CS8011- Machine Learning

Sarcasm Detection

Members

Rahul Panwar - 20BCS171

Adarsh Kumar - 20BCS016

SUBMITTED TO - **Dr. Kusum kumar Bharti**

***Abstract*—**Nowadays, social media has an enormous amount of news content with a sarcastic message. It is often expressed in the form of verbal and non-verbal. In this paper, we have aimed to identify sarcasm in news headlines using supervised learning. We address this task with the Bag-of-words features, context-independent features, and context-dependent features. Specifically, we employ six supervised learning models, namely, Naive Bayes-support vector machine, logistic regression,Decision Tree. Our experimental results indicate that Naive Bayes achieves a better performance than others. Index Terms—Sarcasm Detection, Supervised learning, News Headlines Data.

• Introduction

Today, the digital world generates a huge volume of news content about an individual, an organization, a service, or a product in various formats such as images, videos, audio, and text. Therefore, it becomes one of the powerful resources for people to learn about current events or things in the world. Specifically, readers like to read only the headlines rather than the entire news content. The news headlines influence the reader's understanding, reasoning, and deceiving towards the news statement. In this case, sentiment analysis plays a vital role in identifying the news headlines without any misconception. Specifically, sentiment analysis computes a semantic orientation or sentiment polarity of a given news headline into either positive, negative, or neutral. However, it fails to detect a nuanced form of language from the spoken or given news headline. For instance, the sentiment analysis determines the given news headline 'Voters shocked Christie botched such an easy political cover-up' as positive. On the other hand, researchers used sarcasm or humor detection tasks to identify the nuanced form of languages in the given text. The sarcasm detection task identifies whether the given text or news headline is sarcastic or

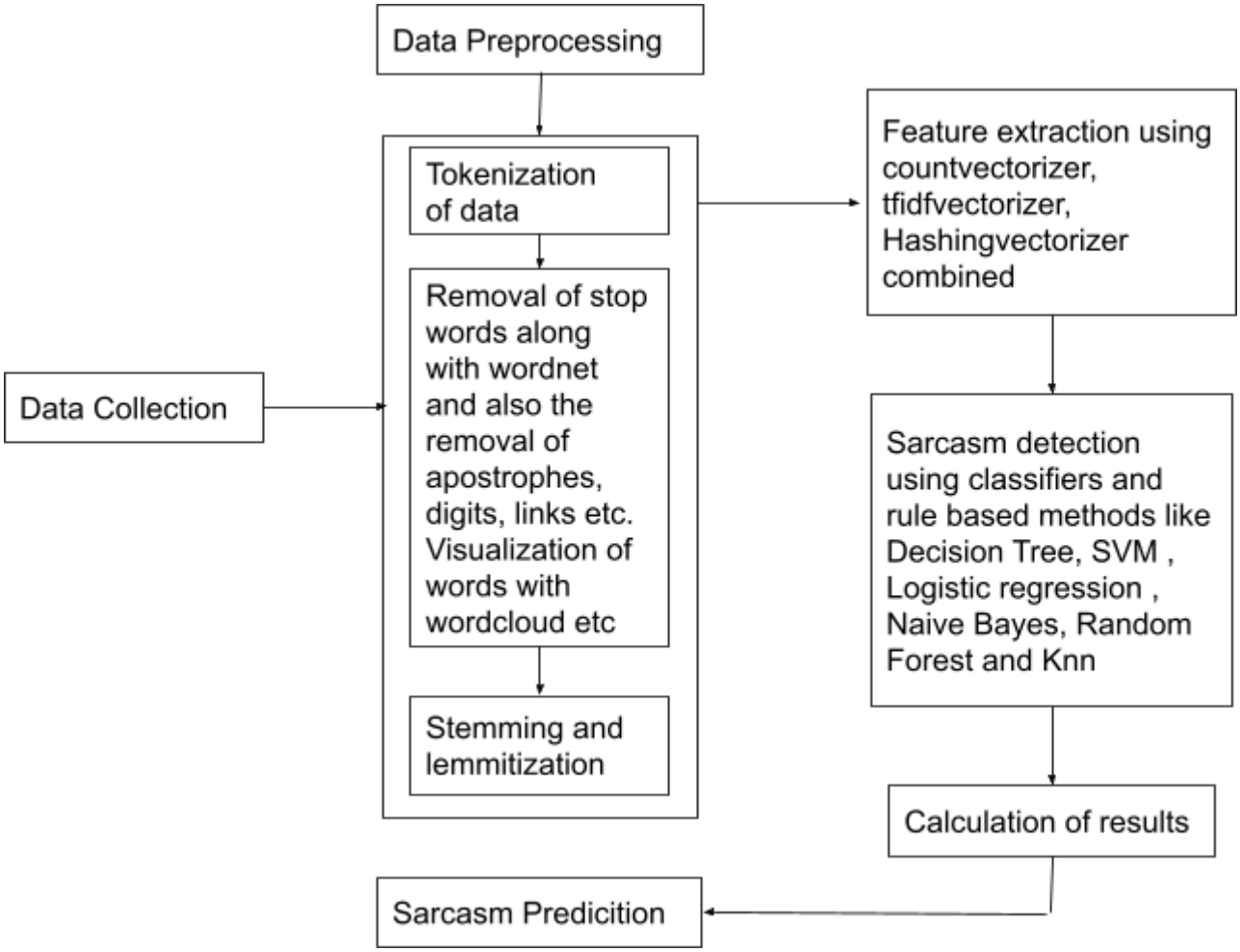
not sarcastic. For instance, the sarcasm detection task determines the given news headline ‘Voters shocked Christie botched such an easy political cover-up’ as sarcasm. Similarly, it determines the given news headline ‘Obama visits Arlington National Cemetery to honor veterans’ as not sarcastic. Therefore, sarcasm detection is a challenging task in natural language processing (NLP). It is widely used in various NLP applications such as marketing research and information categorization. Researchers studied the sarcasm detection task using different techniques, namely, rule-based techniques, machine learning-based techniques, and deep learning-based techniques. First, rule-based techniques identify sarcasm in a text through user-specific rules. These rules are designed to capture human knowledge of a text in a specialized domain. Second, machine learning techniques identify sarcasm in a text using feature-engineering methods. Third, deep learning-based techniques use a semantic representation of a text to identify sarcasm. Both machine learning and deep learning techniques are broadly studied into supervised, unsupervised, and semi-supervised learning. The supervised learning models use the labeled data to map an input text to the desired output. In unsupervised learning, models use unlabeled data to group or cluster similar texts together. The semi-supervised learning models use a large amount of unlabeled data and their part of labeled data to predict the desired output. Recently, transformers-based models achieved a better result in various NLP tasks such as text classification, sentiment analysis, dialogue systems, and recommendation systems. In this paper, we mainly focus on the sarcasm detection task in news headlines. Specifically, we employ supervised learning techniques such as NBSVM (Naive Bayes , Support Vector Machine), LR (Logistic regression) , Decision Tree ,KNN , Random Forest models for the task of sarcasm detection. Both the NBSVM and LR learn bag-of-words (BoW) features.

• Related work

In day-to-day life, everyone uses sarcasm in different situations, where it positively conveys the negative message. Researchers studied sarcasm detection in cognitive sciences, psychology, and linguistics. In this paper, we briefly describe the existing research works in sarcasm detection. Amir et al. developed a system to automatically detect sarcasm in social media using a deep neural network. Specifically, the authors applied user embeddings with word embeddings for recognizing sarcasm. Their study indicated that modeling of authors' information significantly improves the performance. Hazarika et al. designed a contextual sarcasm detection system to identify sarcasm in online social media content. This system adopts both user embeddings (context driven) and content embeddings for boosting classification performance. Kolchinski et al. explored two data-driven methods for sarcasm detection in social media. In particular, the authors used a Bayesian approach to represent the authors' behavior to be sarcastic and a dense embedding approach to learning the author and text interactions, respectively. Then, they employed an augmented BiRNN (Bidirectional Recurrent Neural Networks) on these representations to improve the classification performance. Castro et al. introduced a multimodal sarcasm detection dataset based on popular TV shows. The labels are annotated based on the conversation of audiovisual utterances. Their results indicated that the SVM reduces the error rate up to 12.9% in F1-score. Jena et al.implemented BERT-based C-Net (Contextual-Network) architecture for sarcasm detection. This method uses an SSE (Simple exponential smoothing) in the fusion layer of the proposed C-Net architecture. Their results indicated that the CNet with SSE achieves 75.0% in the Twitter dataset and 66.2% in the Reddit dataset. Nayel et al. developed the SVM model to detect sentiment and sarcasm in Arabic Twitter. They achieved 85.55% accuracy for sentiment detection and 84.22% accuracy for sarcasm detection. Bouazizi and Ohtsuki used a random forest (RF) model to implement a pattern-based approach for detecting sarcasm. The author defined four sets of feature sets such as sentiment features, punctuation features,

