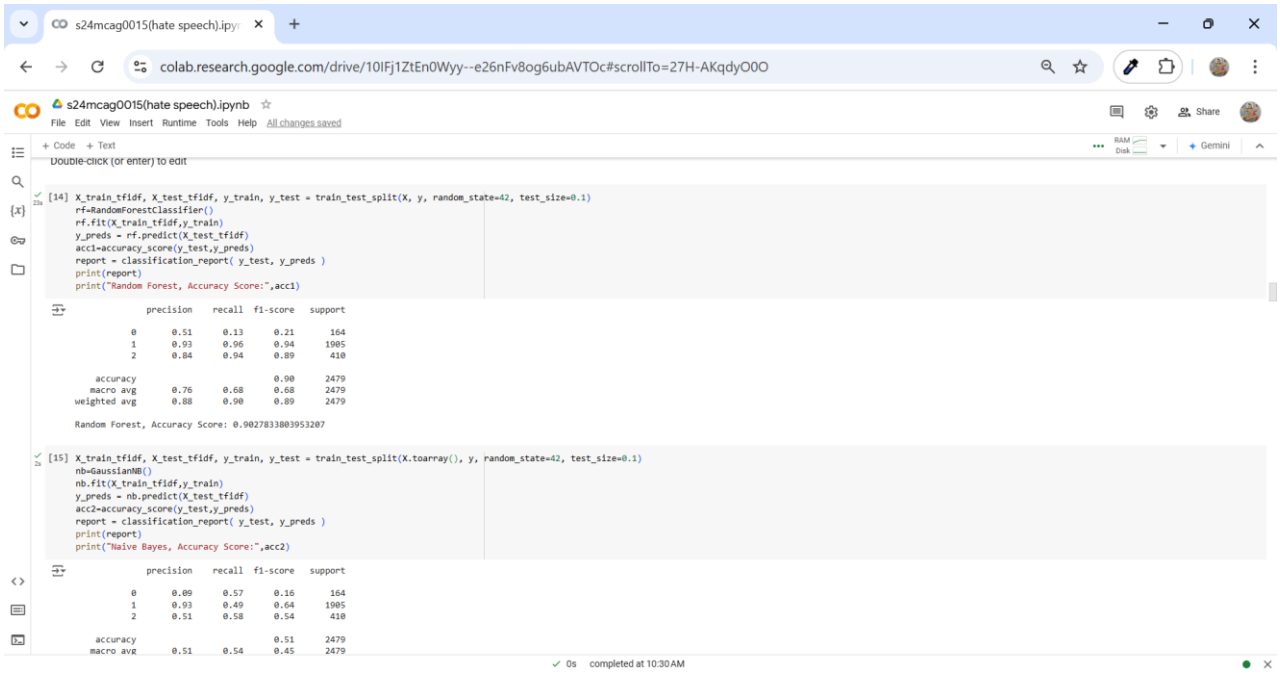
visualizing which of the word is most commonly used for offensive speech

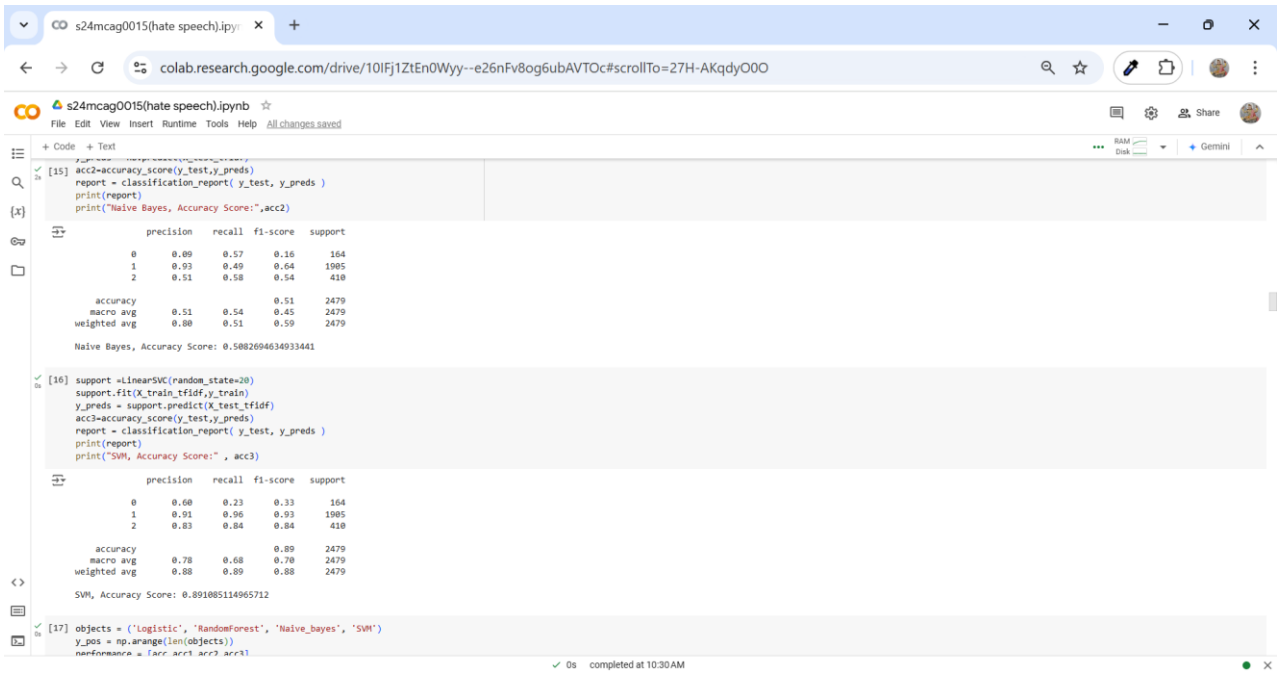
Feature Extraction:

For each tweet, they changed it into lower case and stemmed it with the help of the Porter stemmer. The features included unigrams, bigrams, and trigrams, each weighted by its TF-IDF score. POS tag features based on Penn tagger were provided to understand the structure of the sentences. Other features included readability scores such as the Flesch-Kincaid Grade Level and the Flesch Reading Ease, sentiment scores derived by a social media sentiment lexicon, and binary indicators for hashtags, mentions, retweets, and URLs.

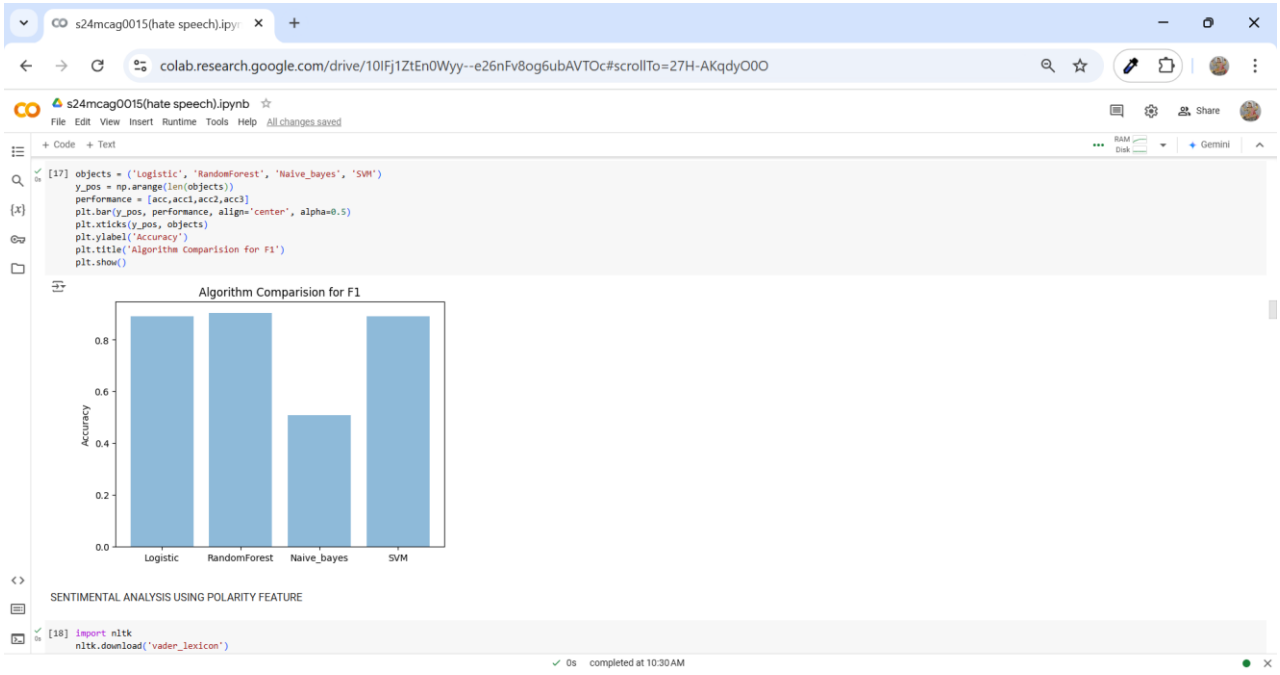
Running Various Models



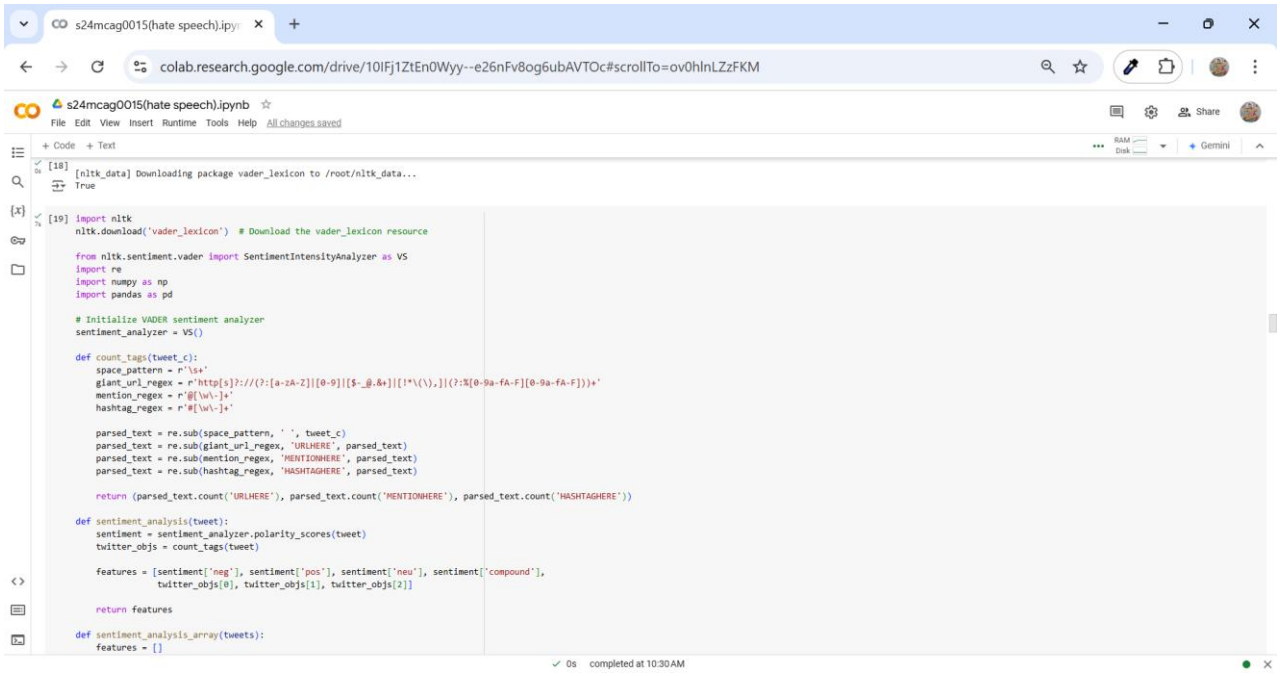

s24mcag0015(hate speech).ipyn
colab.research.google.com/drive/10IFj1ZtEn0Wyy--e26nFv8og6ubAVTOc#scrollTo=27H-AKqdyO0O
s24mcag0015(hate speech).ipynb
File Edit View Insert Runtime Tools Help All changes saved
Share
+ Code + Text
Gemini
Double-click (or enter) to edit
[14] X_train_tfidf, X_test_tfidf, y_train, y_test = train_test_split(X, y, random_state=42, test_size=0.1)
rf=RandomForestClassifier()
rf.fit(X_train_tfidf,y_train)
y_preds = rf.predict(X_test_tfidf)
acc1=accuracy_score(y_test,y_preds)
report = classification_report(y_test, y_preds)
print(report)
print("Random Forest, Accuracy Score:",acc1)
precision recall f1-score support
0 0.51 0.13 0.21 164
1 0.93 0.96 0.94 1905
2 0.84 0.94 0.89 410
accuracy 0.90 2479
macro avg 0.76 0.68 0.68 2479
weighted avg 0.88 0.90 0.89 2479
Random Forest, Accuracy Score: 0.9027833803953207
[15] X_train_tfidf, X_test_tfidf, y_train, y_test = train_test_split(X.toarray(), y, random_state=42, test_size=0.1)
nb=GaussianNB()
nb.fit(X_train_tfidf,y_train)
y_preds = nb.predict(X_test_tfidf)
acc2=accuracy_score(y_test,y_preds)
report = classification_report(y_test, y_preds)
print(report)
print("Naive Bayes, Accuracy Score:",acc2)
precision recall f1-score support
0 0.09 0.57 0.16 164
1 0.93 0.49 0.64 1905
2 0.51 0.58 0.54 410
accuracy 0.51 2479
macro avg 0.51 0.54 0.45 2479
0s completed at 10:30 AM

