Project Report On Sarcasm Detection

Harsh Jain

PROJECT - SARCASM DETECTION

Table of Contents

Chapter 1

Introduction

Sarcasm, which is both positively funny and negatively nasty, plays an important part in human social interaction. But sometimes it is difficult to detect whether someone is making fun of us with some irony. So to make it easy we built something which helps you in detecting sarcastic text with the help of **Natural Language Processing.**

Problem Statement

This case requires trainees to develop a text classification model to label a news headline as sarcastic or not. This News Headlines dataset for Sarcasm Detection is collected from two news website. We collect real (and non-sarcastic) news headlines from different Post.

Data

The given data set are :-
Sarcasm_Headlines_Dataset

Understanding of data is the very first and important step in the process of finding solution of any business problem. we need to go through each and every variable of it to understand and for better functioning..

- Size of Dataset Provided: - 16067 rows, 7 Columns
- Missing Values: No

Below mentioned is a list of all the variable names with their meanings:

Variables	**Description**
1. is_sarcastic	‘ 1’ if the record is sarcastic otherwise ‘0’
2. Headline:	the headline of the news article
3. Article_link	link to the original news article.

Chapter 2
Pre-Processing of data

Natural Language Processing

The field of study that focuses on the interactions between human language and computers is called Natural Language Processing, or NLP for short. It sits at the intersection of computer science, artificial intelligence, and computational linguistics

N stands for **Natural** – how we use our brain and body to produce our life results.

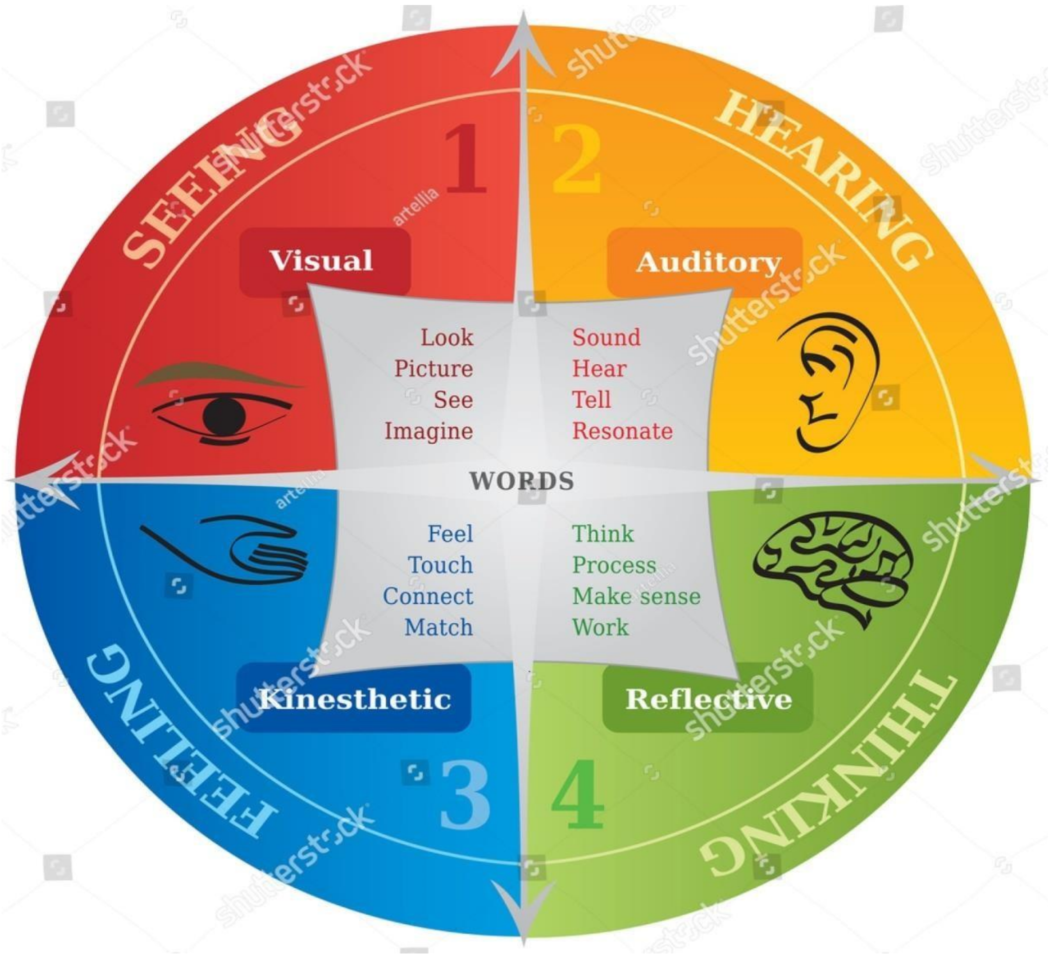
L stands for **Linguistic** – how we code information from the world around us and internal thoughts so that we can make sense of them. How we communicate our thoughts to others.

P stands for **Programming** – How our brain and body reacts to various internal thoughts and external information to produce repetitive behaviour, habits and thinking.

NLP is a field of artificial intelligence in which computers analyze, understand and derive meaning from human language in a smart and useful way. Natural Language Processing is ubiquitous and has multiple application. By utilizing NLP, developers can organize and structure knowledge to perform various tasks .

Applications of NLP:-

- sentiment analysis
- speech recognition
- topic segmentation.
- named entity recognition,
- parts-of-speech tagging
- relationship extraction
- automatic summarization
- chatbots & AI agents
- translation