syntactic and semantic features, and pattern-related features. They achieved an 81.3% accuracy for sarcasm detection. Moreover, Bamman and Smith adopted the binary LR method to implement contextualized sarcasm detection. The authors divided the features into four classes, namely, tweet features, author features, audience features, and response features. Their results indicate that the proposed LR achieves 85.1% by including all features. Zhang et al. developed a BiGRU with a pooling layer to detect sarcasm in Twitter. The BiGRU model captures the syntactic and semantic features and the pooling layer automatically extracts features from tweets history. They achieved a 90.74% F1 score for balanced data and a 90.26% F1 score for imbalanced data. Liu et al. implemented an A2Text-Net (a deep neural network) for sarcasm detection. This network combines multiple auxiliary data such as part-of-speech (POS), punctuations, emoji, numeral, etc. The authors have shown a better performance using multiple auxiliary data. Mukherjee and Bala proposed a Naive Bayes and fuzzy clustering model to detect sarcasm. They practically tested this model with different feature sets, namely, content words, function words, POS tags, POS ngrams, content and function words, function and POS n-grams, and content, function words, and POS n-grams. Their study indicated that the NB and fuzzy clustering model achieves a better result with content and function words. Joshi et al.developed an SVM with a radial basis kernel to detect sarcasm in tweets and posts. They mainly used context incongruity and sarcastic to define the linguistic features such as implicit congruity, explicit congruity, lexical, and pragmatic features. Their results indicated that the proposed SVM model improves 40% accuracy for rule-based algorithms and 5% for statistical classifiers. Overall, the existing researchers studied sarcasm detection tasks using frequency and unidirectional features in tweets and posts. In this paper, the authors explore sarcasm detection in news headlines using supervised learning methods. Specifically, the authors study various feature techniques such as frequency, unidirectional, and bidirectional

• **Proposed methodology**



Data Preprocessing: In Preprocessing we First did Tokenization. It is the process of tokenizing each word in the sentence. Second step is Stemming and lemmatization. It is the process of achieving the root form of the derived words in the sentence, the words are further converted into present tense format. Third Step is The Removal of stop words. It is the process of removing the unnecessary words in the document, the commonly considered

stop words include (the, a, an, in etc.,) these are some words that make no sense in the sentence.

Tokenization: It is the process of splintering an order of strings into chunks like words, clause, sign and further components called tokens. Tokens can also be discrete words, clauses or full sentences. In the Tokenization concept, few attributes such as punctuation marks are removed. The tokens itself act as an input for upcoming tasks such as parsing and text mining. Example of Tokenization: “Identifying the words” From: flight hasn’t arrived. To: flight has n’t arrived

Removal of stop words: Stop words in general are words that make no sense in the article, but act as a disturbance to it. Stop words like (the, a, an, in etc.) make no change in the article. So these stop words are removed for better performance. Along with the stop words removal we used wordnet for the removal of the grammatically incorrect words and also removed the texts that contains links, digits, apostrophes,commas and parentheses ,URLs (http:// followed by any characters) using regular expressions, Converting the text to lowercase.

Example for stop word removal

Sample text with stop words	Text without stop words
Flight has not arrived yet	Flight not arrived
Thanks for the response. We are hopeful!	Thanks response hope

Stemming and Lemmatization: This concept is used to convert the words into their appropriate root forms so that they can be examined as an individual unit. In this process the words are then converted or changed into their appropriate present tense format.

Original word	Stemmed word
Flight	Fligh
Response	respons
Happy0	Happi

Feature extraction: For Feature extraction we have used the countvectorizer, tfidfvectorizer, Hashingvectorizer combined for the better results and we have also reduced the test size from 0.3 to 0.2. And the ngrams_range in countvectorizer is from 1 to 2 i.e it can take the unigrams and bigrams.

Now the classifiers we used in the our proposed paper are:

Sarcasm detection classifiers

1. **Decision Tree**: This algorithm comes under the supervised learning algorithms. It has been widely used to solve regression and categorization problems. One main reason for using the decision tree algorithm is the creation of the training example which is used to forecast class or value of the target variables by studying the rules and conditions of the decision tree algorithm which is deduced from the training example. This algorithm is very easy compared to other machine learning algorithms because it solves the problem by making use of trees i.e., tree representation,

where each internal node agrees with the credit and each leaf node agrees with the labeling class

Algorithm:

Begin

1. Place the good attribute of the dataset at the root of the tree
2. Split the training set into subsets
3. Each subset contains data with the same value for an attribute
4. Repeat step 1 and step 2
5. On each subset until you find leaf nodes in all the branches of the tree

End

2. Support Vector Machine: Support Vector Machine algorithm is one of the Supervised Learning algorithms, that is used for both organization and Regression scenarios. Initially it was used for categorization problems in Machine Learning. The motive of the Support Vector Machine algorithm is to develop a decision boundary that isolates n-dimensional space into categories in order that they can easily be placed in a new data center in the perfect class in the upcoming scenarios. Hyper plane is to be considered as the perfect decision boundary. Additional uses of this support vector machine algorithm includes text classification, image categorization and face detection etc.

Pros of SVM

SVM algorithm gives best accuracy and also works well with huge dimensional areas. This algorithm generally makes use of a subset of training points and finally uses limited memory space.

Cons of SVM

SVM have a lot of training time that is why they are not apt for big datasets. One more disadvantage is that SVM algorithms do not work well with overlapping classes.

3. Random Forest: Random forest is also considered as one of the supervised learning algorithms used in machine learning that is used in grouping and regression problems. Whereas the main usage is for the organization scenarios. We all already know that when talking about a forest, it's full of trees and when there are more trees then it's considered to be a completely robust forest. Likewise, a random forest algorithm in machine learning develops trees called as the decision trees out of the given examples and then finally predicts the best suited outcome. It generally dissolves the situation of overfitting by making the combination of the results of various decision trees. Random forest machine learning algorithms also undergo a larger range of data products than one individual decision tree algorithm.

Algorithm:

Begin

1. Select of random samples from a given dataset
2. Constructs a decision tree for each sample
3. Gets the prediction result from every decision tree
4. Perform voting for every predicted result
5. Select the most voted prediction result as the final prediction result

End

4. Logistic regression:It is a type of supervised learning algorithm in machine learning that is commonly used for classification problems. The goal of logistic regression is to predict a binary

outcome, i.e., whether an observation belongs to one class or another.

In logistic regression, the input data is a set of independent variables or features, which are used to predict the output or dependent variable, which is a binary variable. The algorithm models the relationship between the input variables and the output variable by estimating the probabilities of the observation belonging to each of the two classes.
The logistic regression model uses a logistic function or sigmoid function to map the input features to the output probabilities. The sigmoid function maps any real-valued number to a probability value between 0 and 1. The output of the logistic regression model is the probability of an observation belonging to a particular class, and this probability is used to make the classification decision.

Begin

1. Load and preprocess the dataset
2. Split the dataset into training and testing sets
3. Define the logistic regression model
4. Train the logistic regression model on the training set using optimization techniques such as gradient descent
5. Evaluate the performance of the trained model on the testing set using a suitable metric such as accuracy, precision, recall, or F1 score
6. Fine-tune the hyperparameters of the logistic regression model to improve its performance on the testing set using techniques such as cross-validation or grid search
7. Save the trained logistic regression model for future use

End

5. Naive bayes: Naive Bayes is a probabilistic supervised learning algorithm used in machine learning for classification problems. The

algorithm is based on Bayes' theorem and assumes that the features are independent of each other, which is often an oversimplification in practice but still works well in many real-world applications.