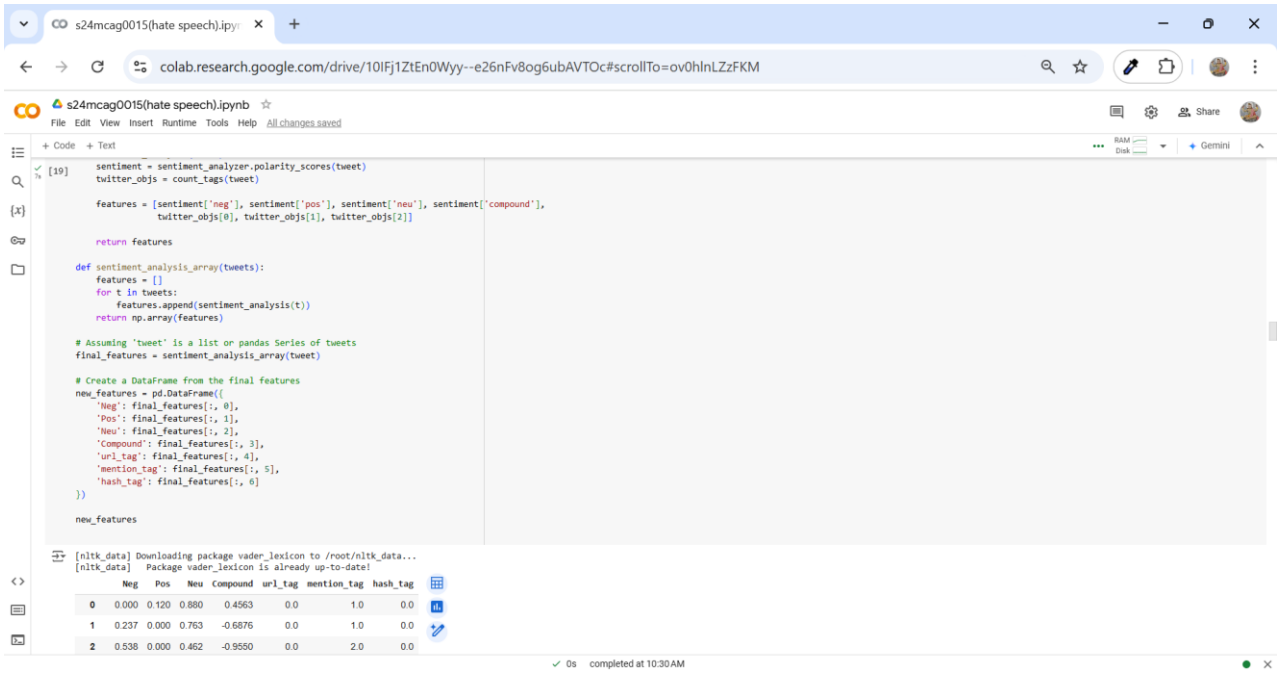

s24mcag0015(hate speech).ipyn
colab.research.google.com/drive/10IFj1ZtEn0Wyy--e26nFv8og6ubAVTOc#scrollTo=27H-AKqdyO0O
s24mcag0015(hate speech).ipynb
File Edit View Insert Runtime Tools Help All changes saved
Share
+ Code + Text
Gemini
[15] acc2=accuracy_score(y_test,y_preds)
report = classification_report(y_test, y_preds)
print(report)
print("Naive Bayes, Accuracy Score:",acc2)
precision recall f1-score support
0 0.09 0.57 0.16 164
1 0.93 0.49 0.64 1905
2 0.51 0.58 0.54 410
accuracy 0.51 2479
macro avg 0.51 0.54 0.45 2479
weighted avg 0.80 0.51 0.59 2479
Naive Bayes, Accuracy Score: 0.5082694634933441
[16] support =LinearSVC(random_state=20)
support.fit(X_train_tfidf,y_train)
y_preds = support.predict(X_test_tfidf)
acc3=accuracy_score(y_test,y_preds)
report = classification_report(y_test, y_preds)
print(report)
print("SVM, Accuracy Score:" , acc3)
precision recall f1-score support
0 0.60 0.23 0.33 164
1 0.91 0.96 0.93 1905
2 0.83 0.84 0.84 410
accuracy 0.89 2479
macro avg 0.78 0.68 0.70 2479
weighted avg 0.88 0.89 0.88 2479
SVM, Accuracy Score: 0.891085114965712
[17] objects = ('Logistic', 'RandomForest', 'Naive_bayes', 'SVM')
y_pos = np.arange(len(objects))
0s completed at 10:30 AM

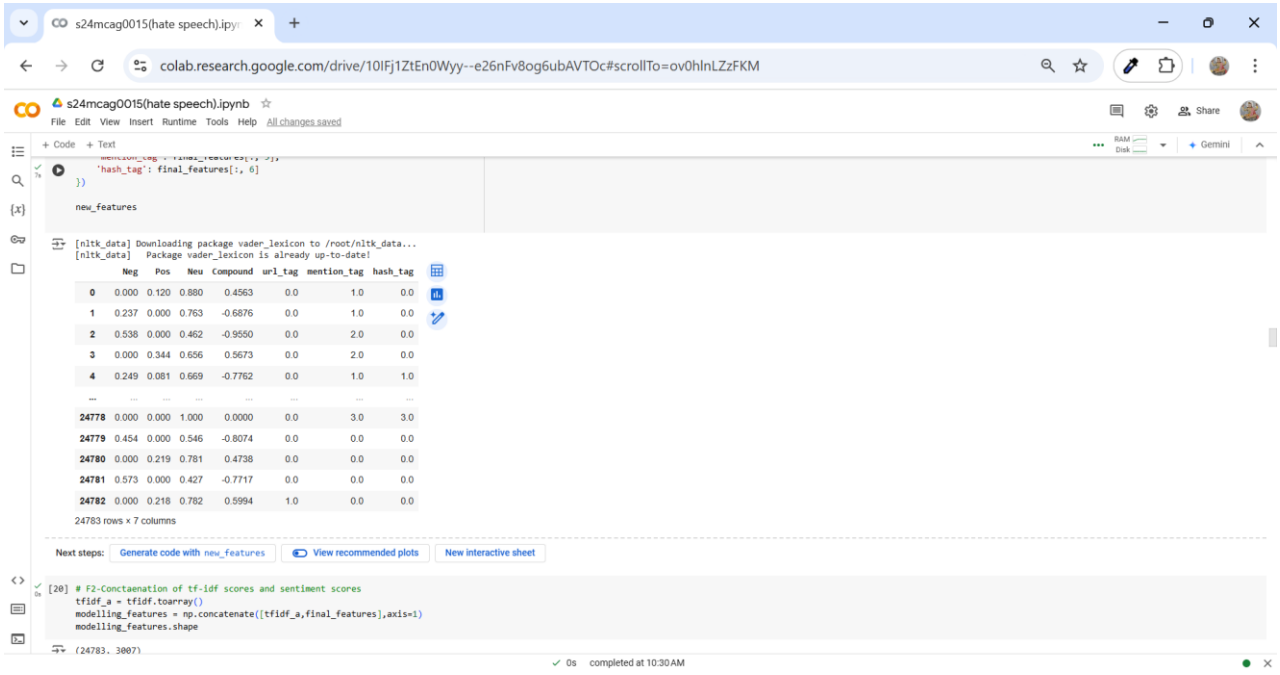
Algorithm Comparision F1



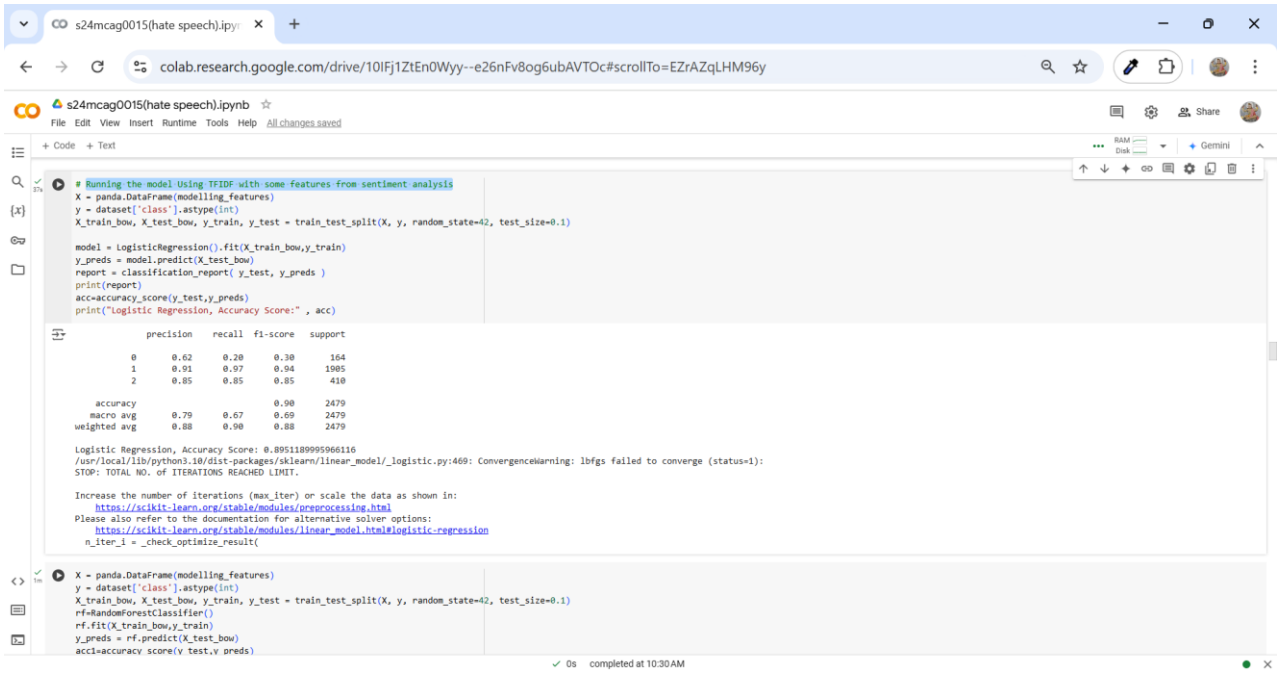
SENTIMENTAL ANALYSIS USING POLARITY FEATURE

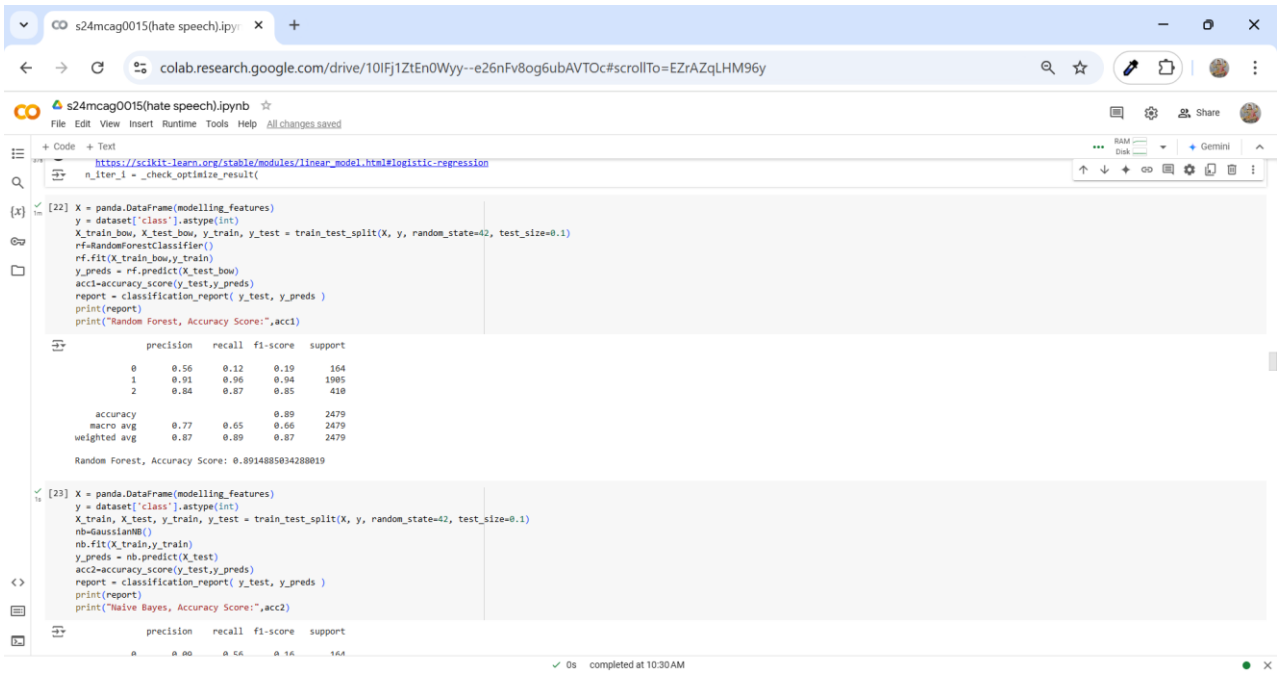






Running the model Using TFIDF with some features from sentiment analysis


s24mcag0015(hate speech).ipyn
colab.research.google.com/drive/10IFj1ZtEn0Wyy--e26nFv8og6ubAVTOc#scrollTo=EZrAZqLHM96y
s24mcag0015(hate speech).ipynb
File Edit View Insert Runtime Tools Help All changes saved
Share
Gemini
# Running the model Using TFIDF with some features from sentiment analysis
X = panda.DataFrame(modelling_features)
y = dataset['class'].astype(int)
X_train_bow, X_test_bow, y_train, y_test = train_test_split(X, y, random_state=42, test_size=0.1)
model = LogisticRegression().fit(X_train_bow,y_train)
y_preds = model.predict(X_test_bow)
report = classification_report(y_test, y_preds)
print(report)
acc=accuracy_score(y_test,y_preds)
print("Logistic Regression, Accuracy Score:" , acc)
precision recall f1-score support
0 0.62 0.20 0.30 164
1 0.91 0.97 0.94 1905
2 0.85 0.85 0.85 410
accuracy 0.90 2479
macro avg 0.79 0.67 0.69 2479
weighted avg 0.88 0.90 0.88 2479
Logistic Regression, Accuracy Score: 0.8951189995966116
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:469: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
Increase the number of iterations (max_iter) or scale the data as shown in:
https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
n_iter_i = _check_optimize_result(
X = panda.DataFrame(modelling_features)
y = dataset['class'].astype(int)
X_train_bow, X_test_bow, y_train, y_test = train_test_split(X, y, random_state=42, test_size=0.1)
rf=RandomForestClassifier()
rf.fit(X_train_bow,y_train)
y_preds = rf.predict(X_test_bow)
acc1=accuracy_score(y_test,y_preds)
0s completed at 10:30 AM


s24mcag0015(hate speech).ipyn
colab.research.google.com/drive/10IFj1ZtEn0Wyy--e26nFv8og6ubAVTOc#scrollTo=EZrAZqLHM96y
s24mcag0015(hate speech).ipynb
File Edit View Insert Runtime Tools Help All changes saved
Share
Gemini
https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
n_iter_i = _check_optimize_result(
[22] X = panda.DataFrame(modelling_features)
y = dataset['class'].astype(int)
X_train_bow, X_test_bow, y_train, y_test = train_test_split(X, y, random_state=42, test_size=0.1)
rf=RandomForestClassifier()
rf.fit(X_train_bow,y_train)
y_preds = rf.predict(X_test_bow)
acc1=accuracy_score(y_test,y_preds)
report = classification_report(y_test, y_preds)
print(report)
print("Random Forest, Accuracy Score:",acc1)
precision recall f1-score support
0 0.56 0.12 0.19 164
1 0.91 0.96 0.94 1905
2 0.84 0.87 0.85 410
accuracy 0.89 2479
macro avg 0.77 0.65 0.66 2479
weighted avg 0.87 0.89 0.87 2479
Random Forest, Accuracy Score: 0.8914885034288019
[23] X = panda.DataFrame(modelling_features)
y = dataset['class'].astype(int)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42, test_size=0.1)
nb=GaussianNB()
nb.fit(X_train,y_train)
y_preds = nb.predict(X_test)
acc2=accuracy_score(y_test,y_preds)
report = classification_report(y_test, y_preds)
print(report)
print("Naive Bayes, Accuracy Score:",acc2)
precision recall f1-score support
0s completed at 10:30 AM

```
print(report)
print("Naive Bayes, Accuracy Score:",acc2)
```

```
              precision    recall  f1-score   support

           0       0.09      0.56      0.16       164
           1       0.93      0.49      0.64      1905
           2       0.53      0.63      0.58       410

    accuracy                           0.52      2479
   macro avg       0.52      0.56      0.46      2479
weighted avg       0.81      0.52      0.60      2479

Naive Bayes, Accuracy Score: 0.5191609519967729
```

```
X = panda.DataFrame(modelling_features)
y = dataset['class'].astype(int)
X_train_bow, X_test_bow, y_train, y_test = train_test_split(X, y, random_state=42, test_size=0.1)
support =LinearSVC(random_state=20)
support.fit(X_train_bow,y_train)
y_preds = support.predict(X_test_bow)
acc3=accuracy_score(y_test,y_preds)
report = classification_report( y_test, y_preds )
print(report)
print("SVM, Accuracy Score:" , acc3)
```