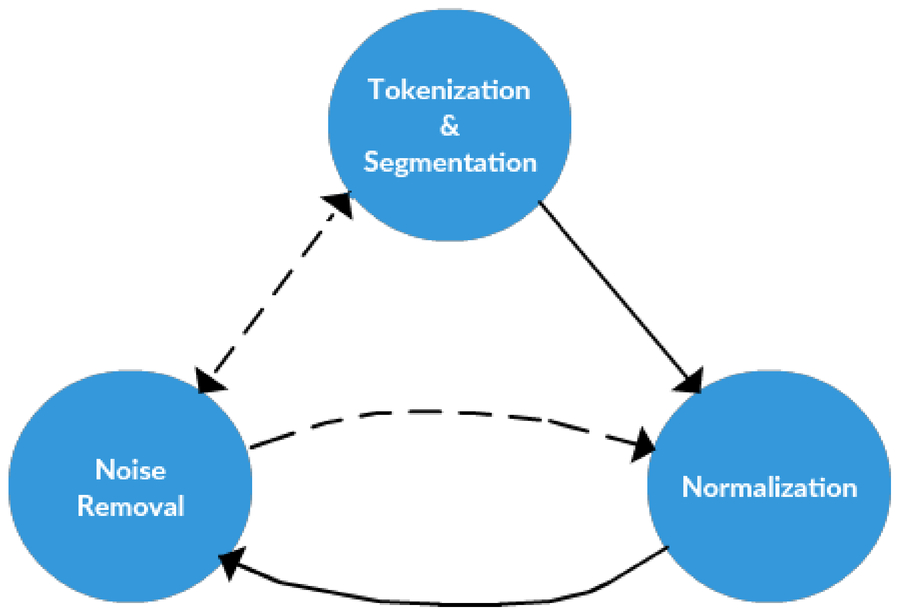
Data Processing

The work starts with data preprocessing, means looking at the data to get insights. However, in data mining terms looking at data refers to so much more than just looking. Looking at data refers to exploring the data, cleaning the data as well as visualizing the data through graphs and plots. This is often called as **Exploratory Data Analysis**

A good rule of thumb is to look at the data first and then clean it up. **A clean dataset will allow a model to learn meaningful features and not overfit on irrelevant noise.** Generally, text data contains a lot of noise either in the form of symbols or in the form of punctuations and stopwords. Therefore, it becomes necessary to clean the text, not just for making it more understandable but also for getting better insights.

If the data is scrapped or data is given for analyzing it would always be in its natural human format of sentences, or paragraphs etc. Before doing analysis on that we need to transform that language and clean it so that the computer is able to understand that language in the desired format.

Data preprocessing is a fundamental step while building a machine learning model. If the data is fairly pre-processed the results would be reliable.



The common pre-processing steps are:

- Conversion to lower case
- Removing punctuation
- Removing stopwords
- Tokenization
- Stemming
- Lemmetization

Conversion to lower case

Words having the same meaning like nlp and NLP if they are not converted into lowercase then these both will constitute as non-identical words in the vector space model.so we convert all the letter inti lower case.

```
df['headline'] = df.headline.apply(lambda x:x.lower())
```

Removing punctuation

We want words in the text data but without punctuation because these are noise for the machine learing models so it is necessary to remove punctuation for the effective model performance.

```
df['headline'] = df.headline.apply(lambda x: ' '.join(word.strip(string.punctuation) for word
in x.split()))
```

Removing stopwords

- These are unhelpful words like 'the', 'is', 'at'.
- These are not helpful because the frequency of such stopwords is high in the corpus, but they don't help in differentiating the target classes.
- The removal of Stopwords also reduces the data size.
- These are the most often used that do not have any significance while determining the two different documents like (a, an,the, etc.) so they are to be removed.

stopwords = set(STOPWORDS)

stopwords.update(["hi","will","still","thing","first","say","new","now","one","day","make","getting","take","back","year"])

stopwords_removed = [word for word in final_text_sar.split() if word not in stopwords]

Tokenization

Tokenization is the process by which big quantity of text is divided into smaller parts called **tokens**.

It becomes vital to understand the pattern in the text to achieve the above-stated purpose. These tokens are very useful for finding such patterns as well as is considered as a base step for stemming and lemmatization**.**

Natural Language toolkit has very important module **tokenize** which further comprises of sub-modules

1. sentence tokenize
2. word tokenize

sentence tokenize

Imagine you need to count average words per sentence. For accomplishing such a task, you need both sentence tokenization as well as words to calculate the ratio. Such output serves as an important feature for machine training as the answer would be numeric. We have taken the same sentence. Further sent module parsed that sentences and show output. It is clear that this function breaks each sentence.

```
from nltk.tokenize import sent_tokenize
text = "God is Great! I won a lottery."
print(sent_tokenize(text))

Output: ['God is Great!', 'I won a lottery ']
```

word tokenize

The output of word tokenization can be converted to Data Frame for better text understanding in machine learning applications. It can also be provided as input for further text cleaning steps such as punctuation removal, numeric character removal or stemming. Machine learning models need numeric data to be trained and make a prediction. Word tokenization becomes a crucial part of the text (string) to numeric data conversion.

```
tokens =  word_tokenize(text1)
lowercase_tokens = [t.lower() for t in tokens]
print(lowercase_tokens)
```

Stemming

It is the process in which the words are converted to its base form..The goal of stemming is to reduce the number of inflectional forms of words appearing in the text. This causes words such as “argue”, "argued", "arguing", "argues" to be reduced to their common stem “argu”. This helps in decreasing the size of the vocabulary space.

There are many ways to perform Stemming, the popular one being the “**Porter Stemmer**” method by Martin Porter.

```
from nltk.stem import PorterStemmer

stemmer = PorterStemmer()
```

Lemmetization

lemmatization lowers the words to word in the present language. Lemmatization is related to stemming, differing in that lemmatization is able to capture canonical forms based on a word's lemma.

For example, stemming the word "better" would fail to return its citation form (another word for lemma); however, lemmatization would result in the following:

$$\text{better} \rightarrow \text{good}$$

It should be easy to see why the implementation of a stemmer would be the less difficult feat of the two.

```
def get_lemma(word):
    """this function is to lemmatize the words in a headline into its root form"""
    lemma = wn.morphy(word)
```