In Naive Bayes, the input data consists of a set of features and a corresponding class label, and the goal is to learn a model that can predict the class label of new observations based on their features. The algorithm models the probability of each class given the observed features, and then uses Bayes' theorem to calculate the posterior probability of each class given the observed features.
The Naive Bayes algorithm makes a "naive" assumption that the features are conditionally independent given the class label, which means that the presence or absence of one feature does not affect the likelihood of any other feature being present or absent. This assumption simplifies the computation of the model parameters and allows for fast and efficient training and prediction

Algorithm

Begin

1. Load and preprocess the dataset
2. Split the dataset into training and testing sets
3. Calculate the prior probability of each class based on the training set
4. For each feature and each class, calculate the conditional probability of the feature given the class based on the training set using Bayes' theorem and the assumption of conditional independence
5. For each observation in the testing set, calculate the posterior probability of each class given the observed features using Bayes' theorem and the conditional probabilities calculated in step 4
6. Predict the class label of the observation based on the class with the highest posterior probability
7. Evaluate the performance of the Naive Bayes classifier on the testing set using a suitable metric such as accuracy, precision, recall, or F1 score
8. Fine-tune the hyperparameters of the Naive Bayes classifier to improve its performance on the testing set using techniques such as cross-validation or grid search
9. Save the trained Naive Bayes classifier for future use

End

6. KNN: It is a popular supervised learning algorithm used in machine learning. It is commonly used for classification and regression problems, where the goal is to predict the class or value of a new data point based on its similarity to the nearest neighbors in the training data.

Algorithm

Begin

1. Load and preprocess the dataset
2. Split the dataset into training and testing sets
3. Choose the number of neighbors K to consider
4. For each observation in the testing set, calculate the distance to all observations in the training set using a suitable distance metric such as Euclidean distance or Manhattan distance
5. Select the K nearest neighbors based on the calculated distances
6. For classification problems, predict the class label of the observation as the majority class label among the K nearest neighbors; for regression problems, predict the value of the observation as the average value among the K nearest neighbors
7. Evaluate the performance of the KNN algorithm on the testing set using a suitable metric such as accuracy, precision, recall, or F1 score
8. Fine-tune the hyperparameters of the KNN algorithm to improve its performance on the testing set using techniques such as cross-validation or grid search

End

Rule based methods

It includes methods like Semantic, Syntactic etc.

Lexical Method: Here the class, tasks and features are converted to lexical features like noun, verb and adjective respectively. To check and correct the misspelled words General English language

dictionaries are being used which also is used to remove meaningless words

Semantic Method: The meaning of a language is basically considered as semantics. Natural language processing (NLP) can be used to know the way people think and communicate their views. Semantic matching is compared along with graph-based matching to give rise to a score which is used to detect the level of sarcasm. Semantic processing is used for generating a meaningful review.

- **Experimental results**

Our proposed paper results

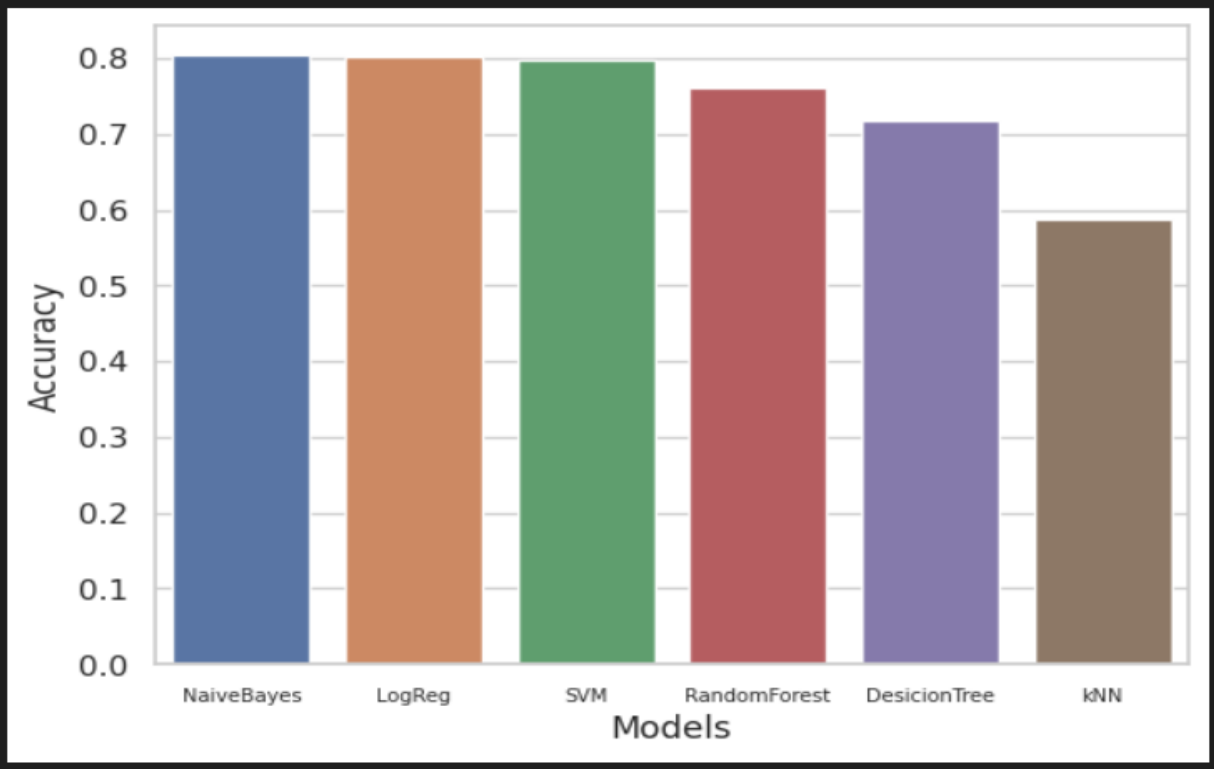
Accuracy comparison for different algorithms.

```
[259]
models = ['DesicionTree','LogReg','SVM', 'RandomForest', 'NaiveBayes', 'kNN']
col = [acc_tree, acc_logr, acc_svm, acc_rf, acc_nb, acc_knn]
data = {'Models':models,'Accuracy':col}
graph_df = pd.DataFrame(data)
graph_df
```

	Models	Accuracy
0	DesicionTree	0.718832
1	LogReg	0.802321
2	SVM	0.797267
3	RandomForest	0.760764
4	NaiveBayes	0.804006
5	kNN	0.587795

```
fig, ax = plt.subplots()

sns.barplot(x=graph_df['Models'], y=graph_df['Accuracy'], data=graph_df)

ax.set_xticklabels(graph_df['Models'], fontsize=7)

plt.show()
```

[261]



Bar plot comparison of accuracy

Most of the Papers are only using either count or tfidf vectorizer for feature extraction but we have used all 3 countvectorizer, tfidfvectorizer, Hashingvectorizer combined as one feature extraction. We have along with stopwords have removed other things also like any tags of html, digits, Apostrophes, commas and parentheses ,URLs (http:// followed by any characters) using regular expressions, Converting the text to lowercase. We have also done the visualization of the

words in various form like wordcloud, barplot, matrix representation formed by tokens and Also we have used the test size to be 0.2 and the ngrams_range is set to 1 to 2 in countvectorizer that contains words in unigrams and bigrams

Implemented Paper results

As we can see the accuracy results of our proposed paper above are better as compared to the research paper.

```
models = ['DesicionTree','LogReg','SVM', 'RandomForest', 'NaiveBayes', 'kNN']
col = [acc_tree, acc_logr, acc_svm, acc_rf, acc_nb, acc_knn]
data = {'Models':models,'Accuracy':col}
graph_df = pd.DataFrame(data)
graph_df
```

[81]

	Models	Accuracy
0	DesicionTree	0.608761
1	LogReg	0.786347
2	SVM	0.772619
3	RandomForest	0.719456
4	NaiveBayes	0.796456
5	kNN	0.572195

Results conforms that Support vector machine provides accuracy of 64 percentage for the SemEval2018-T3-train-taskA.txt given irony detection dataset.

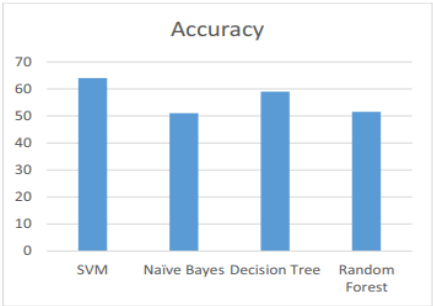


Figure 2. Graph based on Accuracy for SemEval2018-T3-train-taskA.txt given irony detection dataset

Table 4. Accuracy Table SemEval 2018-T3-train-taskA.txt given irony detection dataset

Algorithm	Accuracy
SVM	64%
Naïve Bayes	51%
Decision Tree	59%

Predictions of sarcasm

```
            return True
        else:
            return False

    predictions['correct'] = predictions.apply(lambda x: is_correct(x.predicted, x.actual), axis=1)
    predictions = predictions[['text','predicted','actual','correct']]
```
[166]

```
predictions[predictions['correct']==True].sample(100)
```
[167]

	text	predicted	actual	correct
25052	game chang play week nfl	0	0	True
2238	russia need turkey war isi	0	0	True
22516	bodi found may miss year old left outsid dad p...	0	0	True
22573	hong kong choo new beij back leader amid polit...	0	0	True
17052	night uninterrupt deep sleep realli throw man day	1	1	True
...	...	...	...	...
13553	digitalhealth learn fight hiv aid	0	0	True
26409	sheer beauti montana intrigu meet peopl living...	0	0	True
17740	report resign gop campaign alleg tri block dam...	0	0	True
18464	wednesday morn email conserv love propo gop ob...	0	0	True
18652	quick guid year oscar best pictur nomin	0	0	True

100 rows × 4 columns

• Discussion

Considering these six machine learning algorithms for the sarcasm detection of the news headlines dataset, it has been found that the Naive Bayes provides more accurate results compared to the other machine learning algorithms when the feature extraction is done combinedly by Countvectorizer, tfidfvectorizer and HashingVectorizer after the data preprocessing but when only countvectorizer is performed as the feature extraction the SVM gives more accurate results. In the X axis it represents the various algorithms like SVM, Naïve Bayes. Decision Tree and Random Forest, KNN, Logistic Regression and Y axis represent accuracy of the each algorithms.