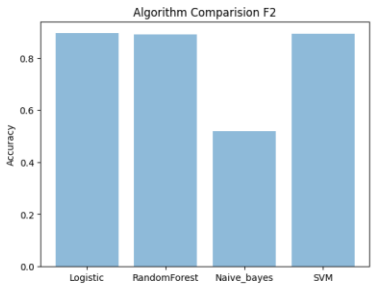
```
              precision    recall  f1-score   support

           0       0.61      0.24      0.34       164
           1       0.91      0.96      0.94      1905
           2       0.85      0.84      0.84       410

    accuracy                           0.89      2479
   macro avg       0.79      0.68      0.71      2479
weighted avg       0.88      0.89      0.88      2479

SVM, Accuracy Score: 0.8935054457442517
```

```
objects = ('Logistic', 'RandomForest', 'Naive_bayes', 'SVM')
```

0s completed at 10:30 AM

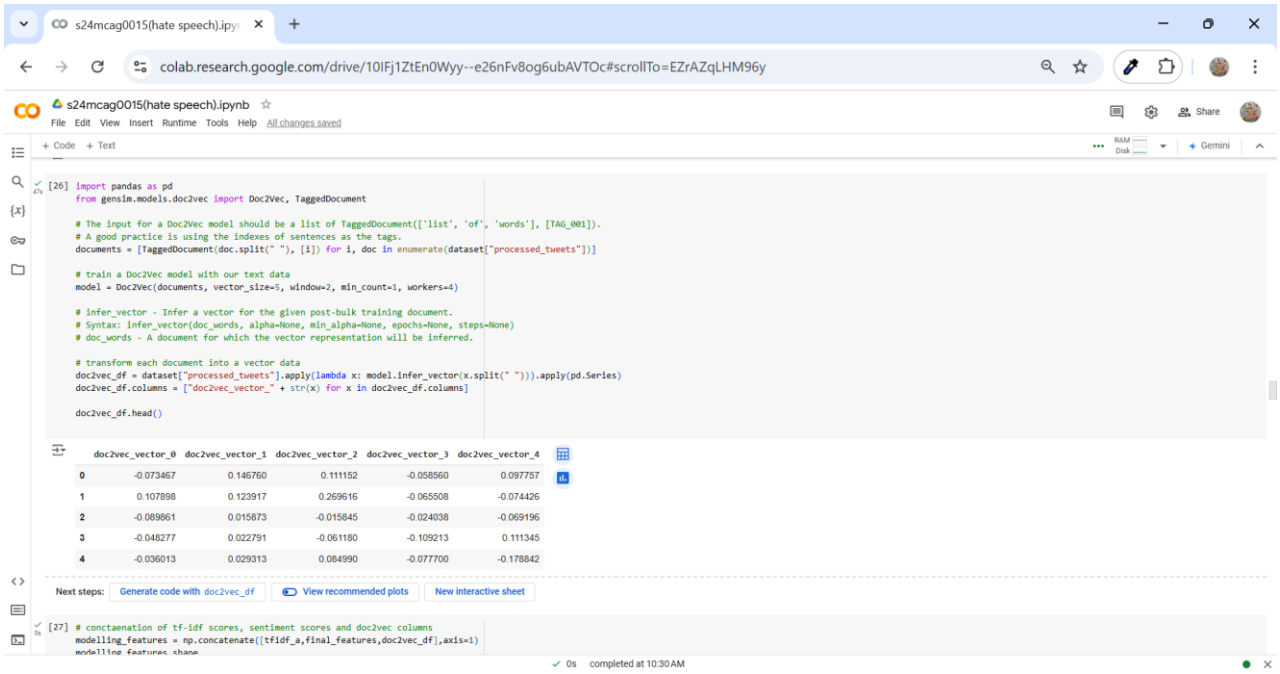
Algorithm Comparision F2

```
objects = ('Logistic', 'RandomForest', 'Naive_bayes', 'SVM')
y_pos = np.arange(len(objects))
performance = [acc,acc1,acc2,acc3]
plt.bar(y_pos, performance, align='center', alpha=0.5)
plt.xticks(y_pos, objects)
plt.ylabel('Accuracy')
plt.title('Algorithm Comparision F2')
plt.show()
```

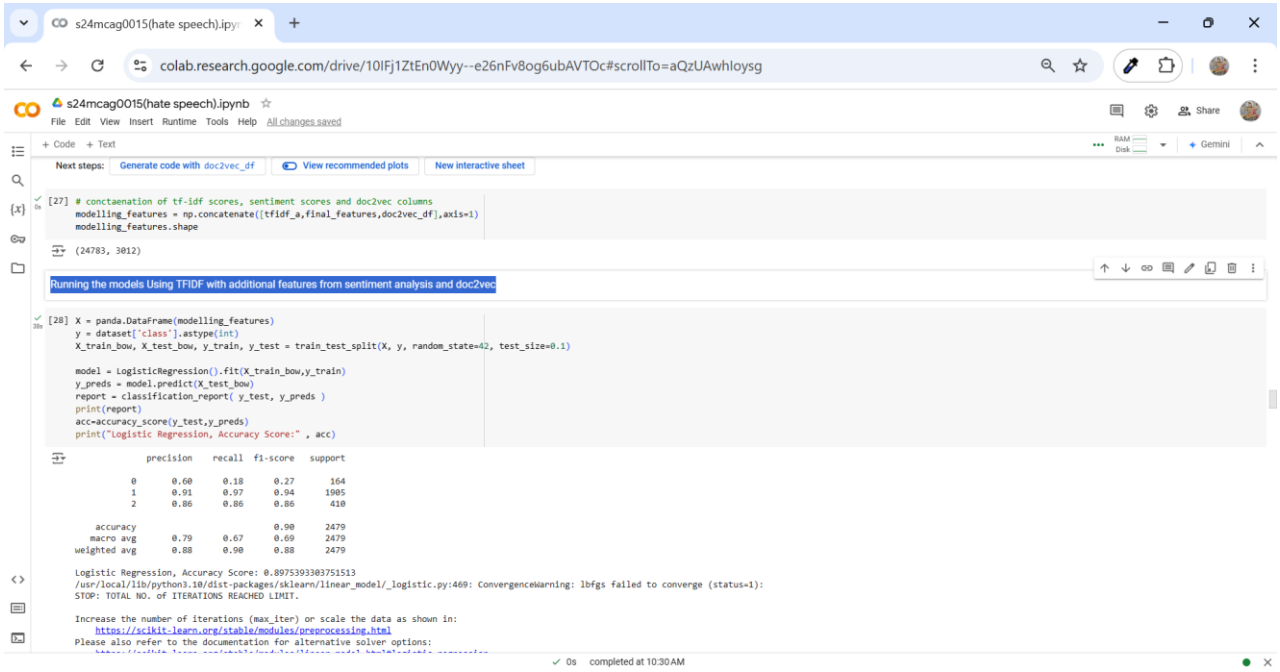


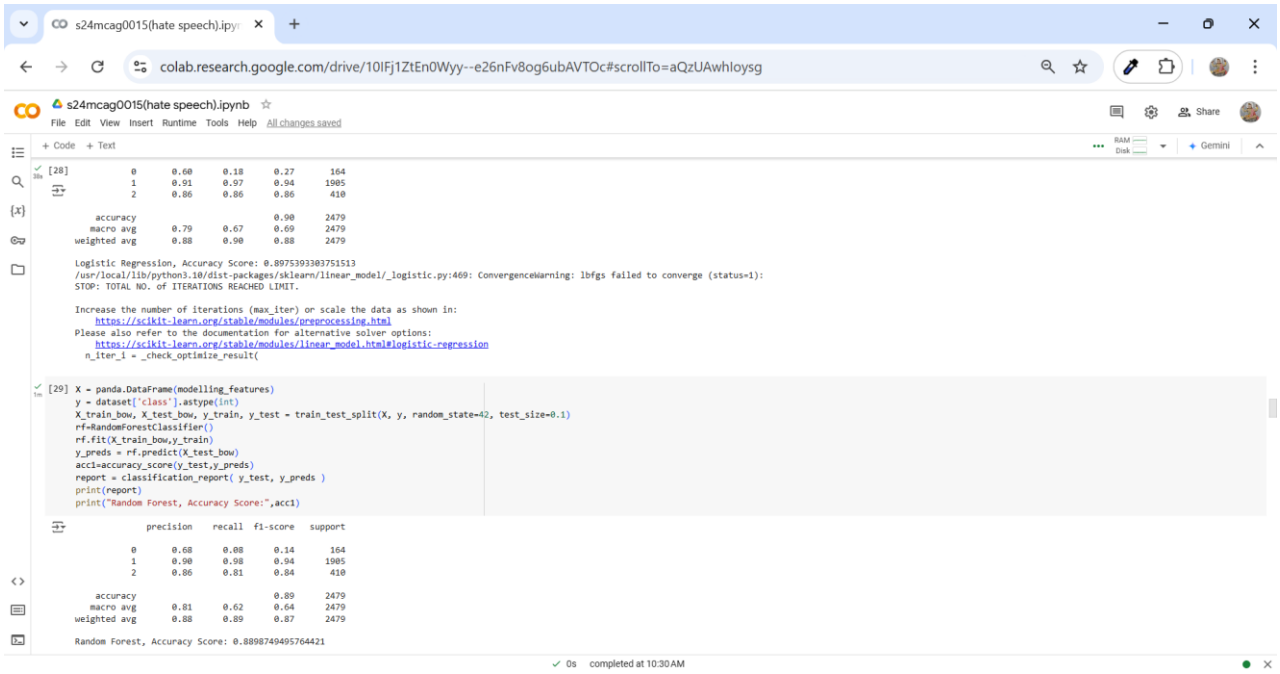

```
import pandas as pd
from gensim.models.doc2vec import Doc2Vec, TaggedDocument

# The input for a Doc2Vec model should be a list of TaggedDocument(['list', 'of', 'words'], [TAG_001]).
```