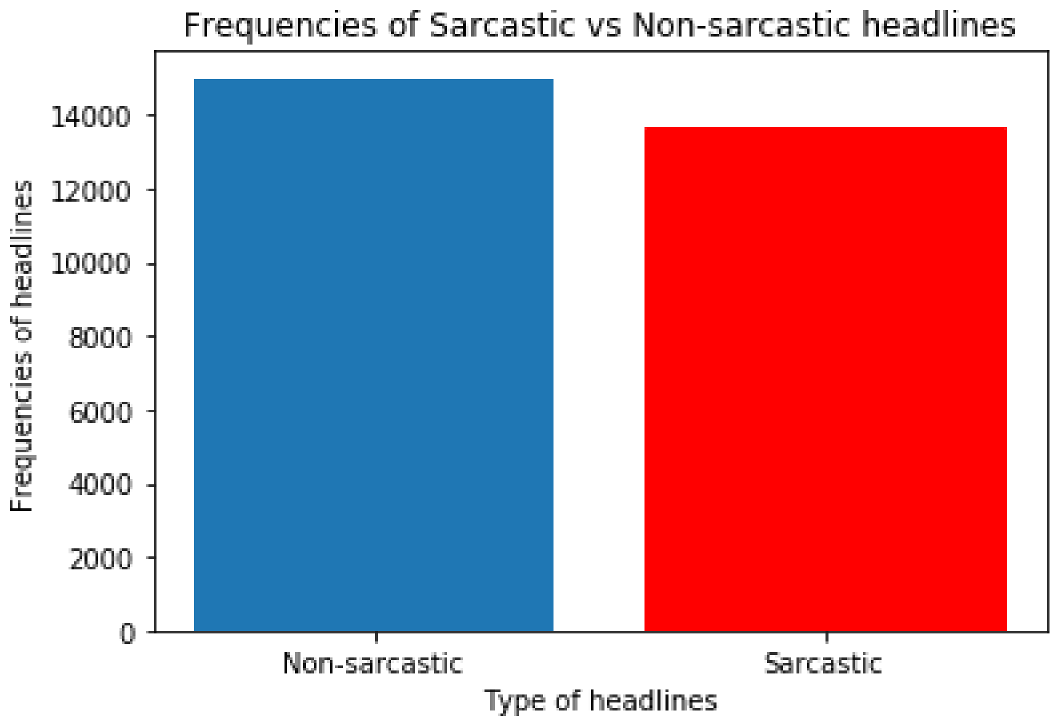
```
if lemma is
    None: return
    word
else:
    return lemma
```

Data Visualisation

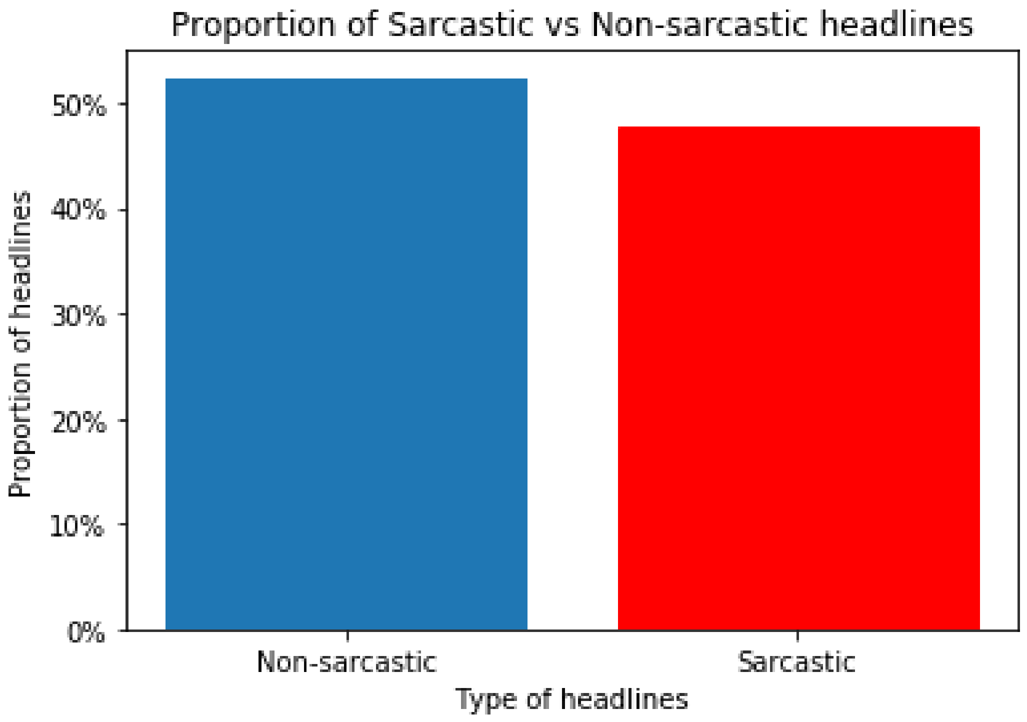
Data visualization is the graphical representation of information and data. By using visual elements like charts, graphs, and maps, data visualization tools provide an accessible way to see and understand trends, outliers, and patterns in data.

Data visualization is another form of visual art that grabs our interest and keeps our eyes on the message. When we see a chart, we quickly see trends and outliers. If we can see something, we internalize it quickly.

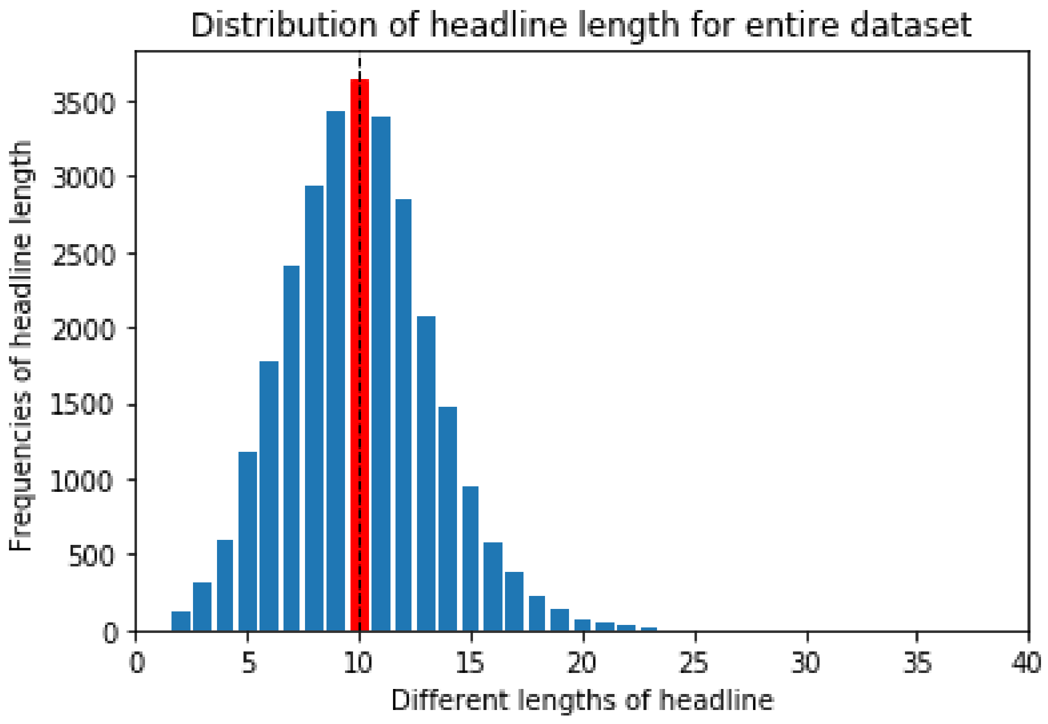
- To check frequencies of Sarcastic vs Non-sarcastic headline2222