• Conclusion and Future direction

Several pre-processing techniques are used for the sarcasm detection of the dataset. Developers worked on the categorization and results have been provided. A comparative study on these algorithms has been performed to identify which of these algorithms give the best results. Machine Learning algorithms used to detect the sarcasm here are Support Vector Machine, Naïve Bayes ,Decision Tree , Logistic Regression, KNN, Random Forest for the News Headlines dataset and found Naive bayes algorithm to be the best suited for the particular dataset. Algorithms like Random Forest and SVM are used for the dataset Sarcasm Detection.txt and found Random Forest algorithm to give best results with the accuracy of 76%. Future work can be proceeded by using some more machine learning classifiers and comparing the results to obtain the accuracy of the best suited algorithm

.

• References

1] S. K. Bharti, et al., “Sarcastic sentiment detection in tweets streamed in real time: a big data approach,” Digital Communications and Networks, vol. 2, no. 3, pp. 108–121, 2016.

[2] Mondher Bouazizi and Tomoaki Otsuki, “A pattern-based approach for sarcasm detection on twitter,” IEEE Transl., 2016.

[3] S. Rossano, J. Paloma, and T. Joel, “Detecting sarcasm in multimodal social platforms,” in Proceedings of the 2016 ACM on Multimedia Conference, ACM, 2016.

[4] Santosh Kumar Bharti, Korra Sathya Babu, and Sanjay Kumar Jena, “Parsing-based sarcasm sentiment recognition

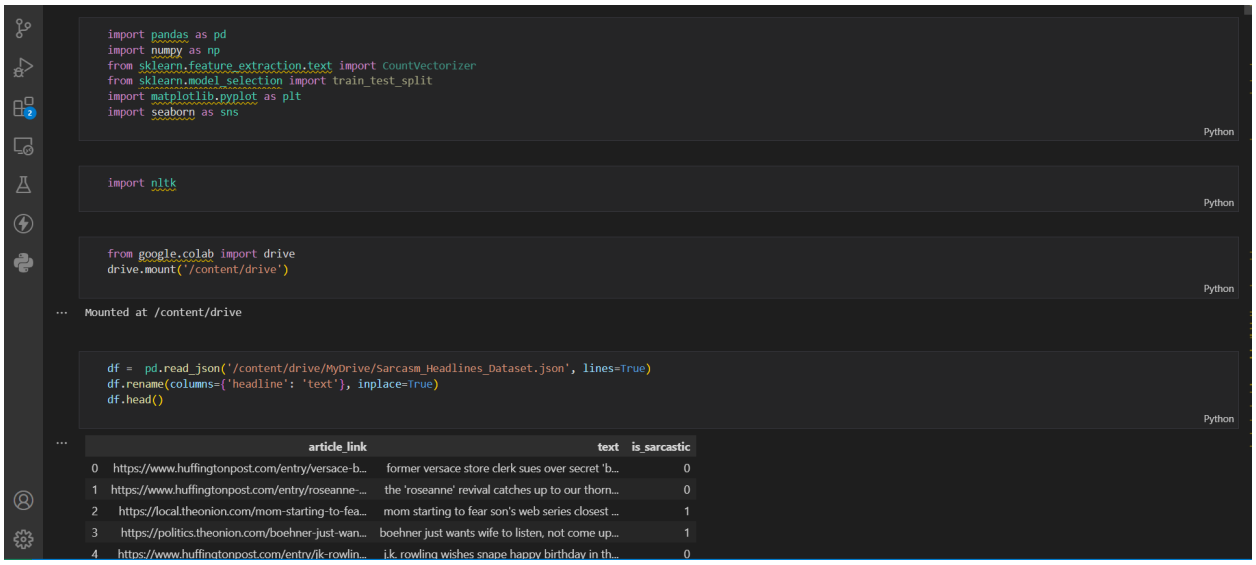
in Twitter data," in 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM). IEEE, 2015.

[5] Debanjan Ghosh, Weiwei Guo, and Smaranda Muresan, "Sarcastic or not: word embeddings to predict the literal or sarcastic meaning of words," in Proccedings of the 2015 Conference on Empirical Methods in Natural Language Processing, Accociation for Computational Linguistics, 2015.

[6] Florian Kunneman, et al., "Signaling sarcasm: From hyperbole to hashtag," Information Processing & Management, vol. 51, no. 4, pp. 500–509, 2015

- **Appendix (Complete code with procedure)**

Data collection



```
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import seaborn as sns
```

Python

```
import nltk
```

Python

```
from google.colab import drive
drive.mount('/content/drive')
```

Python

```
Mounted at /content/drive
```

```
df =  pd.read_json('/content/drive/MyDrive/Sarcasm_Headlines_Dataset.json', lines=True)
df.rename(columns={'headline': 'text'}, inplace=True)
df.head()
```

Python

	article_link	text	is_sarcastic
0	https://www.huffingtonpost.com/entry/versace-b...	former versace store clerk sues over secret 'b...	0
1	https://www.huffingtonpost.com/entry/roseanne-...	the 'roseanne' revival catches up to our thorn...	0
2	https://local.theonion.com/mom-starting-to-fea...	mom starting to fear son's web series closest ...	1
3	https://politics.theonion.com/boehner-just-wan...	boehner just wants wife to listen, not come up...	1
4	https://www.huffingtonpost.com/entry/jk-rowlin...	j.k. rowling wishes snape happy birthday in th...	0

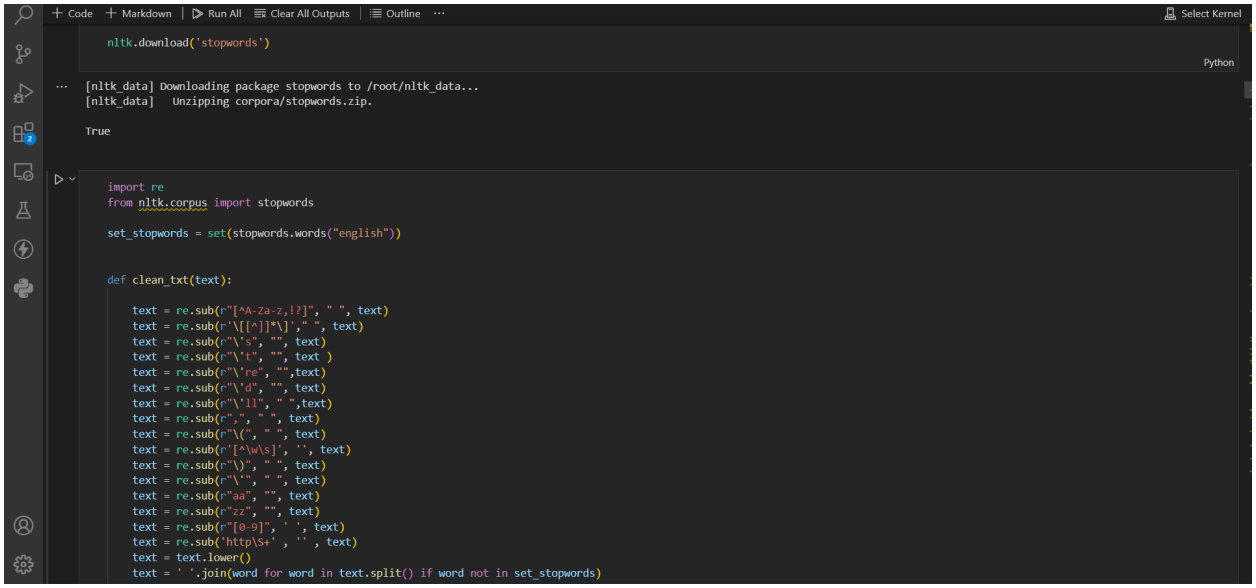
Here we have first we have downloaded and imported necessary libraries and then imported the newsheadlines dataset by mounting

the google drive and have read the dataset that contains 3 columns with 26709 rows.