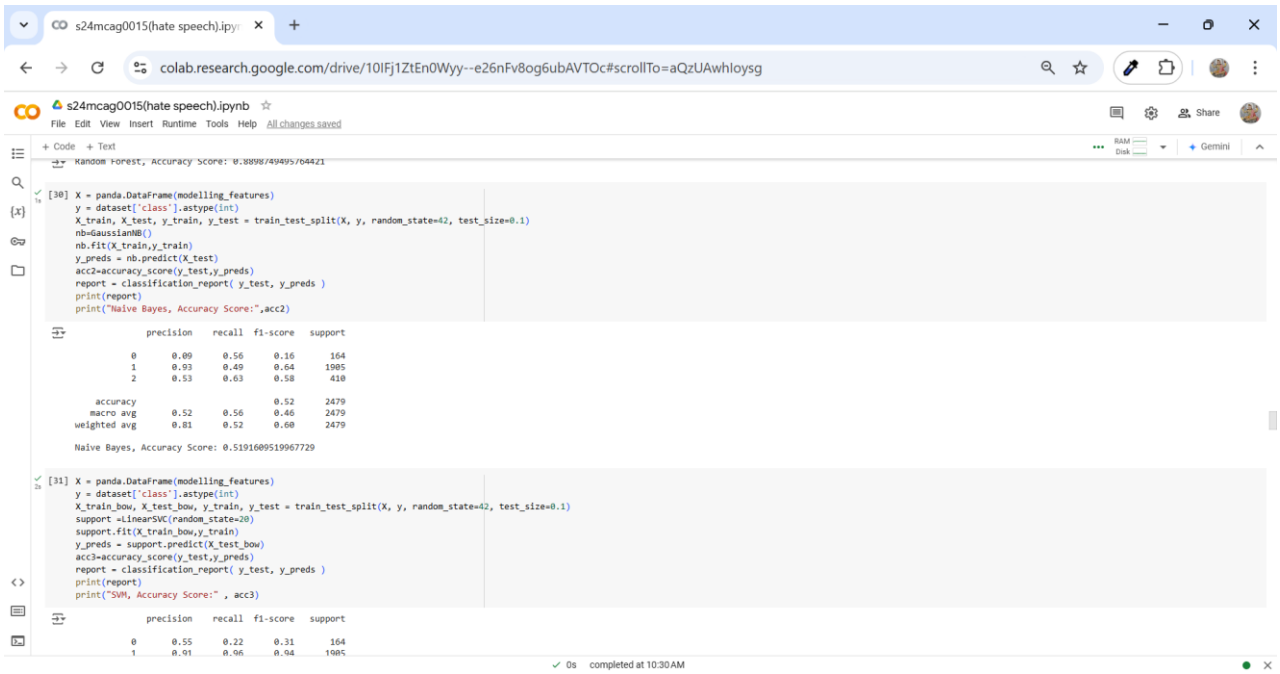
0s completed at 10:30 AM



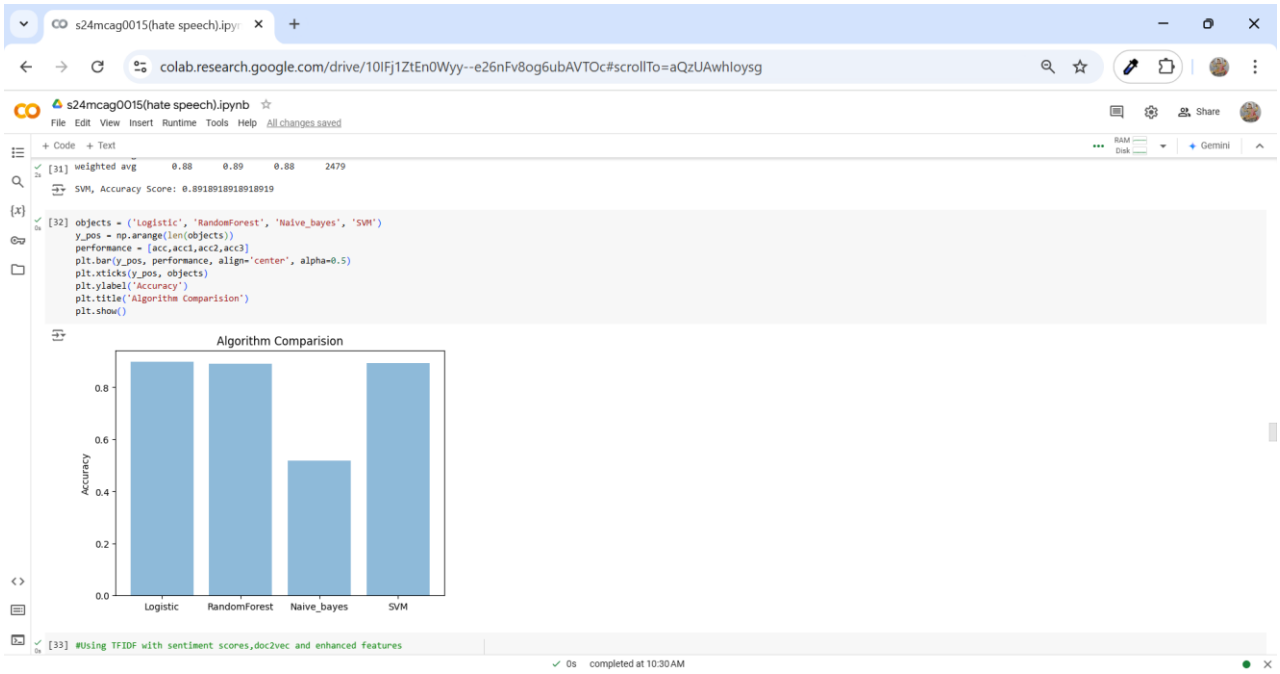
Running the models Using TFIDF with additional features from sentiment analysis and doc2vec



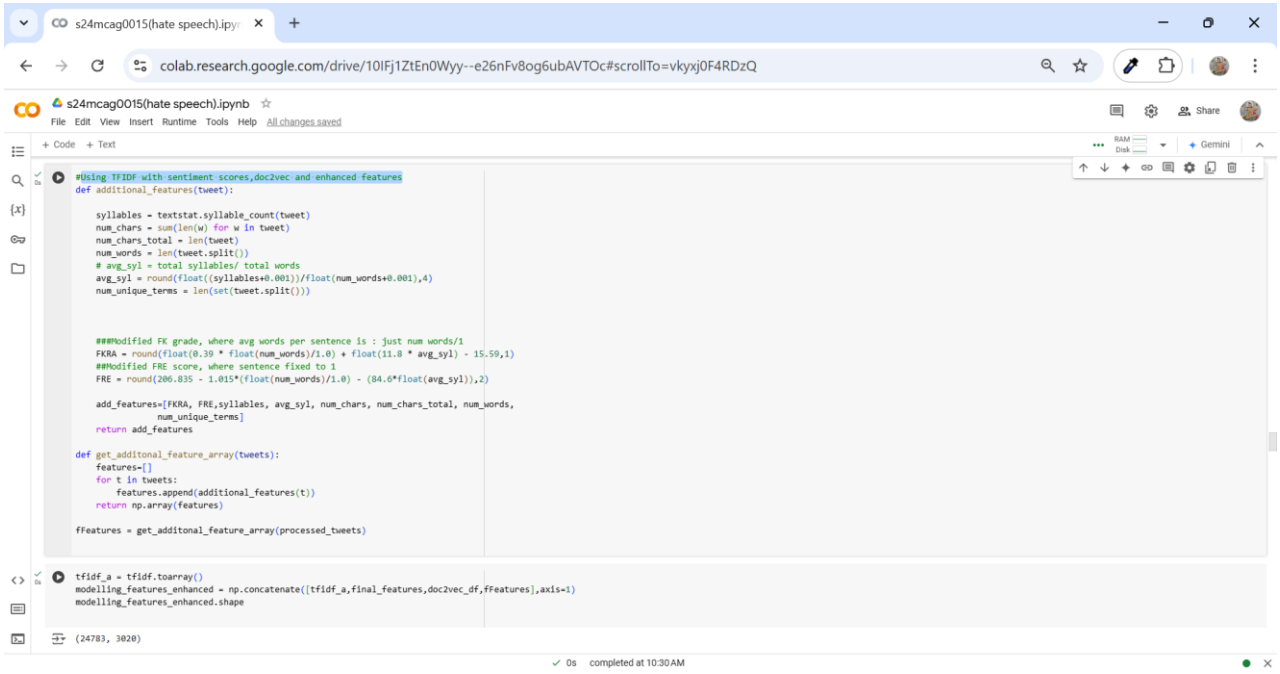




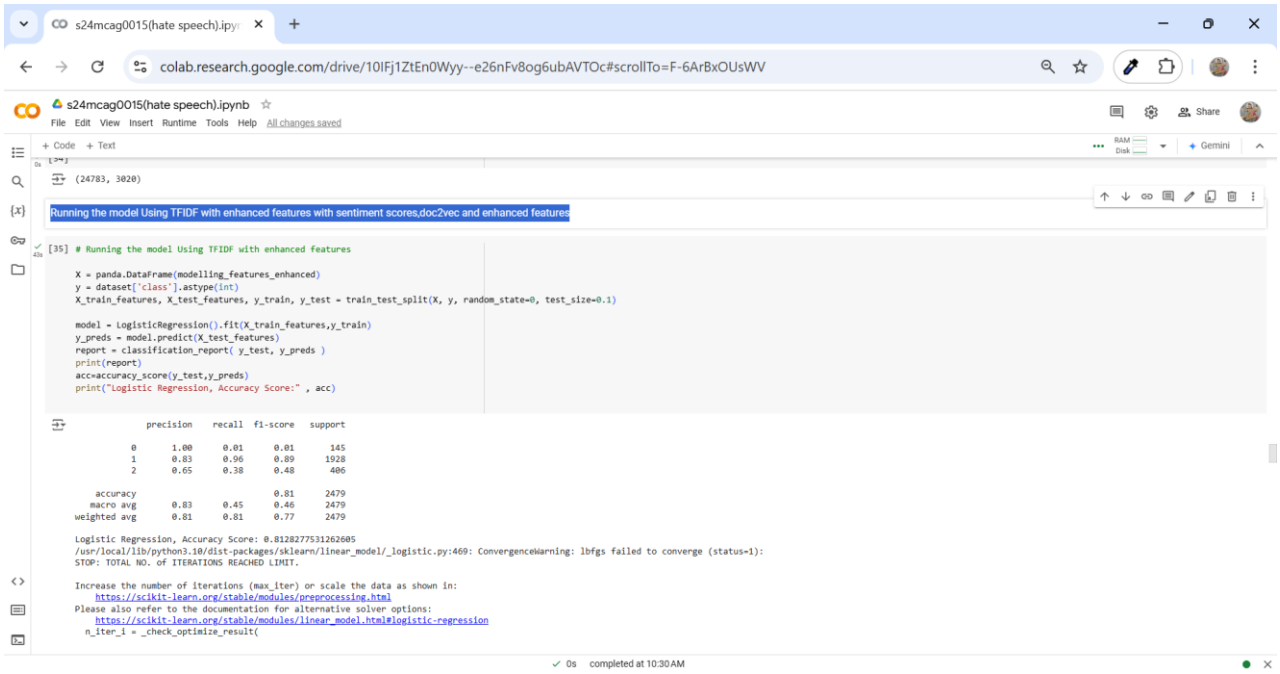
Algorithm Comparision



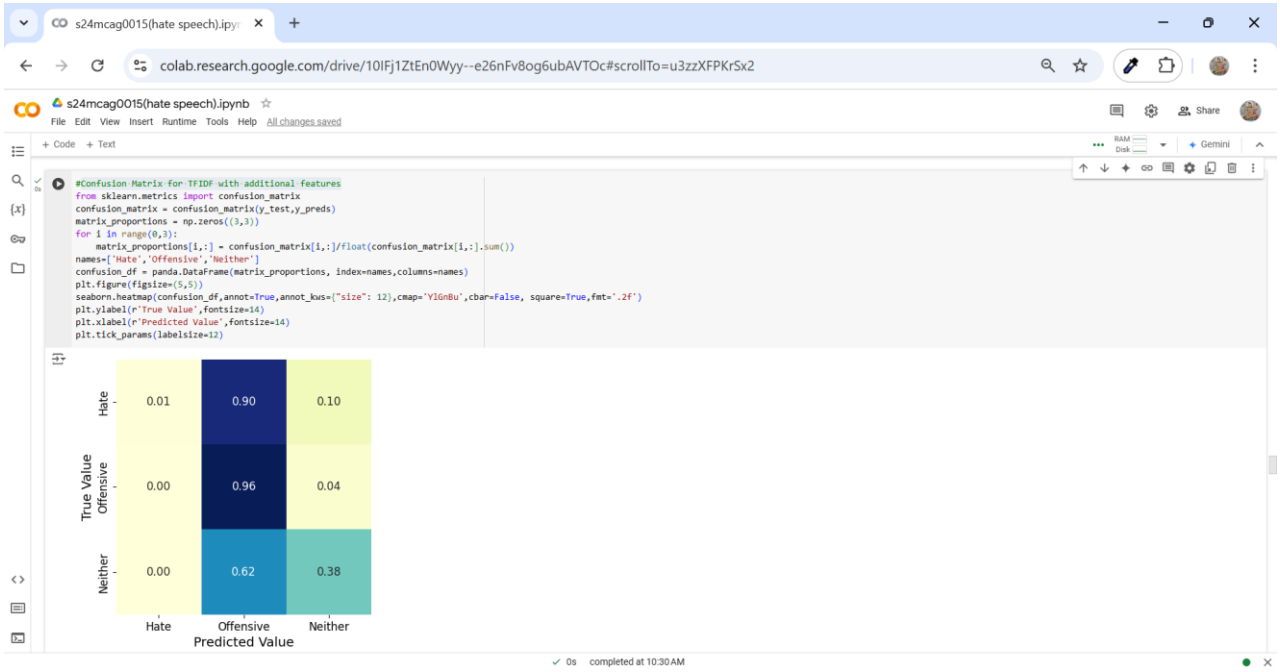
Using TFIDF with sentiment scores,doc2vec and enhanced features



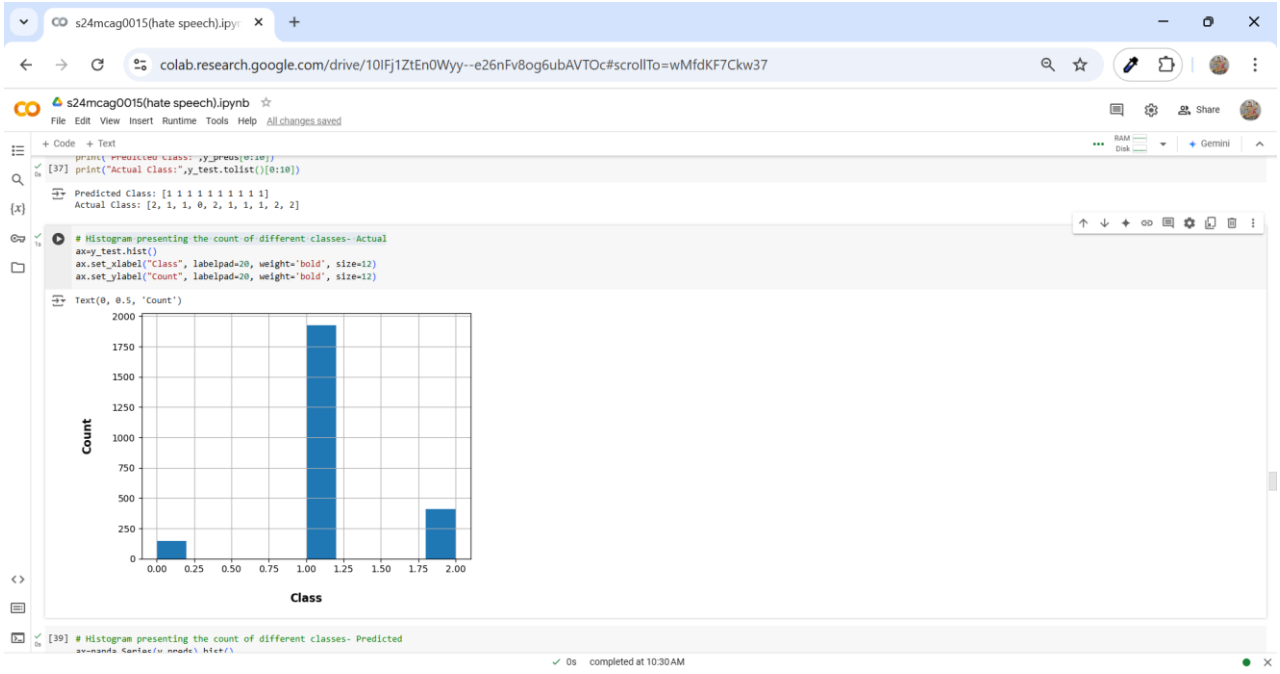
Running the model Using TFIDF with enhanced features with sentiment scores,doc2vec and enhanced features