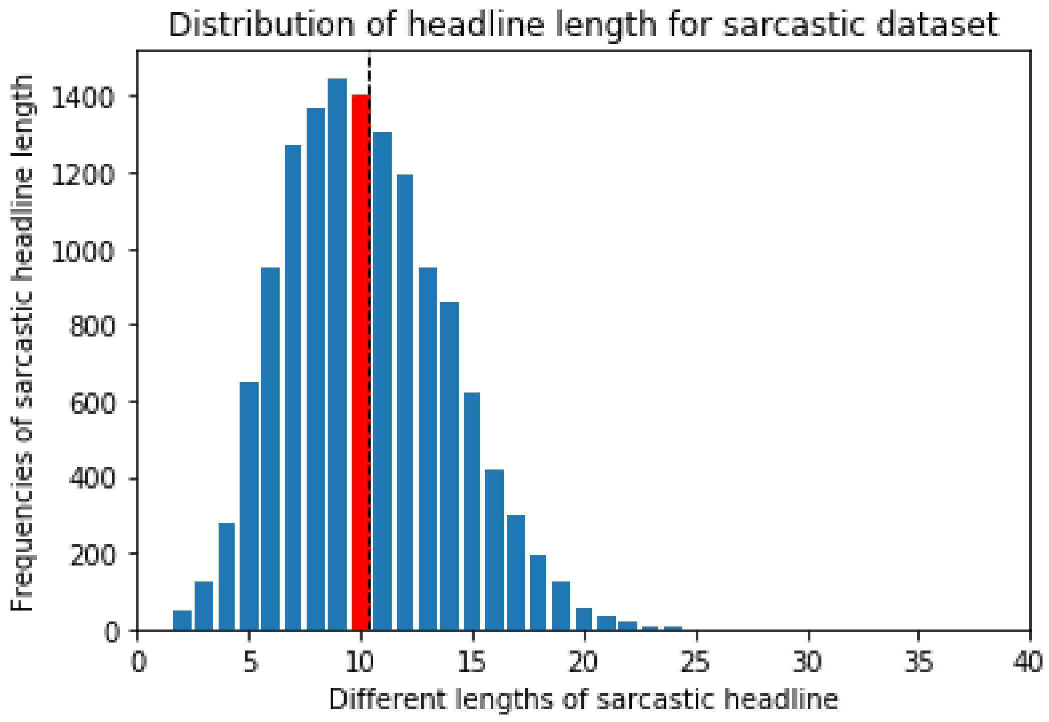
- To check proportion of Sarcastic vs Non-sarcastic headlines



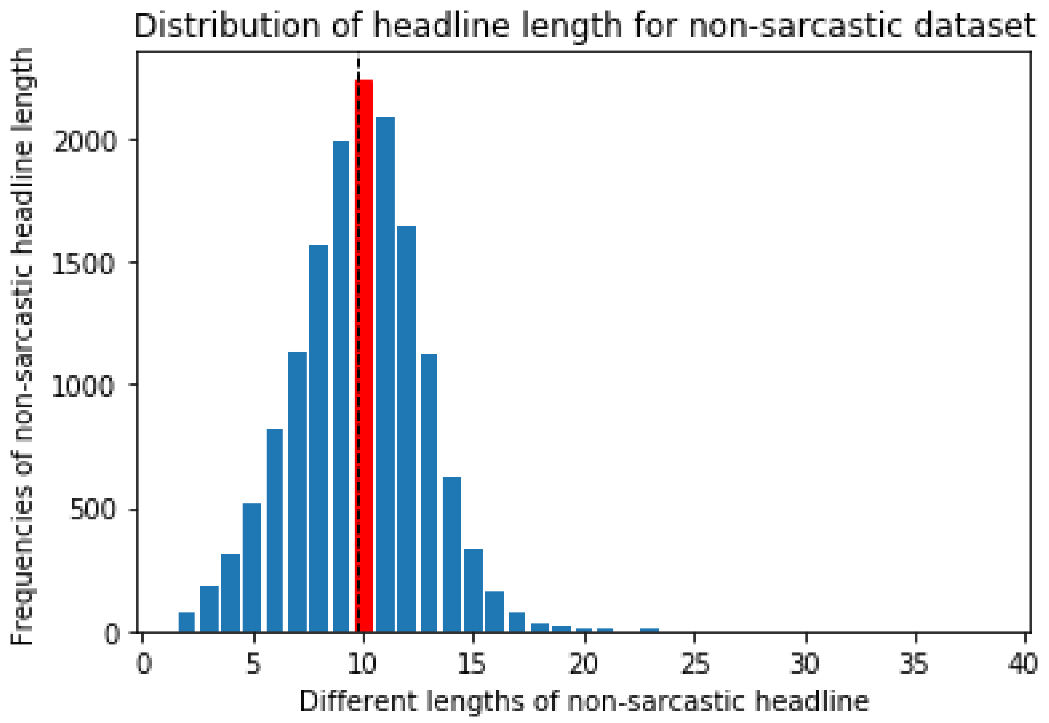
- To check distribution of headlines length of full dsataset.