Now here we have dropped the article link column and then we have calculated the number of words in the dataset that are 263036. Then we check if certain rows contain null data or not. After that we count the number of sarcastic and non-sarcastic headlines, 0 means non-sarcastic and 1 means sarcastic.

Data preprocessing

Now we have plotted the countplot for the number of text that are sarcastic and non-sarcastic.



```
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.

True
```

```
import re
from nltk.corpus import stopwords

set_stopwords = set(stopwords.words("english"))


def clean_txt(text):

    text = re.sub(r"[^A-Za-z,!?]", " ", text)
    text = re.sub(r'\[[^]]*\]'," ", text)
    text = re.sub(r"\'s", "", text)
    text = re.sub(r"\'t", "", text )
    text = re.sub(r"\'re", "",text)
    text = re.sub(r"\'d", "", text)
    text = re.sub(r"\'ll", " ",text)
    text = re.sub(r",", " ", text)
    text = re.sub(r"\(", " ", text)
    text = re.sub(r'[^\w\s]', '', text)
    text = re.sub(r"\)", " ", text)
    text = re.sub(r"\"", " ", text)
    text = re.sub(r"aa", "", text)
    text = re.sub(r"zz", "", text)
    text = re.sub(r"[0-9]", ' ', text)
    text = re.sub('http\S+' , '' , text)
    text = text.lower()
    text = ' '.join(word for word in text.split() if word not in set_stopwords)
```

Now we have downloaded the stopwords from nltk that will contains all the stopwords in the english that does not make any sense or are of no use in sarcasm detection.Along with the removal of the stop words we have also removed the non-alphabetic characters , apostrophes,commas and parenthesis , special characters and punctuation , digits, URLs.

```
from nltk.corpus import stopwords
import string
stop = set(stopwords.words('english'))
a = list(string.punctuation)
stop.update(a)
```

```
len(stop)
```

```
211
```

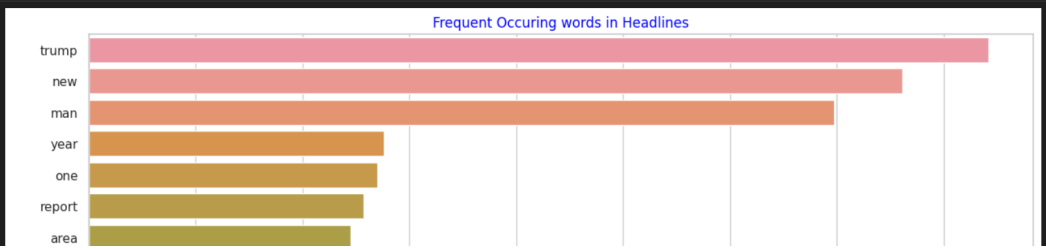
```
df_clean_len = df['text'].apply(lambda x: len(x.split(' '))).sum()
print(f'After text cleaning we have only {df_clean_len} words to work with')
```

```
After text cleaning we have only 187817 words to work with
```

Now we have imported the stopwords from the english there are 211 stopwords in it. And after cleaning all the text we have left with 187817 words.

```python
from collections import Counter
import seaborn as sns

text = df['text']
words = text.str.split(expand=True).unstack()
result_count = Counter(words).most_common()
result_df = pd.DataFrame(result_count).reset_index().drop(0)

sns.set_theme(style="whitegrid")
f, ax = plt.subplots(figsize=(15, 15))
sns.barplot(y=result_df[0][0:30], x=result_df[1][0:30], data=result_df, palette=None)
plt.ylabel('Words', color="blue")  # Add an x-label to the axes.
plt.xlabel('Count', color="blue")  # Add a y-label to the axes.
plt.title("Frequent Occuring words in Headlines", color="blue")
plt.xticks(rotation=50);
ax.tick_params(axis='x', colors='black')
plt.show()
```

Python



Now we have visualized the most frequent occurring words in the whole text of the dataset.

```python
from bs4 import BeautifulSoup
from nltk import stem
import re
snow = stem.SnowballStemmer('english')
def strip_html(text):
    soup = BeautifulSoup(text , 'html.parser')
    return soup.get_text()
```

Python

```python
def removing_stopwords(text):
    text = text.lower()
    final_text = []
    for i in text.split():
        if(i.strip().lower() not in stop):
            final_text.append(snow.stem(i.strip()))
    return ' '.join(final_text)
```

Python

```python
def denoise(text):
    text = text.lower()
    text = strip_html(text)
    text = clean_txt(text)
    text = removing_stopwords(text)
    return text
```

Python

```python
df['cleaned'] =df['text'].apply(denoise)
```

Python

Now we have used beautifulsoup library for removing any html tags. And we have used snowball stemmer for reducing the words to its root form eg. walking to walk.

Now we have applied all the defined functions for the cleaning of the text.

```
df.head()
```

	text	is_sarcastic	cleaned
0	former versace store clerk sues secret black c...	0	former versac store clerk sue secret black cod...
1	roseanne revival catches thorny political mood...	0	roseann reviv catch thorni polit mood better wors
2	mom starting fear son web series closest thing...	1	mom start fear son web seri closest thing gran...
3	boehner wants wife listen come alternative deb...	1	boehner want wife listen come altern debt redu...
4	j k rowling wishes snape happy birthday magica...	0	j k rowl wish snape happi birthday magic way

```
df['cleaned'].iloc[1]
```

```
'roseann reviv catch thorni polit mood better wors'
```

```
df['all_cleaned'] = df['cleaned'].apply(denoise)
df.head()
```

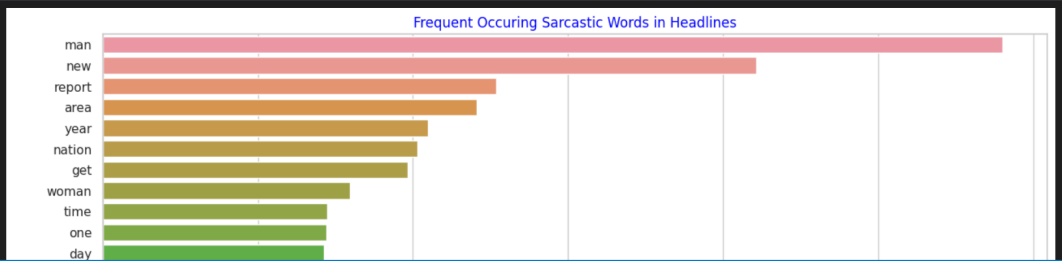
	text	is_sarcastic	cleaned	all_cleaned
0	former versace store clerk sues secret black c...	0	former versac store clerk sue secret black cod...	former versac store clerk sue secret black cod...
1	roseanne revival catches thorny political mood...	0	roseann reviv catch thorni polit mood better wors	roseann reviv catch thorni polit mood better wor
2	mom starting fear son web series closest thing...	1	mom start fear son web seri closest thing gran...	mom start fear son web seri closest thing gran...
3	boehner wants wife listen come alternative deb...	1	boehner want wife listen come altern debt redu...	boehner want wife listen come altern debt redu...
4	j k rowling wishes snape happy birthday magica...	0	j k rowl wish snape happi birthday magic way	j k rowl wish snape happi birthday magic way

Now comparing the cleaned text with the original text

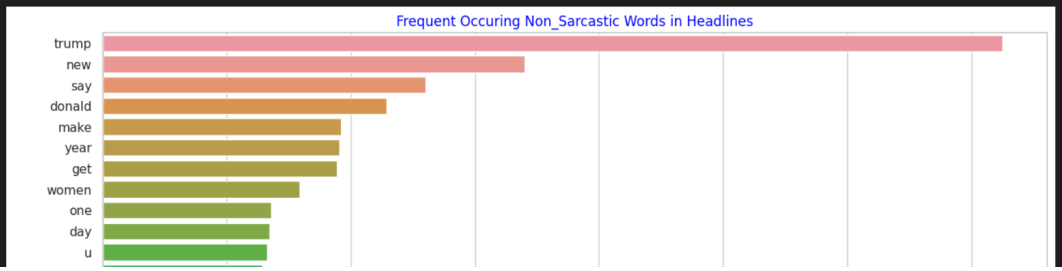
```
sarcastic = pd.DataFrame(df[df['is_sarcastic']==1]['text'].str.split(expand=True).unstack().value_counts()).reset_index()
```

```
non_sarcastic = pd.DataFrame(df[df['is_sarcastic']==0]['text'].str.split(expand=True).unstack().value_counts()).reset_index()
```

```
sns.set_theme(style="whitegrid")
f, ax = plt.subplots(figsize=(15, 10))
sns.barplot(y=sarcastic['index'][0:30], x=sarcastic[0][0:30], data=result_df, palette=None)
plt.ylabel('Words', color="blue")  # Add an x-label to the axes.
plt.xlabel('Count', color="blue")  # Add a y-label to the axes.
plt.title("Frequent Occuring Sarcastic Words in Headlines", color="blue")
plt.xticks(rotation=70);
plt.show()
```