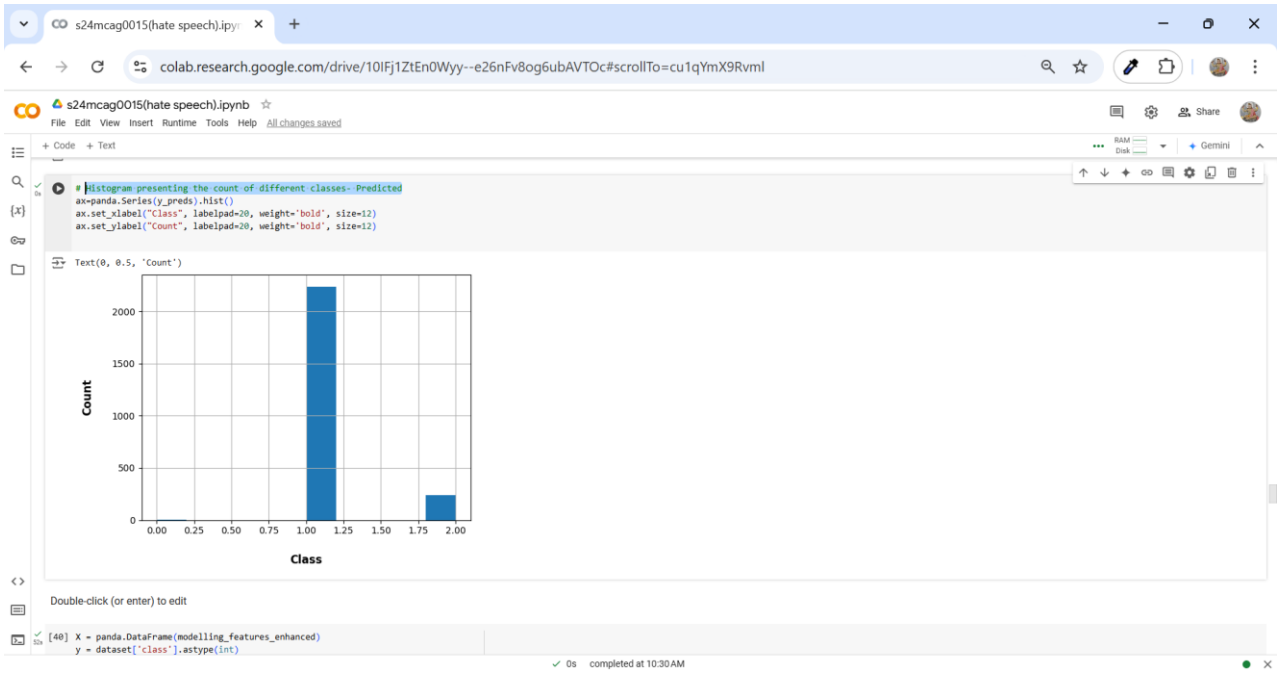
Confusion Matrix for TFIDF with additional features

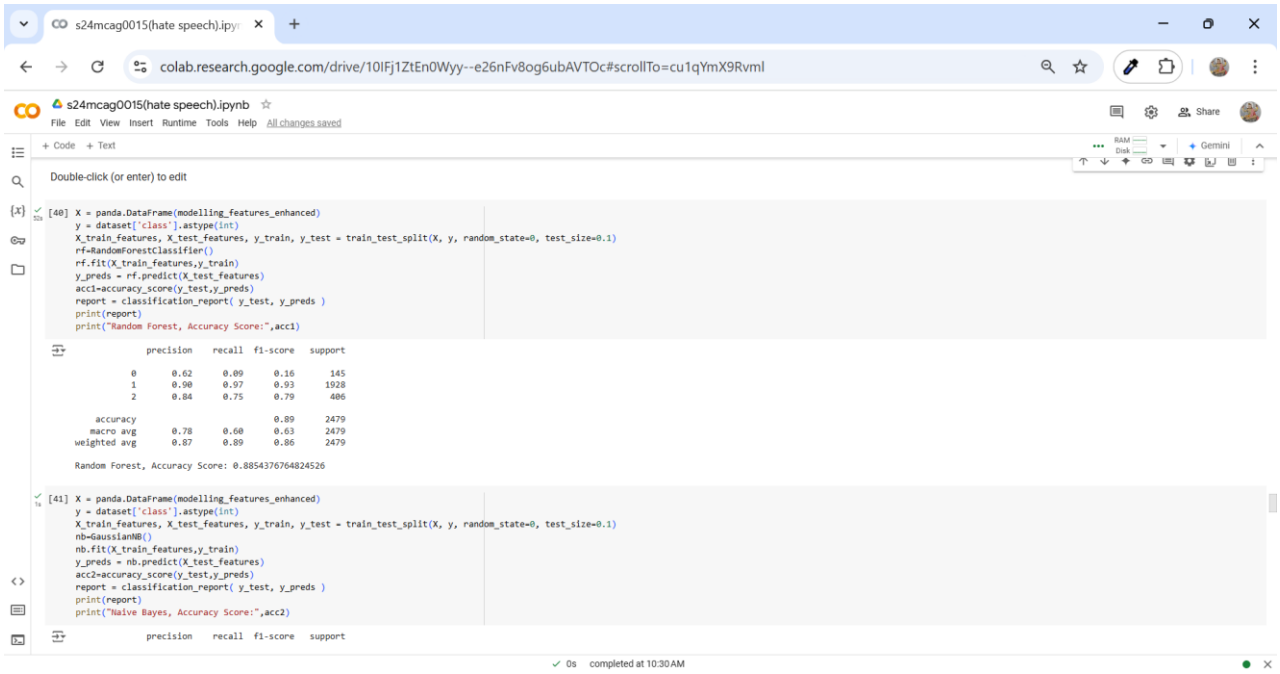


Histogram presenting the count of different classes- Actual

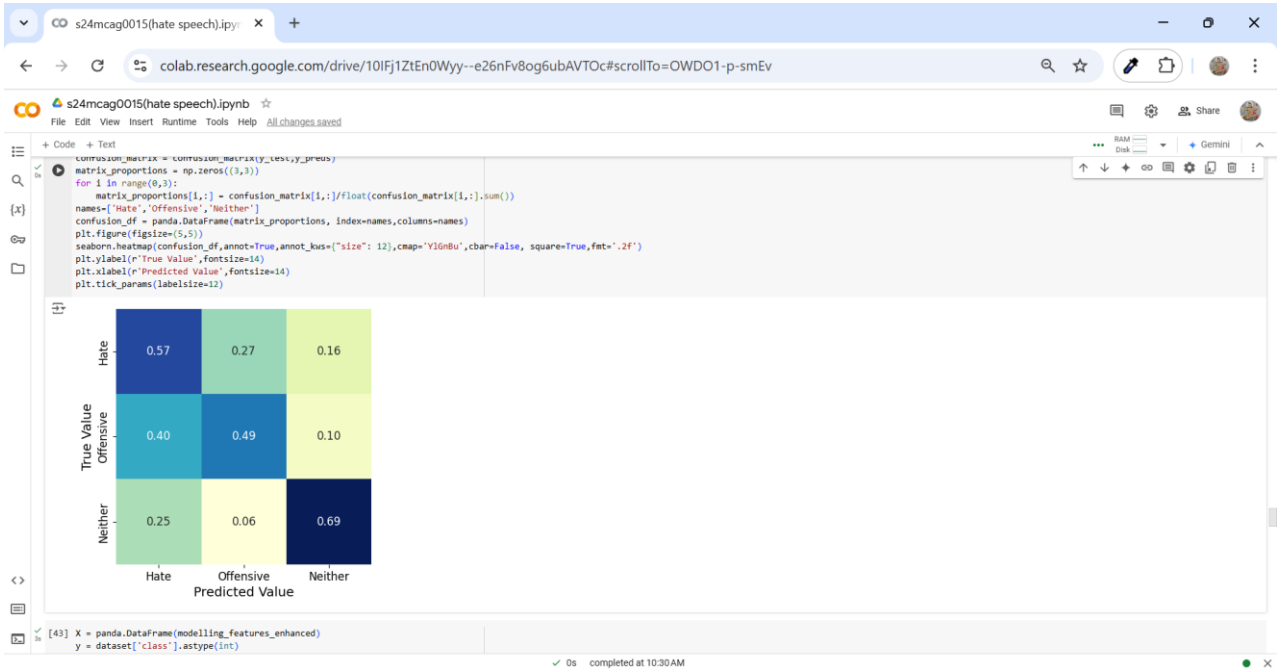


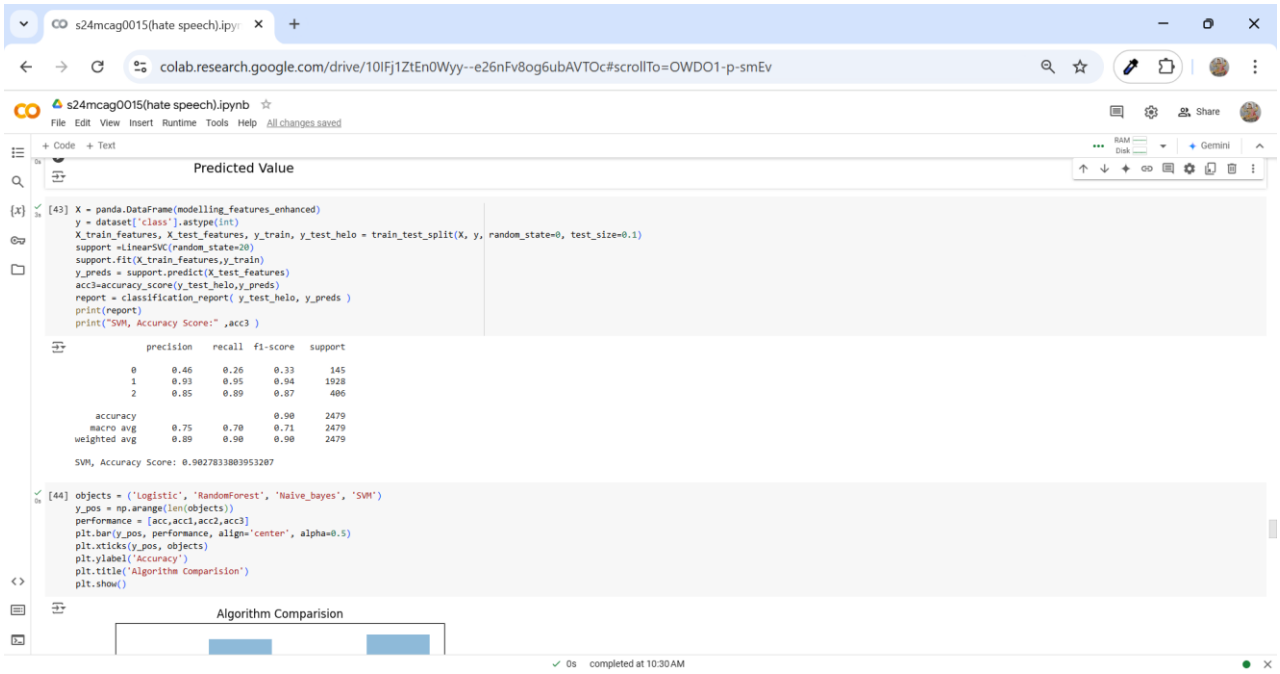
Histogram presenting the count of different classes- Predicted