- check distribution of headline length for sarcastic dataset



- check distribution of headline length for non-sarcastic dataset



Chapter 3
Topic Modelling

In machine learning and natural language processing, a **topic model** is a type of statistical model for discovering the abstract "topics" that occur in a collection of documents. Topic modeling is a frequently used text-mining tool for discovery of **hidden semantic structures** in a text body.

A topic model captures this intuition in a mathematical framework, which allows examining a set of documents and discovering, based on the statistics of the words in each, what the topics might be and what each document's balance of topics is.Topic models are also referred to as probabilistic topic models, which refers to statistical algorithms for discovering the latent semantic structures of an extensive text body.

Topic models can help to organize and offer insights for us to understand large collections of unstructured text bodies. Originally developed as a text-mining tool, topic models have been used to detect instructive structures in data suchas genetic information, images, and networks. They also have applications in other fields such as bioinformatics and computer vision.

We are Using Gensim and Latent Dirichlet Allocation (LDA) for topic modelling

Gensim

Gensim is implemented in Python. Gensim is designed to handle large text collections using data streaming and incremental online algorithms, which differentiates it from most other machine learning software packages that target only in-memory processing.

Gensim includes streamed parallelized implementations of fastText, word2vec and doc2vec algorithms, as well as latent semantic analysis (LSA, LSI, SVD), non-negative matrix factorization (NMF), latent Dirichlet allocation (LDA), tf-idf and random projections.

Gensim is an open source NLP library which can be used for creating and querying a corpus. It works by building word embeddings or vectors which are then used to perform topic modeling.

```
from gensim import corpora
```

```
import pickle
# Convert all headlines into a corpus of words, with each word as a token
dictionary = corpora.Dictionary(text_data)

# Convert each headline into the bag-of-words format
corpus = [dictionary.doc2bow(text) for text in text_data]
pickle.dump(corpus, open('corpus.pkl', 'wb'))
dictionary.save('dictionary.gensim')
```

Latent Dirichlet Allocation (LDA)

Using a powerful statistical method called Latent Dirichlet Allocation (LDA). LDA uses a generative approach to find texts that are similar. It is not a classification technique and does not require labels to infer the patterns. Instead, the algorithm is more of an unsupervised method that uses a **probabilistic model to identify groups of topics**.

In natural language processing, the lda is a generative statistical model that allows sets of observations to be explained by unobserved groups that explain why some parts of the data are similar.

```
# We are asking LDA to find 6 topics in the data
ldamodel = gensim.models.ldamodel.LdaModel(corpus, num_topics = 6, id2word=dictionary, passes=15)
ldamodel.save('model6.gensim')
topics = ldamodel.print_topics(num_words=10)
for topic in topics:
    print(topic)
```

Result

```
(0, '0.015*"woman" + 0.009*"parent" + 0.006*"movie" + 0.006*"child" + 0.006*"video" + 0.005*"watch" + 0.004*"finally" + 0.004*"report" + 0.004*"friend" + 0.004*"never"')

(1, '0.015*"trump" + 0.012*"house" + 0.011*"white" + 0.008*"obama" + 0.007*"president" + 0.006*"congress" + 0.005*"north" + 0.005*"government" + 0.004*"marriage" + 0.004*"twitter"')
```

```
(2, '0.007*"plan" + 0.006*"sexual" + 0.005*"million" + 0.005*"found" + 0.005*"love" + 0.005*"cover" +
0.004*"announce" + 0.004*"death" + 0.004*"francis" + 0.004*"california"')

(3, '0.046*"trump" + 0.016*"donald" + 0.010*"study" + 0.009*"would" + 0.009*"right" + 0.009*"black"
+ 0.008*"american" + 0.008*"find" + 0.007*"report" + 0.006*"people"')

(4, '0.008*"woman" + 0.007*"america" + 0.007*"reveal" + 0.006*"sander" + 0.006*"country" +
0.005*"secret" + 0.005*"bernie" + 0.005*"police" + 0.005*"create" + 0.004*"social"')

(5, '0.011*"school" + 0.009*"change" + 0.008*"report" + 0.007*"world" + 0.007*"clinton" +
0.007*"court" + 0.006*"going" + 0.006*"woman" + 0.005*"student" + 0.005*"better"')
```

Chapter 4
Word Cloud

A **word cloud** is a novelty visual representation of text data, typically used to depict keyword metadata on websites, or to visualize free form text.

Word Cloud provides an excellent option to analyze the text data through visualization in the form of tags, or words, where the importance of a word is explained by its frequency. we will learn how to create word clouds and find important words that can help in extracting insights from the data.

Word are usually single words, and the importance of each word is shown with font size or color.This format is useful for quickly perceiving the most prominent terms to determine its relative prominence**. Bigger term means greater weight**.

Advantages of Wordcloud:-

- **It reveals the essential.** Brand names pop and key words float to the surface.
- **They delight and provide emotional connection.** Both the creation of a word-cloud and the observation of one help to provide an overall sense of the text. The same visceral response doesn't happen when staring at a page of text.
- **They're fast.** Poring over text to develop themes from research takes time.
- **They're engaging.** Visual representation of data tends to have an impact and generates interest amongst the audience

Now we made word cloud using data set because **data cleaning and data pre-processing** is already done.