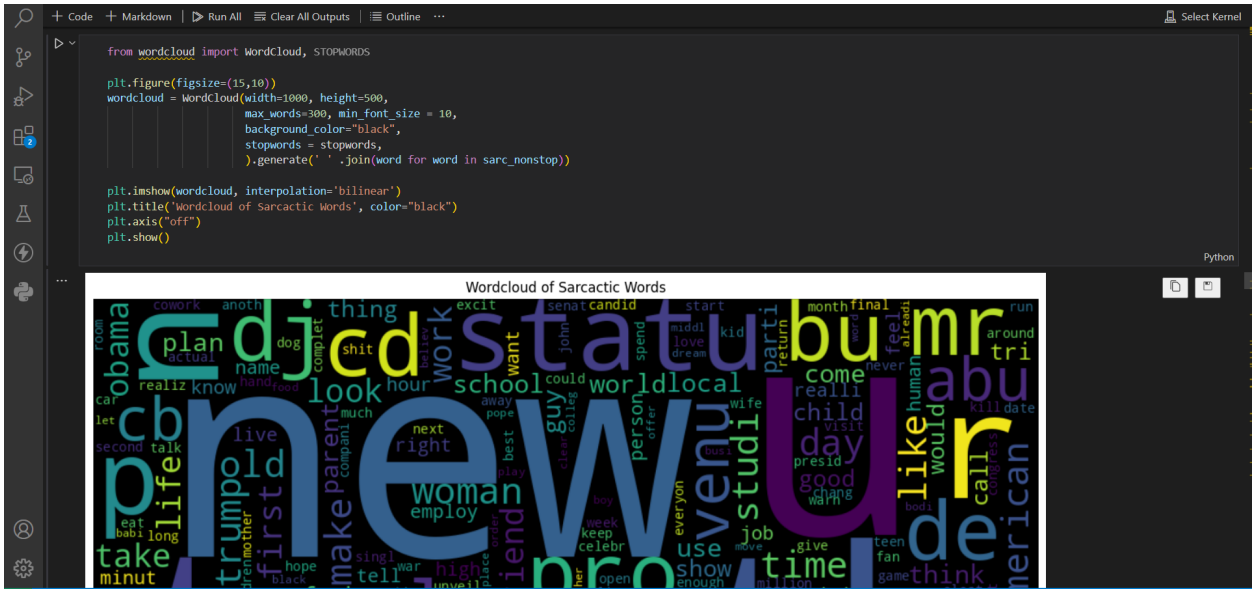

```
sns.set_theme(style="whitegrid")
f, ax = plt.subplots(figsize=(15, 10))
sns.barplot(y=non_sarcastic['index'][0:30], x=non_sarcastic[0][0:30], data=result_df, palette=None)
plt.ylabel('Words', color="blue")  # Add an x-label to the axes.
plt.xlabel('Count', color="blue")  # Add a y-label to the axes.
plt.title("Frequent Occuring Non_Sarcastic Words in Headlines", color="blue")
plt.xticks(rotation=70);
plt.show()
```



Now storing all the words that fall under sarcastic category in one variable and non-sarcastic in another variable and counting and visualizing the most frequent words using the barplots for both the sarcastic and non-sarcastic words.

```
from nltk.corpus import stopwords
stopwords = stopwords.words('english')
sarcasctic_2 = [every_word.lower() for every_word in sarcastic['index']]

sarc_nonstop = [word for word in sarcasctic_2 if word not in stopwords]

non_sarcasctic_2 = [every_word.lower() for every_word in non_sarcastic['index']]

non_sarc_nonstop = [word for word in non_sarcasctic_2 if word not in stopwords]
```

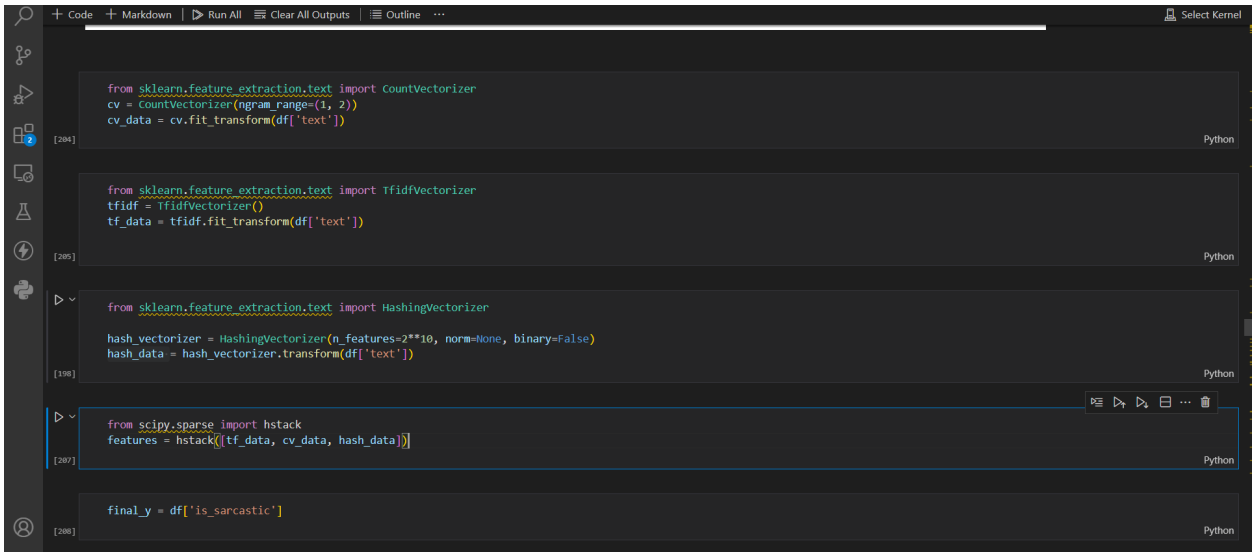
Now storing the non-sarcastic words without stopwords in one variable and sarcastic words without the stopwords in another variable



```
from wordcloud import WordCloud, STOPWORDS

plt.figure(figsize=(15,10))
wordcloud = WordCloud(width=1000, height=500,
                      max_words=300, min_font_size = 10,
                      background_color="black",
                      stopwords = stopwords,
                      ).generate(' '.join(word for word in sarc_nonstop))

plt.imshow(wordcloud, interpolation='bilinear')
plt.title('Wordcloud of Sarcactic Words', color="black")
plt.axis("off")
plt.show()
```


Now visualizing the frequent words using the wordcloud where the largest word represent the most frequent occurring in the dataset.

Feature extraction



```
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer(ngram_range=(1, 2))
cv_data = cv.fit_transform(df['text'])
```

[204] Python

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer()
tf_data = tfidf.fit_transform(df['text'])
```

[205] Python

```
from sklearn.feature_extraction.text import HashingVectorizer

hash_vectorizer = HashingVectorizer(n_features=2**10, norm=None, binary=False)
hash_data = hash_vectorizer.transform(df['text'])
```

[198] Python

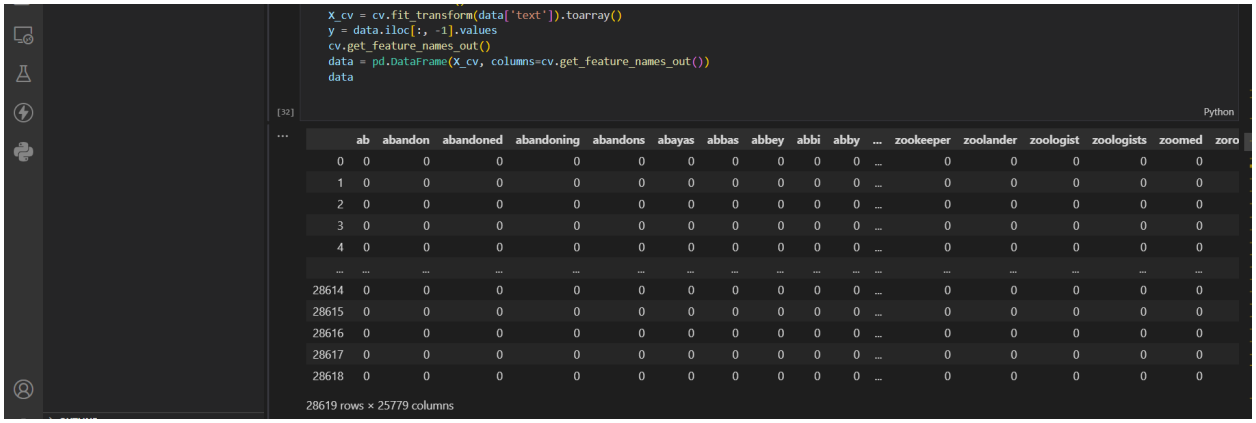
```
from scipy.sparse import hstack
features = hstack([tf_data, cv_data, hash_data])
```

[207] Python

```
final_y = df['is_sarcastic']
```

[208] Python

Now feature extraction I have used all the feature extraction methods combined such as countvectorizer, tfidfvectorizer, Hashingvectorizer and have combined them in one feature using the hstack function.