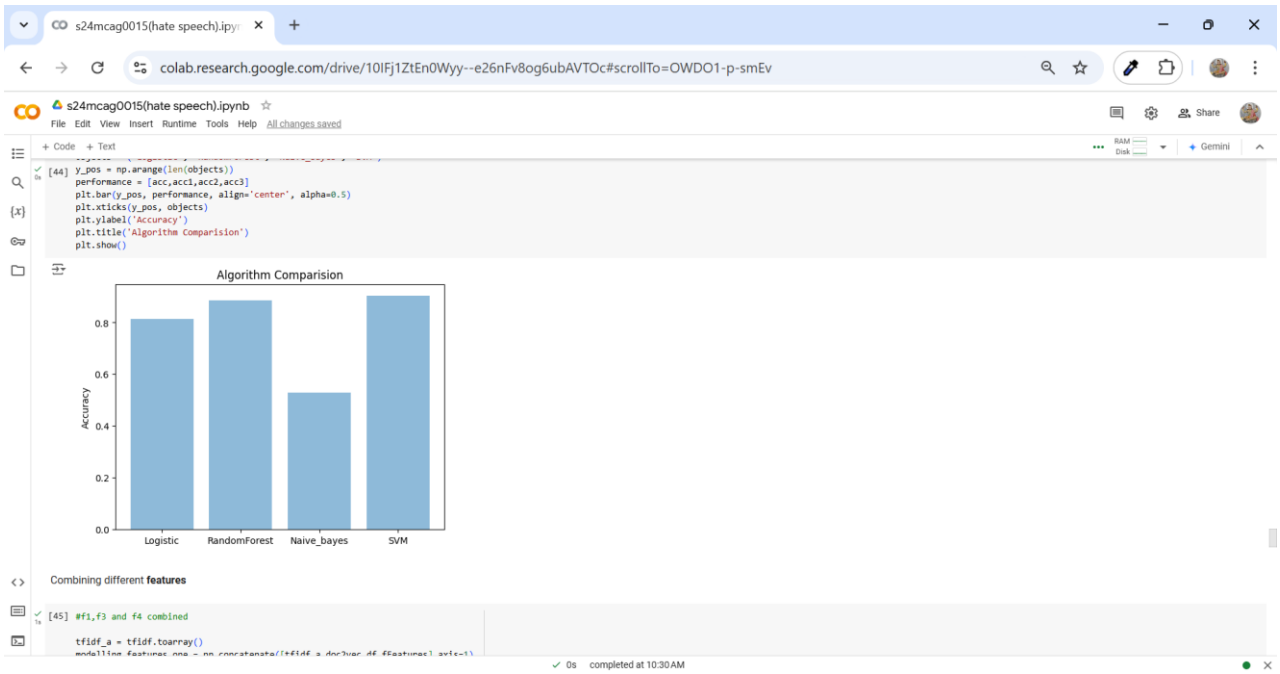




Confusion Matrix for TFIDF with additional features



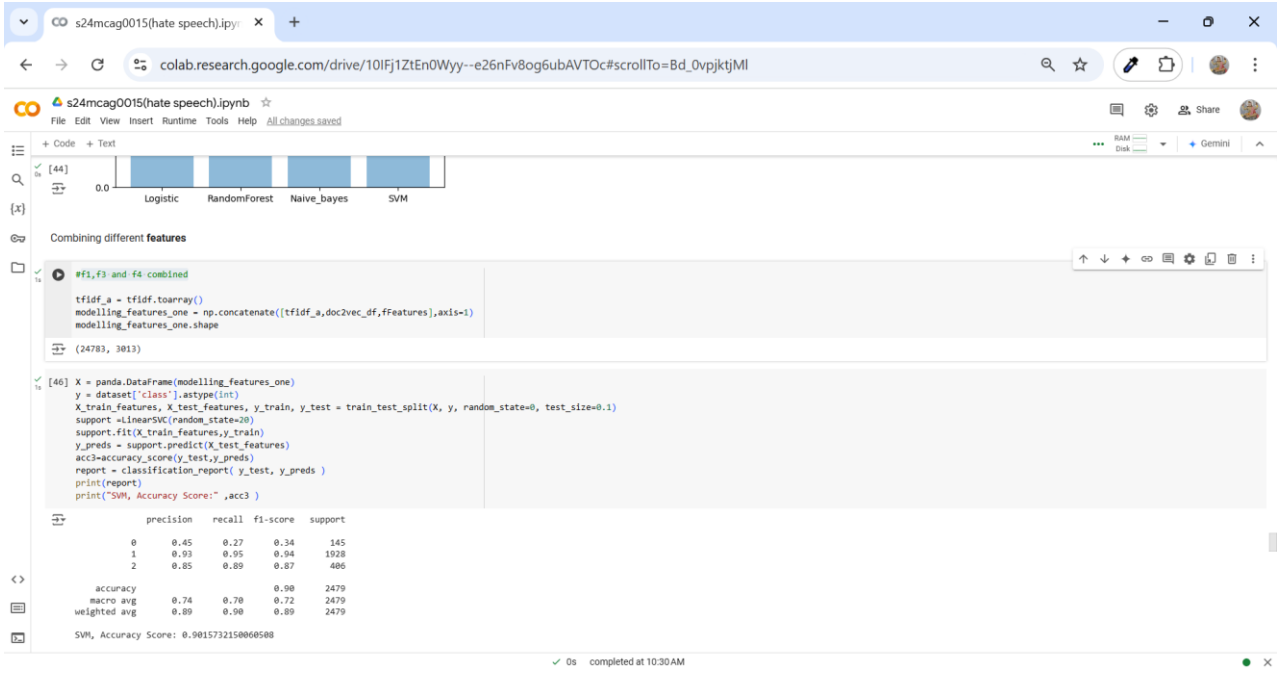


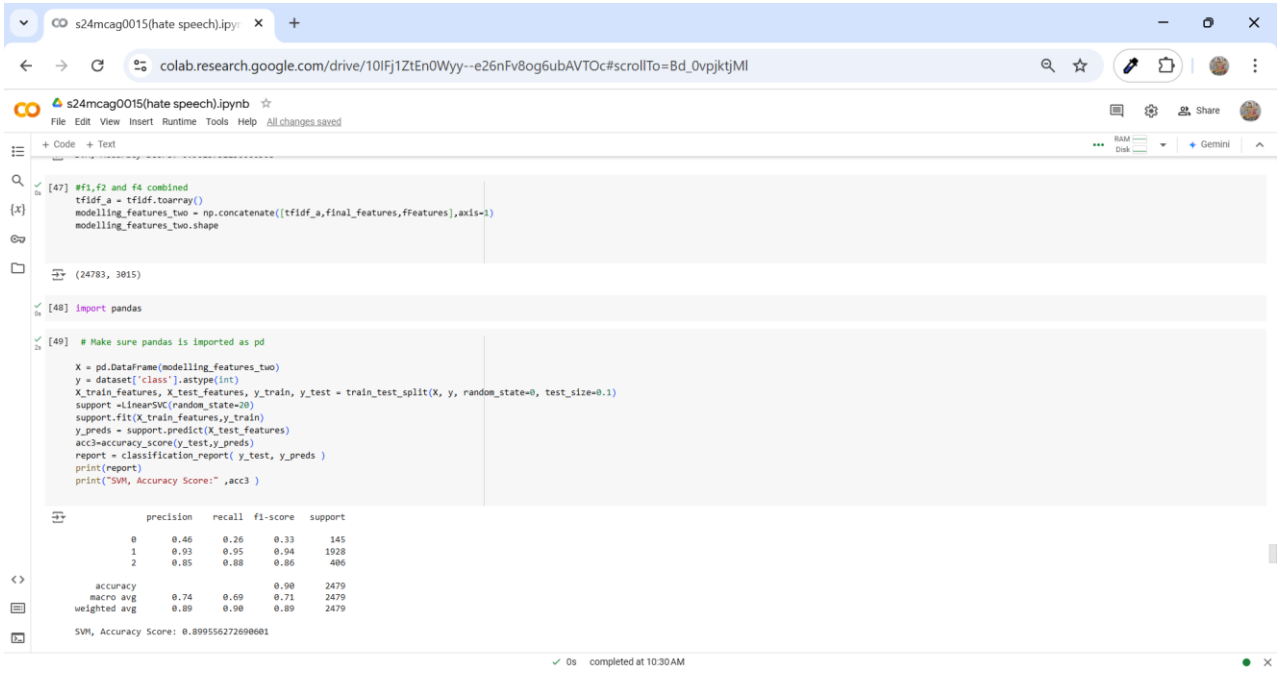


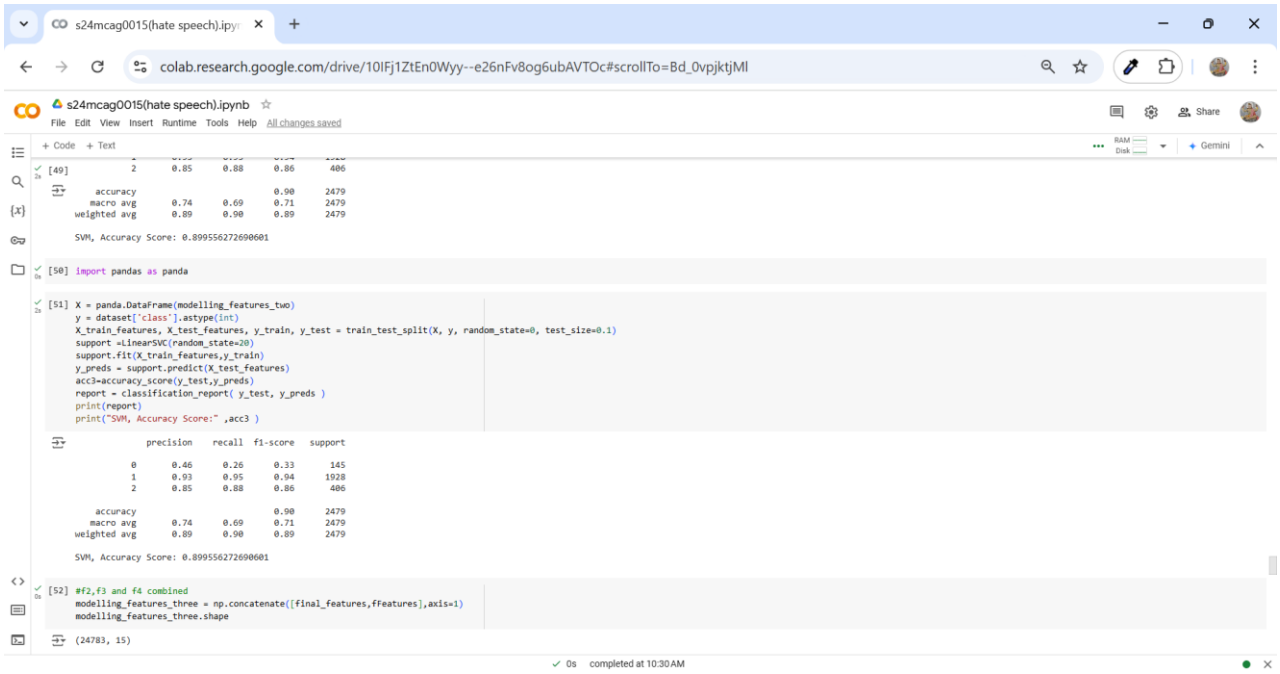
Modeling:

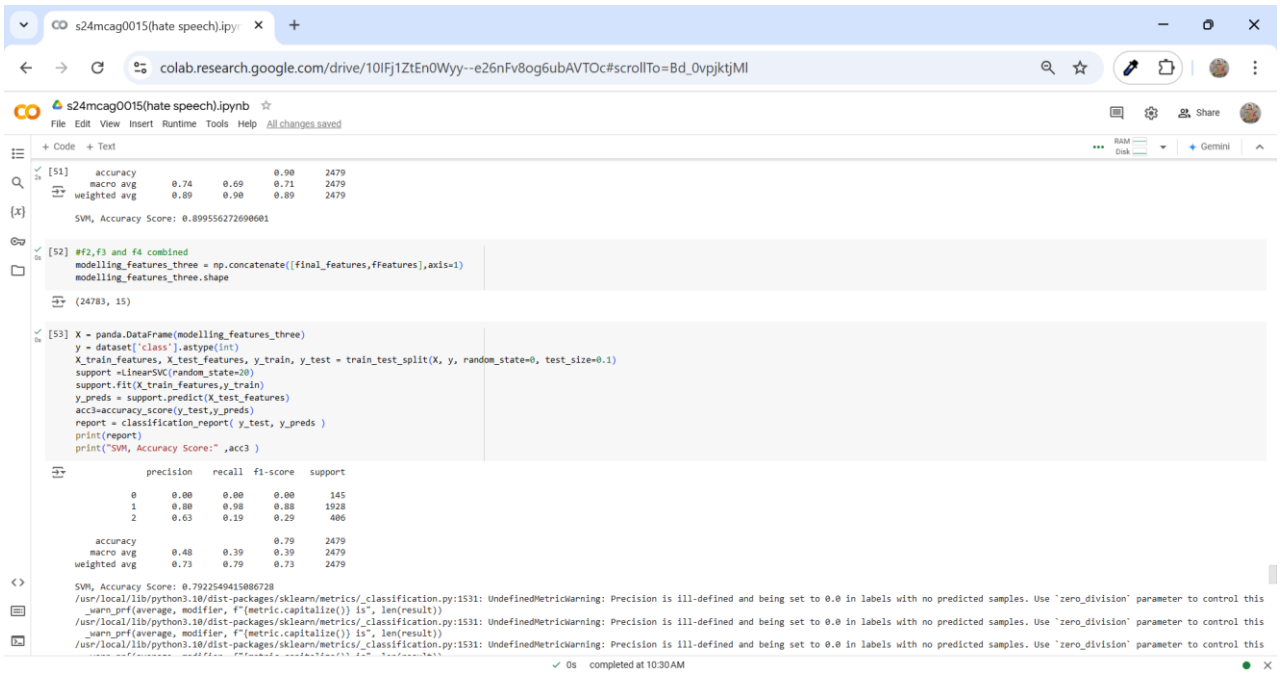
The researchers firstly utilized logistic regression with L1 regularization in order to cut down on the number of features. Then, various models have been tested, like logistic regression, naive Bayes, decision trees, random forests, and linear SVMs, by conducting 5-fold cross-validation. After trying a number of different classifiers and parameter settings, it was determined that logistic regression and linear SVMs were the best performing. Logistic regression with L2 regularization for the final model was selected. By virtue of working with these probabilities, they can analyse predicted probabilities for each class. The model, being "one-vs-rest," was trained on the entire dataset whereby one classifier was trained for each class and the class with the highest score from this classifier was used for predictions. The final model was built on a "one-vs.-rest" approach whereby each class had its own separate classifier trained against all others.