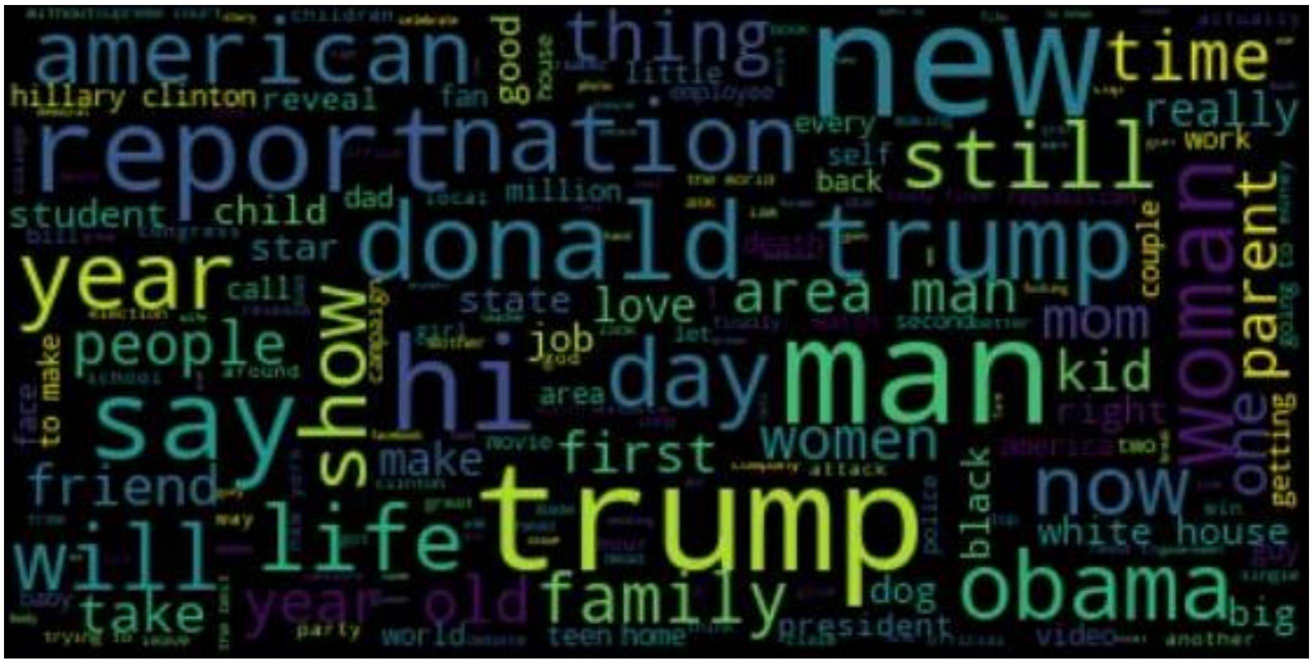
Using these libraries for wordcloud

import collections

```
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator

import matplotlib.cm as cm

%matplotlib inline
```

worldcloud with 200 frequent words

```
stopwords = set(STOPWORDS)

wordcloud_headline = WordCloud(stopwords = stopwords, background_color="black",

                    max_font_size=50, max_words=200).generate(final_text_headline)

plt.figure(figsize = (10,10))

plt.imshow(wordcloud_headline, interpolation='bilinear')

plt.axis("off")

plt.show()
```

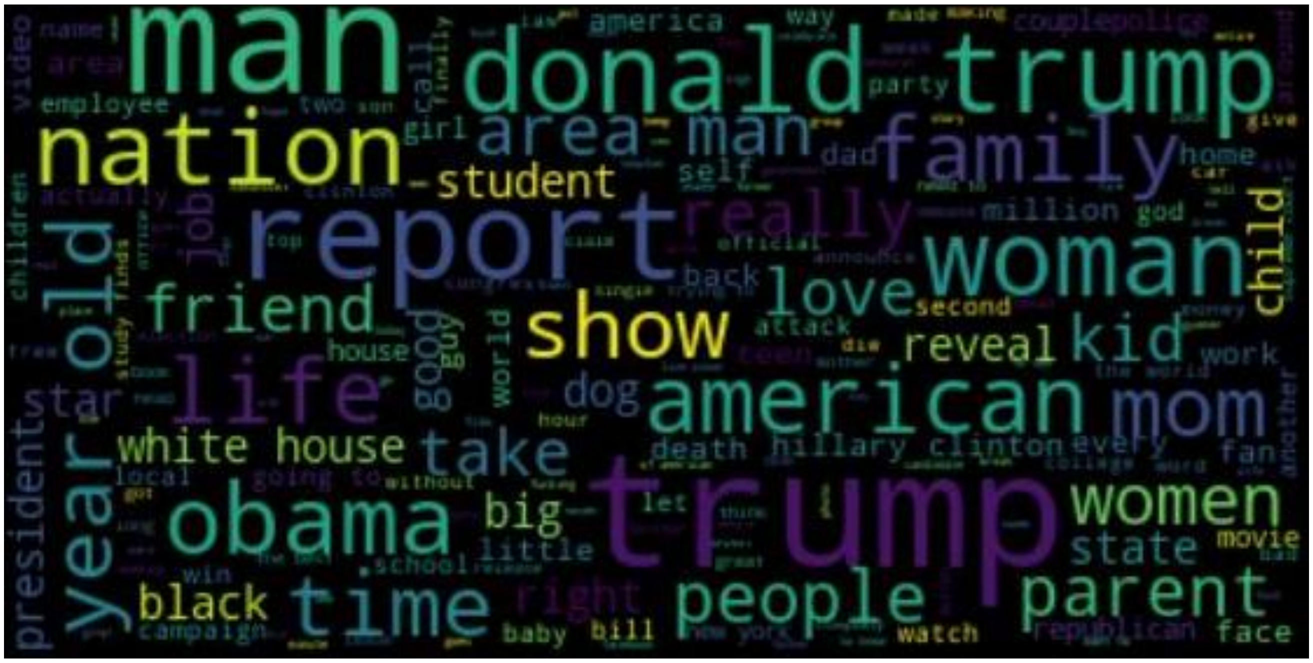


The word cloud displayed above is good, but some of the words are larger than the others. This is because the size of the word in the word cloud is proportional to the frequency of the word inside the corpus

Let us modify the earlier word cloud to include these parameters. Here below utilizes the existing list of stopwords. In the previously built word cloud, words like 'subject', 'will', 'us','enron','re', etc. are common words and do not provide much insight.And **it updates the stopwords** with these words specific to our data.

updated wordcloud

```
stopwords = set(STOPWORDS)
stopwords.update(["hi","will","still","thing","first","say","new","now","one","day","mak
e","getting","find","year"])
wordcloud_headline = WordCloud(stopwords = stopwords, background_color="black",
                        max_font_size=50, max_words=200).generate(final_text_headline)
plt.figure(figsize = (10,10))
plt.imshow(wordcloud_headline, interpolation='bilinear')
plt.axis("off")
plt.show()
```



Top 50 sarcastic words

It is similar to wordcloud but here we find out only top 50 sarcastic words

```
stopwords_removed = [word for word in final_text_sar.split() if word not in stopwords]

lemmatizer = WordNetLemmatizer()

filtered_words_sar = [lemmatizer.lemmatize(t) for t in stopwords_removed]

counted_words_sar = collections.Counter(filtered_words_sar)

word_count_sar = {}

for letter, count in counted_words_sar.most_common(50):

   word_count_sar[letter] = count

for i,j in word_count_sar.items():

      print('Word: {0}, count: {1}'.format(i,j))
```