```
X_cv = cv.fit_transform(data['text']).toarray()
y = data.iloc[:, -1].values
cv.get_feature_names_out()
data = pd.DataFrame(X_cv, columns=cv.get_feature_names_out())
data
```

[32] Python

	ab	abandon	abandoned	abandoning	abandons	abayas	abbas	abbey	abbi	abby	...	zookeeper	zoolander	zoologist	zoologists	zoomed	zoro
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
28614	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
28615	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
28616	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
28617	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
28618	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	

28619 rows × 25779 columns

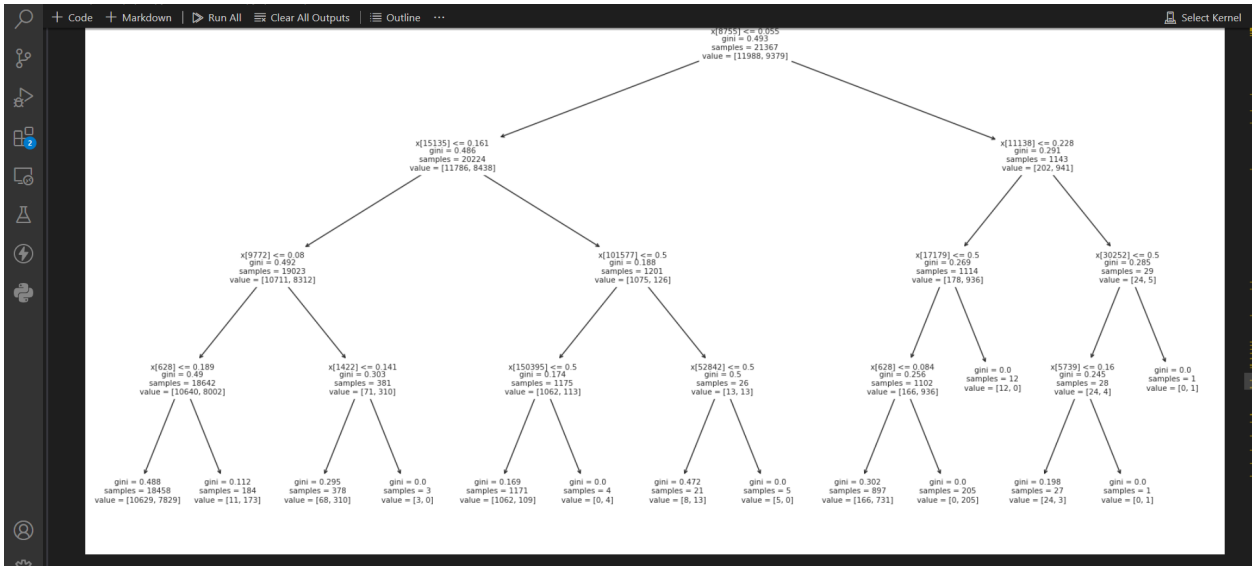
Visualization of words in the matrix form.

Training and testing and sarcasm detection using classifiers

```
[209]
from sklearn.model_selection import train_test_split
X_train  , X_test , y_train , y_test = train_test_split(features , final_y , test_size = 0.2 , stratify = final_y)
Python

[202]
X_train.shape , X_test.shape
Python

...
((21367, 164179), (5342, 164179))

[232]
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, f1_score,  precision_score, recall_score
model = LogisticRegression()
model.fit(X_train , y_train)
pred = model.predict(X_test)
from sklearn.metrics import accuracy_score
print('Logistic regression accuracy', accuracy_score(y_test , pred))
print('Logistic regression F1-score', f1_score(y_test , pred))
print('Logistic regression precision:', precision_score(y_test, pred))
print('Logistic regression recall:',  recall_score(y_test, pred))
Python

...
Logistic regression accuracy 0.8023212280044927
Logistic regression F1-score 0.7586837294332724
Logistic regression precision: 0.8173313638601674
Logistic regression recall: 0.707889125799573 5
```

Now we have divide it into train and test set with the test_size of 0.2 for better accuracy. And we have first do the fitting the x_train and y_train with the logistic regression that gives .80 accuracy.

```
[237]
from sklearn.tree import DecisionTreeClassifier
Python

[238]
tree_model = DecisionTreeClassifier(criterion='gini', max_depth=4, random_state=1)
tree_model.fit(X_train, y_train)
Python

...
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=4, random_state=1)

import matplotlib.pyplot as plt
from sklearn import tree

fig, ax = plt.subplots(figsize=(20, 10))

tree.plot_tree(tree_model, ax=ax)

plt.savefig('decision.pdf')
plt.show()
```




```
from sklearn.metrics import confusion_matrix, classification_report
```
[240] Python

```
y_pred = tree_model.predict(X_test)
test = np.array(y_test)
predictions = np.array(y_pred)
confusion_matrix(test, predictions)
```
[241] Python

```
array([[2933,   64],
       [2022,  323]])
```

```
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier()
model.fit(X_train , y_train)
pred = model.predict(X_test)
print('Decision tree accuracy:' ,accuracy_score(y_test , pred))
print('Decision tree F1 score:',  f1_score(y_test,pred))
```
[242] Python

```
Decision tree accuracy: 0.7188318981654811
Decision tree F1 score: 0.6681396376491382
```

```
print('Decision tree precision:', precision_score(y_test, pred))
print('Decision tree recall:',  recall_score(y_test, pred))
```

Similarly for decision tree that gives the accuracy of 0.71

```
from sklearn.svm import SVC
model = SVC()
model.fit(X_train , y_train)
pred = model.predict(X_test)
accuracy_score(y_test , pred)
```
[245] Python

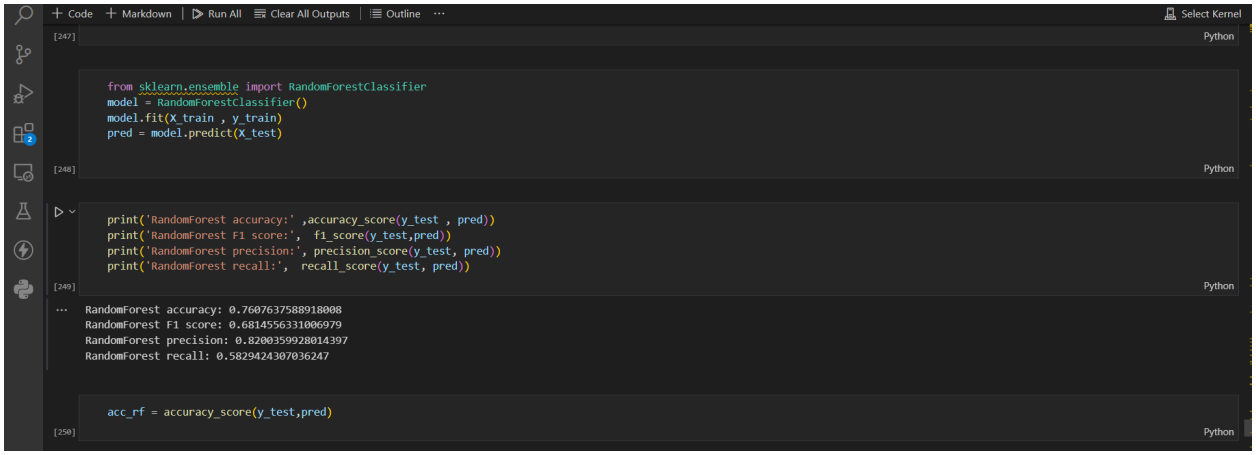
```
0.7972669412205167
```