Combining features F1 F2 F3

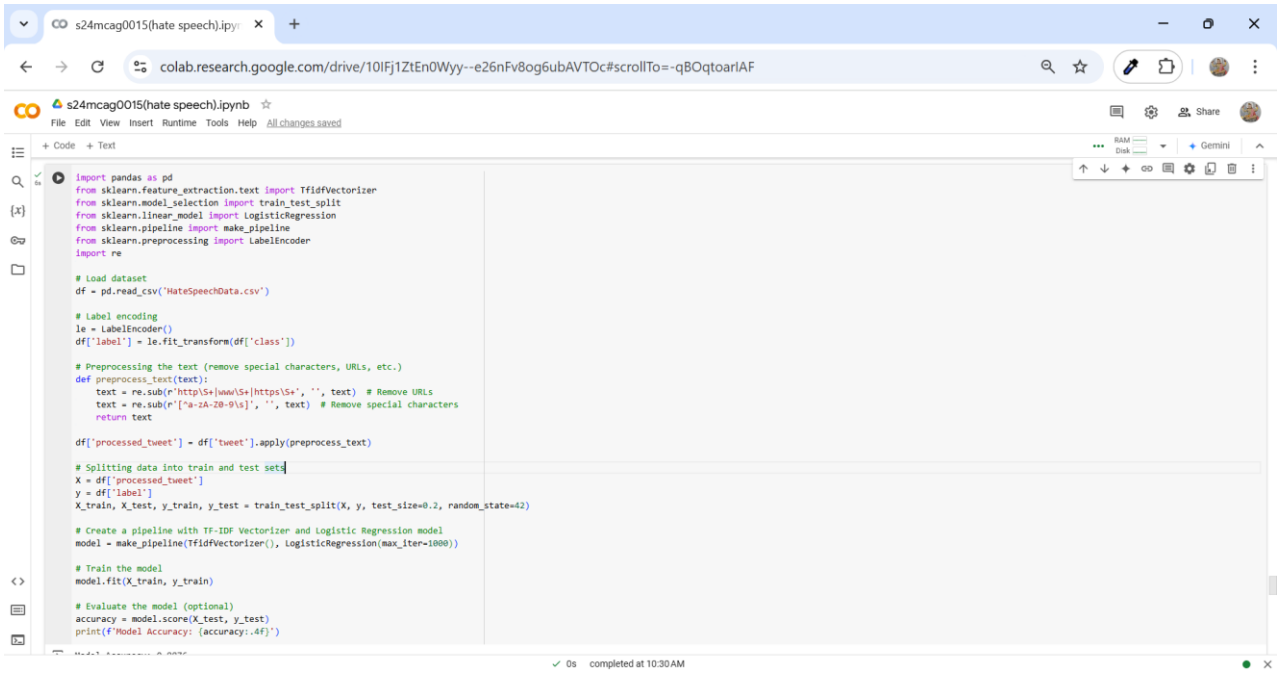


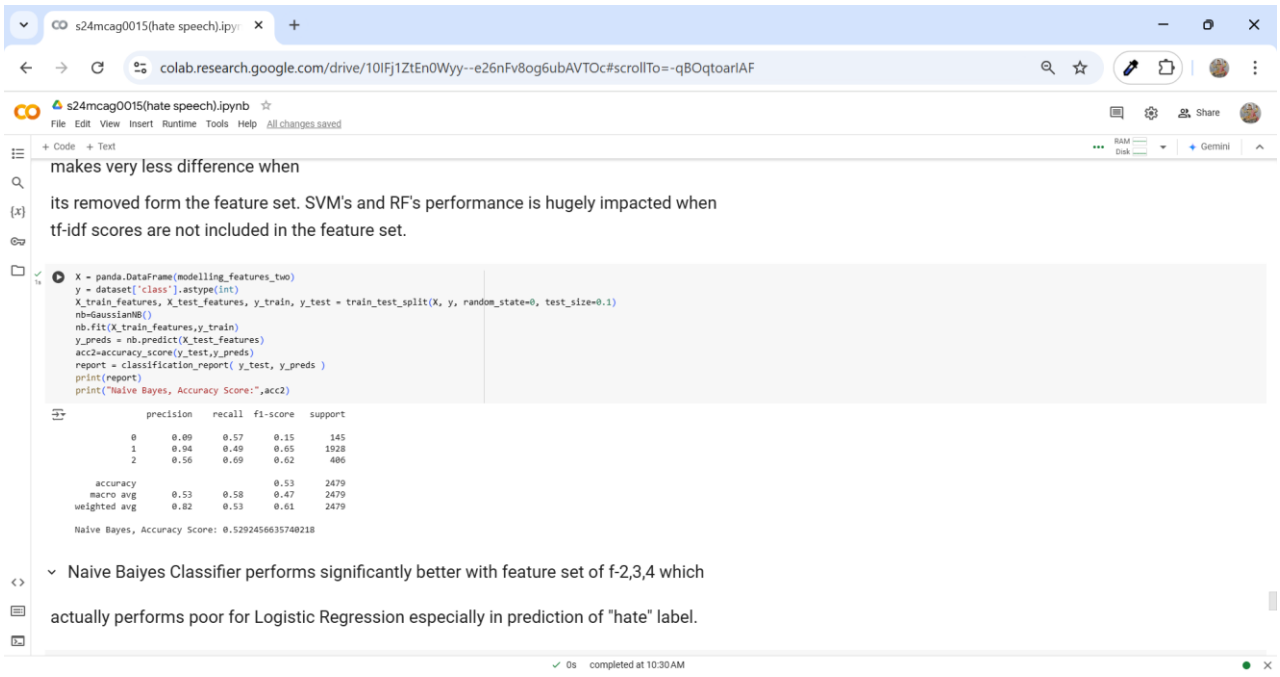


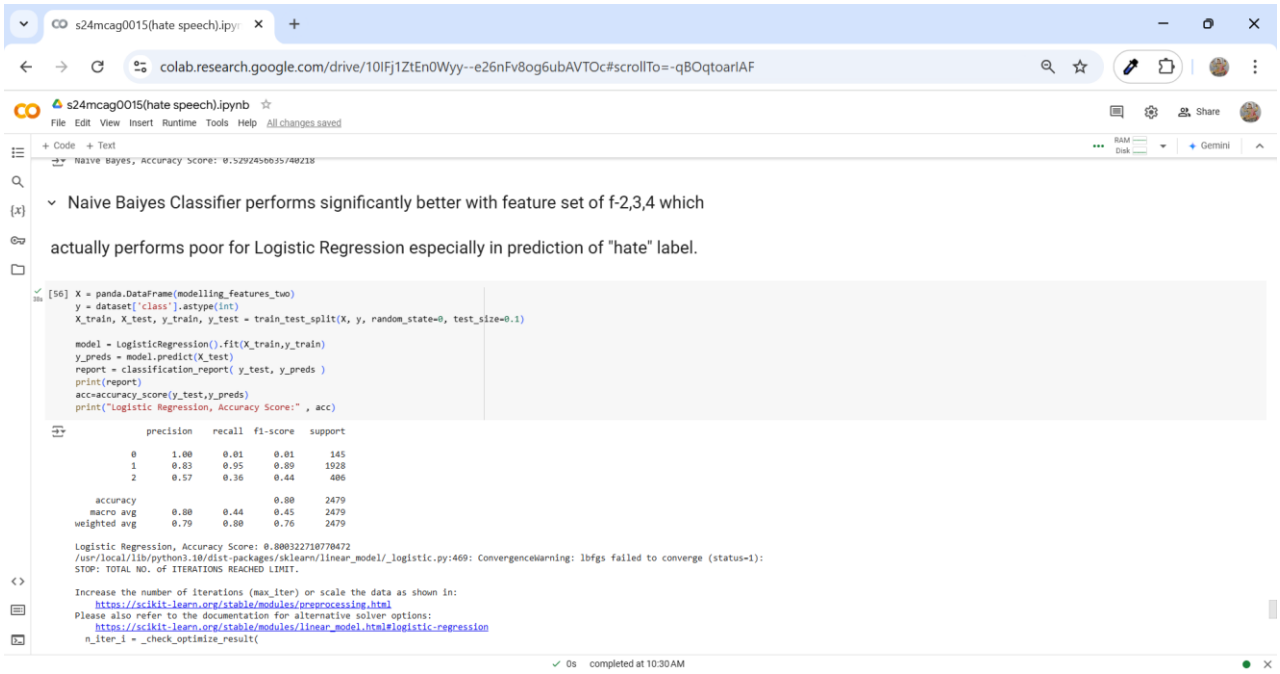

s24mcag0015(hate speech).ipyn
colab.research.google.com/drive/10IFj1ZtEn0Wyy--e26nFv8og6ubAVTOc#scrollTo=Bd_0vpjktjMl
s24mcag0015(hate speech).ipynb
File Edit View Insert Runtime Tools Help All changes saved
Share
+ Code + Text
Gemini
[49] 2 0.85 0.88 0.86 406
accuracy 0.90 2479
macro avg 0.74 0.69 0.71 2479
weighted avg 0.89 0.90 0.89 2479
SVM, Accuracy Score: 0.899556272690601
[50] import pandas as panda
[51] X = panda.DataFrame(modelling_features_two)
y = dataset['class'].astype(int)
X_train_features, X_test_features, y_train, y_test = train_test_split(X, y, random_state=0, test_size=0.1)
support =LinearSVC(random_state=20)
support.fit(X_train_features,y_train)
y_preds = support.predict(X_test_features)
acc3=accuracy_score(y_test,y_preds)
report = classification_report(y_test, y_preds)
print(report)
print("SVM, Accuracy Score:" ,acc3)
precision recall f1-score support
0 0.46 0.26 0.33 145
1 0.93 0.95 0.94 1928
2 0.85 0.88 0.86 406
accuracy 0.90 2479
macro avg 0.74 0.69 0.71 2479
weighted avg 0.89 0.90 0.89 2479
SVM, Accuracy Score: 0.899556272690601
[52] #f2,f3 and f4 combined
modelling_features_three = np.concatenate([final_features,fFeatures],axis=1)
modelling_features_three.shape
(24783, 15)
0s completed at 10:30 AM

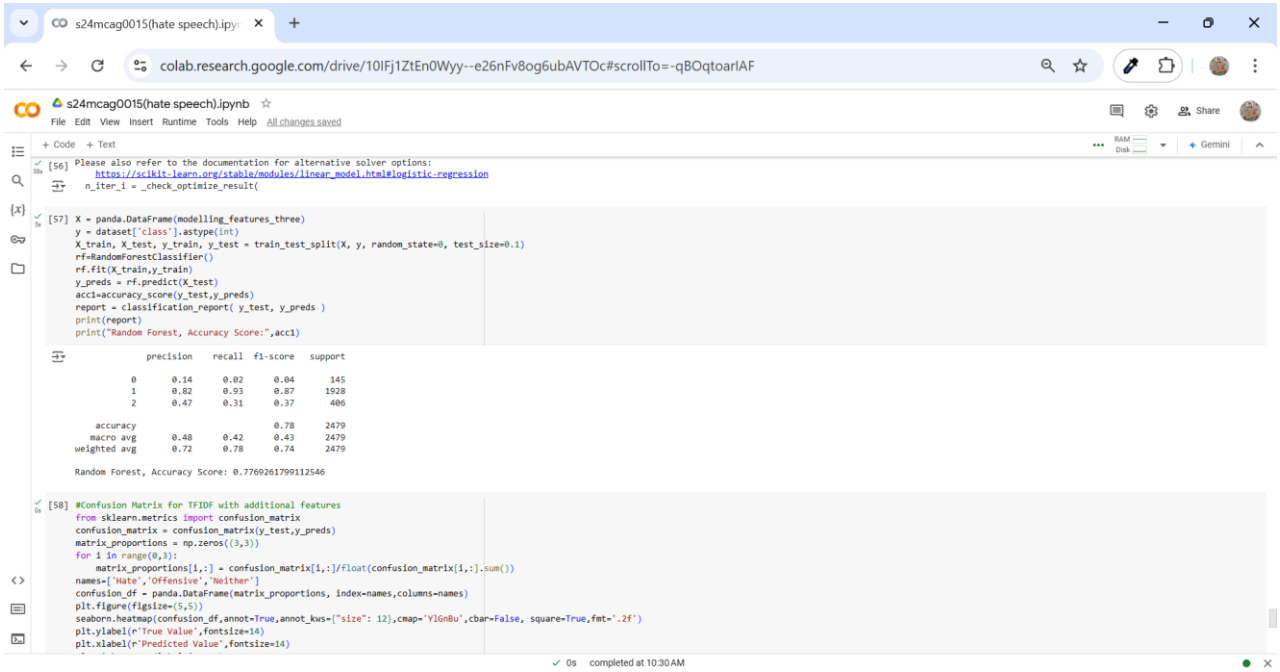

s24mcag0015(hate speech).ipyn
colab.research.google.com/drive/10IFj1ZtEn0Wyy--e26nFv8og6ubAVTOc#scrollTo=Bd_0vpjktjMl
s24mcag0015(hate speech).ipynb
File Edit View Insert Runtime Tools Help All changes saved
Share
+ Code + Text
Gemini
[51] accuracy 0.90 2479
macro avg 0.74 0.69 0.71 2479
weighted avg 0.89 0.90 0.89 2479
SVM, Accuracy Score: 0.899556272690601
[52] #f2,f3 and f4 combined
modelling_features_three = np.concatenate([final_features,fFeatures],axis=1)
modelling_features_three.shape
(24783, 15)
[53] X = panda.DataFrame(modelling_features_three)
y = dataset['class'].astype(int)
X_train_features, X_test_features, y_train, y_test = train_test_split(X, y, random_state=0, test_size=0.1)
support =LinearSVC(random_state=20)
support.fit(X_train_features,y_train)
y_preds = support.predict(X_test_features)
acc3=accuracy_score(y_test,y_preds)
report = classification_report(y_test, y_preds)
print(report)
print("SVM, Accuracy Score:" ,acc3)
precision recall f1-score support
0 0.00 0.00 0.00 145
1 0.80 0.98 0.88 1928
2 0.63 0.19 0.29 406
accuracy 0.79 2479
macro avg 0.48 0.39 0.39 2479
weighted avg 0.73 0.79 0.73 2479
SVM, Accuracy Score: 0.7922549415086728
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this
0s completed at 10:30 AM

Final Model

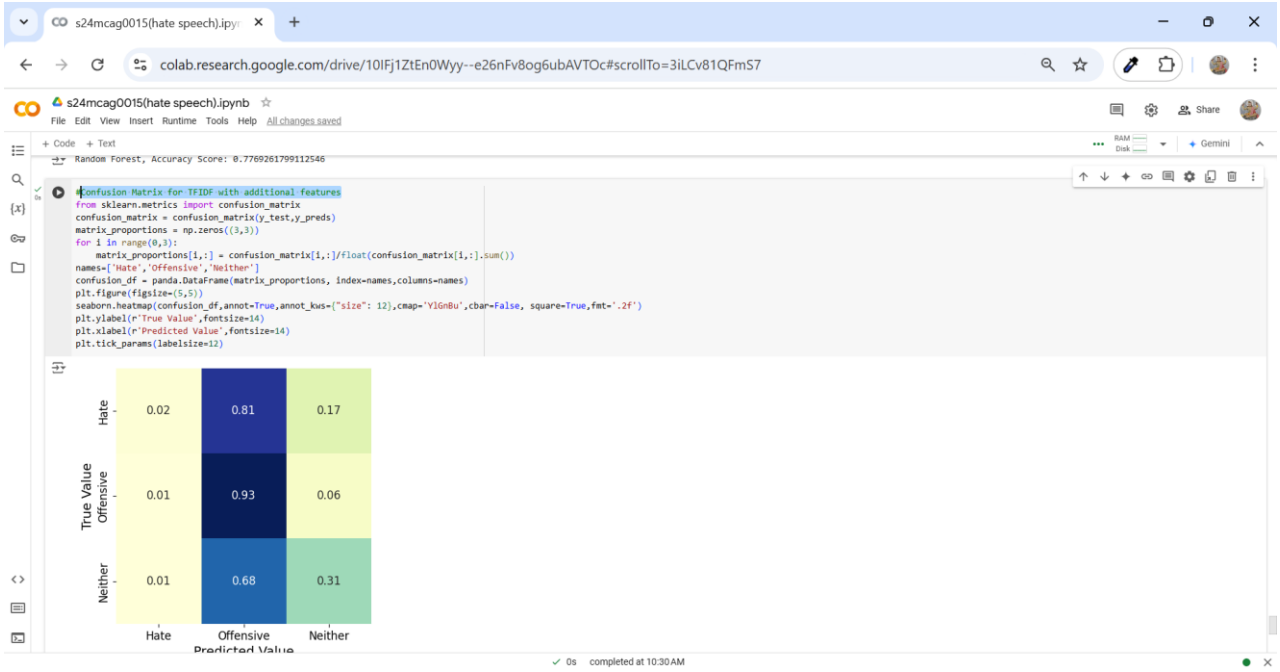

s24mcag0015(hate speech).ipyn
colab.research.google.com/drive/10IFj1ZtEn0Wyy--e26nFv8og6ubAVTOc#scrollTo=-qBOqtoarIAF
s24mcag0015(hate speech).ipynb
File Edit View Insert Runtime Tools Help All changes saved
Share
Gemini
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import LabelEncoder
import re
# Load dataset
df = pd.read_csv('HateSpeechData.csv')
# Label encoding
le = LabelEncoder()
df['label'] = le.fit_transform(df['class'])
# Preprocessing the text (remove special characters, URLs, etc.)
def preprocess_text(text):
text = re.sub(r'http\S+|www\S+|https\S+', '', text) # Remove URLs
text = re.sub(r'[^a-zA-Z0-9\s]', '', text) # Remove special characters
return text
df['processed_tweet'] = df['tweet'].apply(preprocess_text)
# Splitting data into train and test sets
X = df['processed_tweet']
y = df['label']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Create a pipeline with TF-IDF Vectorizer and Logistic Regression model
model = make_pipeline(TfidfVectorizer(), LogisticRegression(max_iter=1000))
# Train the model
model.fit(X_train, y_train)
# Evaluate the model (optional)
accuracy = model.score(X_test, y_test)
print(f'Model Accuracy: {accuracy:.4f}')
0s completed at 10:30 AM