Result

```
Word: man, count: 1156
Word: report, count: 576
Word: area, count: 490
Word: woman, count: 441
Word: trump, count: 343
Word: time, count: 327
Word: american, count: 316
Word: find, count: 281
Word: study, count: 243
Word: nation, count: 234
Word: u.s, count: 219
Word: life, count: 216
Word: house, count: 212
Word: family, count: 209
```

```
Word: child, count: 200
Word: friend, count: 191
Word: white, count: 185
Word: people, count: 177
Word: going, count: 175
Word: obama, count: 174
Word: last, count: 168
Word: way, count: 161
Word: local, count: 160
Word: go, count:  157
Word: guy, count: 155
Word: nation's, count: 155
Word: good, count: 150
Word: work, count: 148
Word: plan, count: 147
Word: every, count: 147
Word: parent, count: 145
Word: little, count: 144
Word: school, count: 141
Word: home, count: 137
Word: know, count: 136
Word: say, count: 136
Word: announces, count: 134
Word: show, count: 134
Word: right, count: 133
Word: mom, count:  132
Word: clinton, count: 132
Word: job, count:  130
Word: really, count: 129
Word: year, count: 128
Word: around, count: 127
Word: state, count: 124
Word: party, count: 123
Word: employee, count: 122
Word: look, count: 120
Word: bush, count: 120
```

Chapter 5
Model Building for Sarcasm Detection

Sarcasm is "a sharp, bitter, or cutting expression or remark; a bitter gibe or taunt". Sarcasm may employ ambivalence, although sarcasm is not necessarily ironic. Most noticeable in spoken word, sarcasm is mainly distinguished by the inflection with which it is spoken and is largely context-dependent.

The objective of the project make prediction based on headlines as whether **the headline is Sarcastic or not**

Now we made word cloud using data set because **data cleaning and data pre-processing** is already done.

Split the dataset and Creating the Training and Test Datasets

Separating data **into training and testing sets** is an important part of evaluating data mining models. Typically, when you separate a data set into a training set and testing set, most of the data is used for training, and a smaller portion of the data is used for testing.

Conversely, the train-test procedure is not appropriate when the dataset available is small. The reason is that when the dataset is split into train and test sets, there will not be enough data in the training dataset for the model to learn an effective mapping of inputs to outputs. There will also not be enough data in the test set to effectively evaluate the model performance. The estimated performance could be overly optimistic (good) or overly pessimistic (bad).
The original training dataset are split into the two subsets using random selection. This is to ensure that the train and test datasets are representative of the original dataset.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(df['headline'], target, test_size=0.20, random_state=100)

print(df.shape); print(X_train.shape); print(X_test.shape); print(y_train.shape); print(y_test.shape)
```

Converting Text to Word Frequency Vectors with TfidfVectorizer

TFIDF is stands for **term frequency–inverse document frequency**, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. It is often used as a weighting factor in searches of information retrieval, text mining, and user modeling. The tf–idf value increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the corpus that contain the word, which helps to adjust for the fact that some words appear more frequently in general. tf–idf is one of the most popular term-weighting schemes today

Term frequency

The number of times a term occurs in a document is called its term frequency. The weight of a term that occurs in a document is simply proportional to the term frequency.

$$\text{term frequency} = \frac{\text{Term appear in document}}{\text{Total term}}$$

Inverse document frequency

An inverse document frequency factor is incorporated which diminishes the weight of terms that occur very frequently in the document set and increases the weight of terms that occur rarely.

$$\text{inverse document frequency} = \frac{\text{Total document}}{\text{No. of document with term}}$$

$$\boxed{\text{Weight} = \text{term frequency} * \text{inverse document frequency}}$$

```
train_tfIdf = vectorizer_tfidf.fit_transform(X_train.values.astype('U'))

test_tfIdf = vectorizer_tfidf.transform(X_test.values.astype('U'))

from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer_tfidf = TfidfVectorizer(stop_words='english', max_df=0.7)

train_tfIdf.shape,test_tfIdf.shape
```

Model Building

Based on the target variable or feature we select the model. Here in our case sarcasm detection data set, the target variable is comprised of Categorical distribution, so the model will be the Classification model, to predict categorical outcomes.

Classification Model

Classification is a process of categorizing a given set of data into classes, It can be performed on both structured or unstructured data. The process starts with predicting the class of given data points. The classes are often referred to as target, label or categories.

The classification predictive modeling is the task of approximating the mapping function from input variables to discrete output variables. The main goal is to identify which class/category the new data will fall into.

There are two approaches to machine learning: supervised and unsupervised. In a **supervised model**, a training dataset is fed into the classification algorithm. That lets the model know what is, for example, “authorized” transactions. Then the test data sample is compared with that to determine if there is a “fraudulent” transaction. This type of learning falls under “Classification”.

Unsupervised models on the other hand, are fed a dataset that is not labeled and looks for clusters of data points. It can be used to search data for similarities, detect patterns, or identify outliers within a dataset. A typical use case would be finding similar images. Unsupervised models can also be used to find “fraudulent” transactions by looking for anomalies within a dataset. This type of learning falls under “Clustering”

In machine learning, classification is a supervised learning concept which basically categorizes a set of data into classes. The most common classification problems are – speech recognition, face detection, handwriting recognition, document classification, etc. It can be either a binary classification problem or a multi-class problem too. There are a bunch of machine learning algorithms for classification in machine learning.

Import sklearn

Logistic Regression

It is a classification algorithm in machine learning that uses one or more independent variables to determine an outcome. The outcome is measured with a binary variable meaning **it will have only two possible outcomes**.

The goal of logistic regression is to find a best-fitting relationship between the dependent variable and a set of independent variables. It is better than other binary classification algorithms like nearest neighbor since it quantitatively explains the factors leading to classification.

For example, if a child has a temperature of 104F (40C) and they have a rash and nausea then the probability that they have chickenpox might be 80%. A rule of thumb in logistic regression, if the probability is > 50% then the decision is **true**. So in this case, the determination is made that the child has chickenpox.

Logistic regression is used in various fields, including machine learning, most medical fields, and social sciences.

Use Cases

- Identifying risk factors for diseases
- Word classification
- Weather Prediction
- Voting Applications

Model:-