```
print('SVM accuracy:' ,accuracy_score(y_test , pred))
print('SVM F1 score:',  f1_score(y_test,pred))
print('SVM precision:', precision_score(y_test, pred))
print('SVM recall:',  recall_score(y_test, pred))
```
[246] Python

```
SVM accuracy: 0.7972669412205167
SVM F1 score: 0.7472578763127188
SVM precision: 0.8252577319587628
SVM recall: 0.68272921108742
```

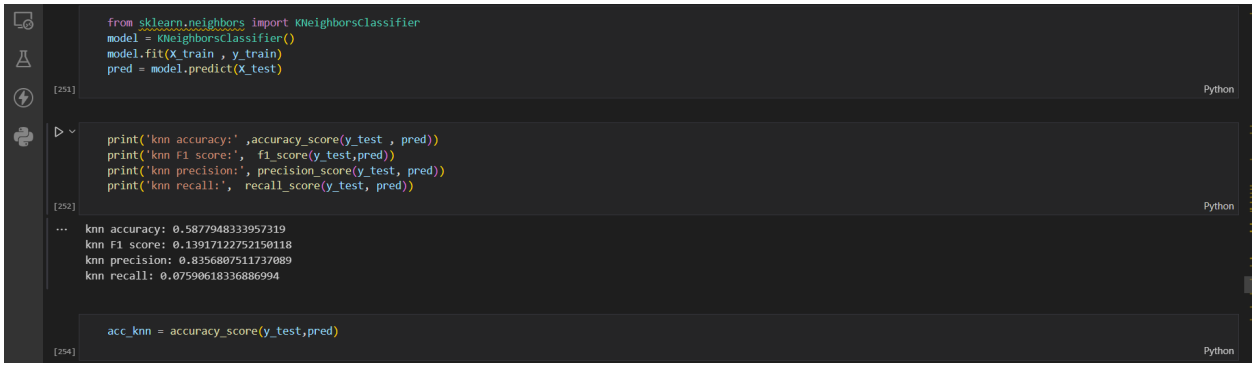
```
acc_svm = accuracy_score(y_test,pred)
```
[247] Python

Similarly SVM that gives the accuracy of 0.79



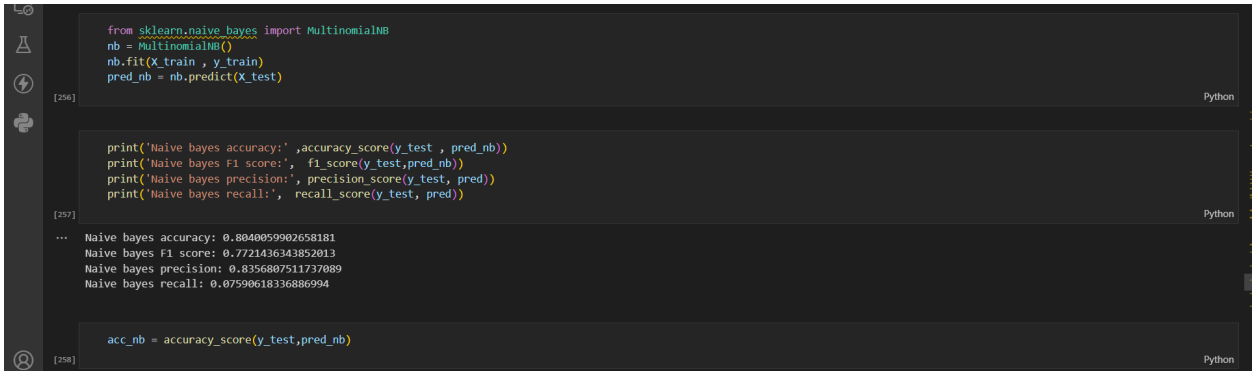
```
[247]
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier()
model.fit(X_train , y_train)
pred = model.predict(X_test)
[248]
print('RandomForest accuracy:' ,accuracy_score(y_test , pred))
print('RandomForest F1 score:',  f1_score(y_test,pred))
print('RandomForest precision:', precision_score(y_test, pred))
print('RandomForest recall:',  recall_score(y_test, pred))
[249]
RandomForest accuracy: 0.7607637588918008
RandomForest F1 score: 0.6814556331006979
RandomForest precision: 0.8200359928014397
RandomForest recall: 0.5829424307036247
acc_rf = accuracy_score(y_test,pred)
[250]
```

Similarly Random Forest that gives the accuracy of 0.76



```
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier()
model.fit(X_train , y_train)
pred = model.predict(X_test)
[251]
print('knn accuracy:' ,accuracy_score(y_test , pred))
print('knn F1 score:',  f1_score(y_test,pred))
print('knn precision:', precision_score(y_test, pred))
print('knn recall:',  recall_score(y_test, pred))
[252]
knn accuracy: 0.5877948333957319
knn F1 score: 0.1391712275215011 8
knn precision: 0.8356807511737089
knn recall: 0.07590618336886994
acc_knn = accuracy_score(y_test,pred)
[254]
```

Similarly KNN that gives the accuracy of 0.58



```
from sklearn.naive_bayes import MultinomialNB
nb = MultinomialNB()
nb.fit(X_train , y_train)
pred_nb = nb.predict(X_test)
[256]
print('Naive bayes accuracy:' ,accuracy_score(y_test , pred_nb))
print('Naive bayes F1 score:',  f1_score(y_test,pred_nb))
print('Naive bayes precision:', precision_score(y_test, pred))
print('Naive bayes recall:',  recall_score(y_test, pred))
[257]
Naive bayes accuracy: 0.8040059902658181
Naive bayes F1 score: 0.7721436343852013
Naive bayes precision: 0.8356807511737089
Naive bayes recall: 0.07590618336886994
acc_nb = accuracy_score(y_test,pred_nb)
[258]
```

Similarly for Naive bayes that gives the accuracy of 0.804

```
models = ['DesicionTree','LogReg','SVM', 'RandomForest', 'NaiveBayes', 'kNN']
col = [acc_tree, acc_logr, acc_svm, acc_rf, acc_nb, acc_knn]
data = {'Models':models,'Accuracy':col}
graph_df = pd.DataFrame(data)
graph_df
```

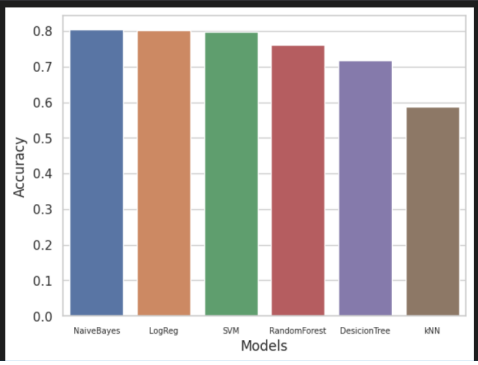
[259] Python

	Models	Accuracy
0	DesicionTree	0.718832
1	LogReg	0.802321
2	SVM	0.797267
3	RandomForest	0.760764
4	NaiveBayes	0.804006
5	kNN	0.587795

Now comparing the accuracy of the all the classifiers.

```
fig, ax = plt.subplots()

sns.barplot(x=graph_df['Models'], y=graph_df['Accuracy'], data=graph_df)

ax.set_xticklabels(graph_df['Models'], fontsize=7)

plt.show()
```

[261] Python



Now similarly comparing the accuracy using barplot

+ Code + Markdown | Run All | Clear All Outputs | Outline ··· Select Kernel

```
results = pd.DataFrame(data={'predicted': y_pred, 'actual': y_test})
predictions = results.join(df)
```

[165] Python

```
def is_correct(predicted, actual):
    if predicted == actual:
        return True
    else:
        return False

predictions['correct'] = predictions.apply(lambda x: is_correct(x.predicted, x.actual), axis=1)
predictions = predictions[['text','predicted','actual','correct']]
```

[166] Python

```
predictions[predictions['correct']==True].sample(100)
```

[167] Python

	text	predicted	actual	correct
25052	game chang play week nfl	0	0	True
2238	russia need turkey war isi	0	0	True
22516	bodi found may miss year old left outsid dad p...	0	0	True
22573	hong kong choo new beij back leader amid polit...	0	0	True
17052	night uninterrupt deep sleep realli throw man day	1	1	True
...	...	...	...	...
13553	digitalhealth learn fight hiv aid	0	0	True
26409	sheer beauti montana intrigu meet peopl living...	0	0	True
17740	report resign gop campaign alleg tri block dam...	0	0	True

Now predicting the results of the trained model.