s24mcag0015(hate speech).ipyn
colab.research.google.com/drive/10IFj1ZtEn0Wyy--e26nFv8og6ubAVTOc#scrollTo=-qBOqtoarIAF
s24mcag0015(hate speech).ipynb
File Edit View Insert Runtime Tools Help All changes saved
Share
Gemini
makes very less difference when
its removed form the feature set. SVM's and RF's performance is hugely impacted when tf-idf scores are not included in the feature set.
X = panda.DataFrame(modelling_features_two)
y = dataset['class'].astype(int)
X_train_features, X_test_features, y_train, y_test = train_test_split(X, y, random_state=0, test_size=0.1)
nb=GaussianNB()
nb.fit(X_train_features,y_train)
y_preds = nb.predict(X_test_features)
acc2=accuracy_score(y_test,y_preds)
report = classification_report(y_test, y_preds)
print(report)
print("Naive Bayes, Accuracy Score:",acc2)
precision recall f1-score support
0 0.09 0.57 0.15 145
1 0.94 0.49 0.65 1928
2 0.56 0.69 0.62 406
accuracy 0.53 2479
macro avg 0.53 0.58 0.47 2479
weighted avg 0.82 0.53 0.61 2479
Naive Bayes, Accuracy Score: 0.5292456635740218
Naive Baiyes Classifier performs significantly better with feature set of f-2,3,4 which
actually performs poor for Logistic Regression especially in prediction of "hate" label.
0s completed at 10:30 AM


s24mcag0015(hate speech).ipynb
colab.research.google.com/drive/10IFj1ZtEn0Wyy--e26nFv8og6ubAVTOc#scrollTo=-qBOqtoarIAF
File Edit View Insert Runtime Tools Help All changes saved
Share
+ Code + Text
Gemini
Naive Bayes, Accuracy Score: 0.5292456635740218
Naive Baiyes Classifier performs significantly better with feature set of f-2,3,4 which
actually performs poor for Logistic Regression especially in prediction of "hate" label.
[56] X = panda.DataFrame(modelling_features_two)
y = dataset['class'].astype(int)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0, test_size=0.1)
model = LogisticRegression().fit(X_train,y_train)
y_preds = model.predict(X_test)
report = classification_report(y_test, y_preds)
print(report)
acc=accuracy_score(y_test,y_preds)
print("Logistic Regression, Accuracy Score:" , acc)
precision recall f1-score support
0 1.00 0.01 0.01 145
1 0.83 0.95 0.89 1928
2 0.57 0.36 0.44 406
accuracy 0.80 2479
macro avg 0.80 0.44 0.45 2479
weighted avg 0.79 0.80 0.76 2479
Logistic Regression, Accuracy Score: 0.8003227107704 72
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:469: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
Increase the number of iterations (max_iter) or scale the data as shown in:
https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
n_iter_i = _check_optimize_result(
0s completed at 10:30 AM

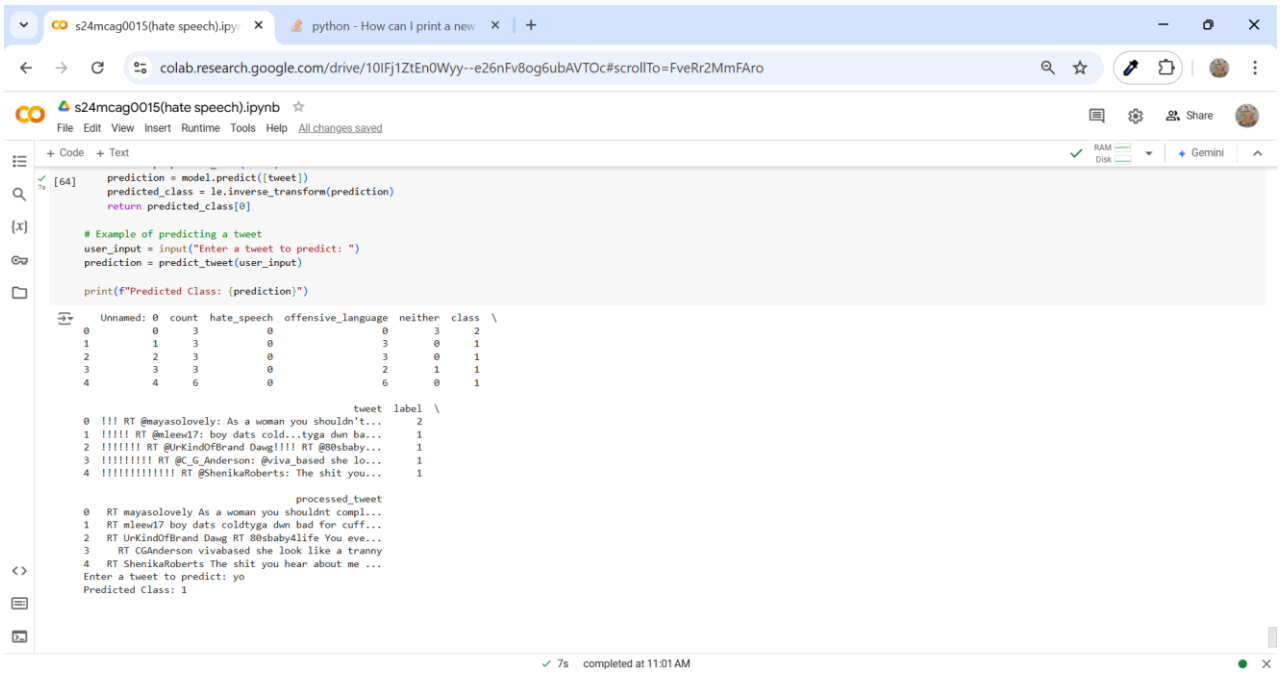

s24mcag0015(hate speech).ipynb
colab.research.google.com/drive/10IFj1ZtEn0Wyy--e26nFv8og6ubAVTOc#scrollTo=-qBOqtoarIAF
File Edit View Insert Runtime Tools Help All changes saved
Share
+ Code + Text
Gemini
[56] Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
n_iter_i = _check_optimize_result(
[57] X = panda.DataFrame(modelling_features_three)
y = dataset['class'].astype(int)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0, test_size=0.1)
rf=RandomForestClassifier()
rf.fit(X_train,y_train)
y_preds = rf.predict(X_test)
acc1=accuracy_score(y_test,y_preds)
report = classification_report(y_test, y_preds)
print(report)
print("Random Forest, Accuracy Score:",acc1)
precision recall f1-score support
0 0.14 0.02 0.04 145
1 0.82 0.93 0.87 1928
2 0.47 0.31 0.37 406
accuracy 0.78 2479
macro avg 0.48 0.42 0.43 2479
weighted avg 0.72 0.78 0.74 2479
Random Forest, Accuracy Score: 0.7769261799112546
[58] #Confusion Matrix for TFIDF with additional features
from sklearn.metrics import confusion_matrix
confusion_matrix = confusion_matrix(y_test,y_preds)
matrix_proportions = np.zeros((3,3))
for i in range(0,3):
matrix_proportions[i,:] = confusion_matrix[i,:]/float(confusion_matrix[i,:].sum())
names=['Hate','Offensive','Neither']
confusion_df = panda.DataFrame(matrix_proportions, index=names,columns=names)
plt.figure(figsize=(5,5))
seaborn.heatmap(confusion_df,annot=True,annot_kws={"size": 12},cmap='YlGnBu',cbar=False, square=True,fmt='.2f')
plt.ylabel(r'True Value',fontsize=14)
plt.xlabel(r'Predicted Value',fontsize=14)
0s completed at 10:30 AM

Confusion Matrix for TFIDF with additional features



Results:

The final model achieved a precision of 0.91, a recall of 0.90, and an F1 score of 0.90, but problems of misclassification occurred. About 40% of hate speech tweets were misclassified, with the hate speech category having lower precision and recall compared to offensive tweets. Indeed, most of these misclassifications arose because of the model's determination that tweets were not as hateful or offensive as human judges decided. Thus, tweets that should have been considered hate speech were labeled offensive or tweets that were borderline being classified as hate speech.



Analysis of Misclassified Tweets:

Tweets with a high predicted probability of hate speech often contained racial or homophobic slurs. However, some tweets were labeled offensive but not hate speech due to the context in which they

appeared, such as tweets that criticized racism, even if the message contained slurs. For example, some tweets that did not contain any slurs but were hateful—anti-immigrant speech or sex-discriminatory anecdotes—probably misclassified as neither offensive nor hate speech. The model detected the most common forms of hate speech such as racism and homophobia comparatively more easily but had extreme difficulties with less frequent forms of hate speech.

Conclusion:
The study highlights the difficulty associated with detecting hate speech on Twitter due to the ambiguous nature of the language and context. It also highlights that offensive language is regularly misclassified as hate speech, perhaps due to a very broad definition. Importantly, the model was able to discriminate between hate speech and offensive language accurately, reducing errors present in previous research.

CONCLUSION

If we mix up the hate speeches and offensive languages, for the first time we may think wrong that people are using hate speech, but in reality their intension are just being offensive (mistakes in the lower part of Figure 1) and we might not notice real hate speech when it happens (mistakes in the upper part of Figure 1). Since hate speech can be harmful, it's important to tell the difference between the two. Words that seem offensive can help find hate speech, but they don't always do a good job. In fact, only a few tweets marke by the Hatebase list were considered real hate speech by people who looked at them.

Automated methods can usually separate these types well, but when we study the results closely, we see that some words can either help or make it harder to correctly label tweets. Some words are better for finding hate speech. Words like fg, btch, and ngga can be used in both hate speech and offensive language, but words like fggot and n*gger are mostly used in hate speech. Many tweets that are the most hateful contain several racist or homophobic slurs, making them easier to identify. But if hate speech doesn't have these bad words, it's harder to figure out. So, we need to find data where people use hate speech without using certain bad words.

Our results also show that hate speech can be used in different ways: it can be directed at someone or a group of people, it can be said to no one in particular, or it can happen in a conversation between people. Future research should look at these different ways and the situations where hate speech happens. We should also learn more about the people who use hate speech, including why they do it and the social groups they belong to.

Hate speech is hard to define, and it's not all the same. The way we classify hate speech depends on our own views. People usually see racist and homophobic slurs as hateful but often think sexist language is just offensive. Our results show that people are good at spotting the worst forms of hate speech, like racism and homophobia, but it's important to understand the biases in our systems, and future work should focus on fixing these biases.

FUTURE SCOPE

Context-Aware Classification
Improvement in Contextual Understanding: Future models should incorporate more advanced techniques for contextual analysis, addressing how words like "gay" or "bitch" may vary in meaning based on context. Enhancing models with deeper semantic and contextual comprehension will minimize false positives and improve classification accuracy for borderline cases.
Sentiment & Intent Detection: The ability to differentiate between hate speech, offensive language, and neutral content will be further enhanced by incorporating sentiment and intent detection, helping machines better understand nuances like sarcasm or satire.

Expansion of Data Sources
Diverse Datasets: Collect data from more varied sources to include different languages and dialects (e.g., Hinglish in your case) to improve model robustness. This will also aid in detecting nuanced forms of hate speech, which are often context-specific.

User Characteristics: Including demographic data (when possible) could further improve the classification model, though ethical considerations around privacy and bias must be handled carefully.
Fine-Grained Classification

Subtype Classification: Moving beyond the current binary or multi-class categories, it would be beneficial to categorize hate speech into subtypes (racist, sexist, homophobic, etc.), helping to identify specific forms of harmful language.

Granularity of Offensive Language: Identifying and differentiating between casual offensive language and deeply harmful hate speech would allow for more targeted interventions and responses by social media platforms.

Bias Mitigation in Algorithms
Addressing Social Biases: It's crucial that future models be regularly audited to ensure they do not amplify societal biases. Research into bias mitigation will help ensure that models don't disproportionately flag certain groups or individuals while missing others.

Model Transparency: Developing transparent models with explainability features will help detect and correct any unintended biases in the hate speech detection system.
Cross-Platform Analysis

Multimedia Data: Incorporating images, videos, and other media types alongside textual data could provide a more comprehensive approach to detecting offensive content, especially since hate speech can sometimes be conveyed in non-textual formats.
Cross-Platform Consistency: Ensuring that hate speech detection systems work effectively across different social media platforms will be essential for large-scale applications.
Real-Time Detection and Response

Improved Real- Time Monitoring: Integrating real-time detection systems with automated reporting and

flagging mechanisms could reduce the time it takes for harmful content to be removed from platforms, ensuring a safer online environment.
Automated Content Moderation Tools: Future advancements could lead to more sophisticated systems that not only detect but also suggest or implement actions like temporary suspension or warning notifications to users spreading harmful content.

REFERENCES

Davidson, T., Warmsley, D., Macy, M., & Weber, I. (2017). Automated Hate Speech Detection and the Problem of Offensive Language. *Proceedings of the International AAAI Conference on Web and Social Media, 11*(1), 512-515. –
https://ojs.aaai.org/index.php/ICWSM/article/view/14955

Bird, s., Loper, E., & Klein, E. (2009). Natural Language Processing with Python. O'Reilly Media.
This book teaches how computers can understand and work with human language using Python programming.

Burnap, P., & Williams, M. L. (2015). Cyber hate speech on Twitter: Using machines to find harmful posts. Policy & Internet, 7(2), 223-242.
This study shows how computers can help find hurtful messages (hate speech) on Twitter.

Hutto, C. J., & Gilbert, E. (2014). VADER: A simple rule-based model for social media sentiment. ICWSM.
This research explain a simple tool that helps computers understand if social media posts are happy, sad, or angry.

Kwok, I., & Wang, y. (2013). Finding hate in tweets. AAAI.
This paper explains a popular tool that helps computers learn from data, making it easierto build smart systems.