```
from sklearn.linear_model import LogisticRegression
from sklearn import metrics

logisticRegr = LogisticRegression()
logisticRegr.fit(train_tfIdf, y_train)
```

```
➔ LogisticRegression (C=1.0, class_weight=None, dual=False, fit_intercept=True,
        intercept_scaling=1, l1_ratio=None, max_iter=100,
        multi_class='warn', n_jobs=None, penalty='l2',
        random_state=None, solver='warn', tol=0.0001, verbose=0,
        warm_start=False)
```

Naive Bayes Model

It is a classification algorithm based on **Bayes's theorem** which gives an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.

Even if the features depend on each other, all of these properties contribute to the probability independently. Naive Bayes model is easy to make and is particularly useful for comparatively large data sets. Even with a simplistic approach, Naive Bayes is known to outperform most of the classification methods in machine learning. Following is the Bayes theorem to implement the Naive Bayes Theorem.

$$P(C_i|\, x_1, x_2 \dots, x_n) = \frac{P(x_1, x_2 \dots, x_n|C_i).P(C_i)}{P(x_1, x_2 \dots, x_n)}\ for\ 1 < i < k$$

The Naive Bayes classifier requires a small amount of training data to estimate the necessary parameters to get the results. They are extremely fast in nature compared to other classifiers.The only disadvantage is that they are known to be a bad estimator

Use Cases

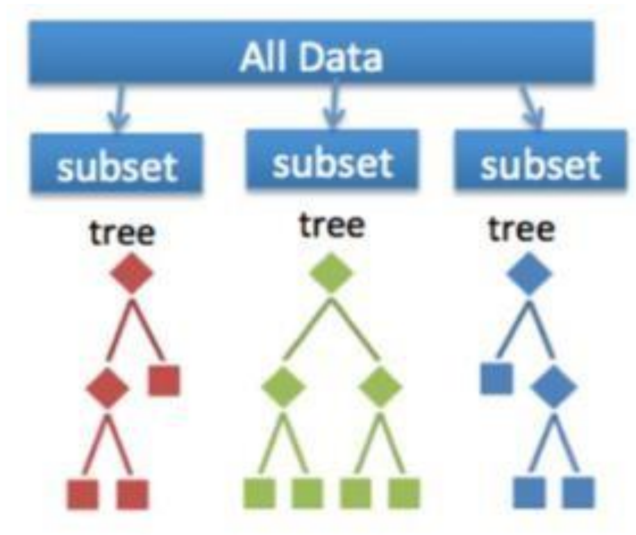
- Disease Predictions
- Document Classification
- Spam Filters
- Sentiment Analysis

Model:-

```
from sklearn.naive_bayes import MultinomialNB

nb_classifier = MultinomialNB()
nb_classifier.fit(train_tfIdf, y_train)
```

Random Forest Model

Random decision trees or random forest are an **ensemble learning method** for classification, regression, etc. It operates by constructing a multitude of decision trees at training time and outputs the class that is the mode of the classes or classification or mean regression of the individual trees.



A random forest is a meta-estimator that fits a number of trees on various subsamples of data sets and then uses an average to improve the accuracy in the model's predictive nature. The sub-sample size is always the same as that of the original input size but the samples are often drawn with replacements.

The goal is to push out bias and group outcomes based upon the most likely positive responses. These collections of positive responses are called **bags**.

The advantage of the random forest is that it is more accurate than the decision trees due to the reduction in the over-fitting. The only disadvantage with the random forest classifiers is that it is quite complex in implementation and gets pretty slow in real-time prediction.

Use Cases

- Industrial applications such as finding if a loan applicant is high-risk or low-risk
- For Predicting the failure of mechanical parts in automobile engines
- Predicting social media share scores
- Performance scores

Model:-

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix

classifier = RandomForestClassifier(n_estimators = 10, criterion = 'entropy',
random_state = 500)
classifier.fit(train_tfIdf, y_train)
```

➔
```
RandomForestClassifier (bootstrap=True, class_weight=None, criterion='entropy',
                        max_depth=None, max_features='auto', max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_split=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, n_estimators=10,
                        n_jobs=None, oob_score=False, random_state=500,
                        verbose=0, warm_start=False)
```

Chapter 6
Model Evaluation

Model Evaluation is an integral part of the model development process. It helps to find the best model that represents our data and how well the chosen model will work in the future. To avoid overfitting use a test set to evaluate model performance.

Model evaluation metrics are required to quantify model performance. The choice of evaluation metrics depends on a given machine learning task. Supervised learning tasks such as classification and regression constitutes a majority of machine learning applications.

Accuracy is a common evaluation metric for classification problems. It's the number of correct predictions made as a ratio of all predictions made. We use sklearn module to compute the accuracy of a classification task.

This project is about classification model . So,according to situation best evaluation matrix is **confusion matrix**.

Confusion matrix

A confusion matrix shows the number of correct and incorrect predictions made by the classification model compared to the actual outcomes in the data. The matrix is *N*x*N*, where *N* is the number of target values. Performance of such models is commonly evaluated using the data in the matrix.

The short explanation of how to interpret a confusion matrix is as follows: The diagonal elements represent the number of points for which the predicted label is equal to the true label, while anything off the diagonal was mislabeled by the classifier. Therefore, the higher the diagonal values of the confusion matrix the better, indicating many correct predictions.

The higher the concentration of observations in the diagonal of the confusion matrix, the higher the accuracy / predictive power of your clustering algorithm

- **Accuracy** : the proportion of the total number of predictions that were correct.
- **Positive Predictive Value or Precision** : the proportion of positive cases that were correctly identified.
- **Negative Predictive Value** : the proportion of negative cases that were correctly identified.
- **Sensitivity or Recall** : the proportion of actual positive cases which are correctly identified.
- **Specificity** : the proportion of actual negative cases which are correctly identified.

$$\boxed{\text{Accuracy} = \frac{\text{True positive} + \text{True negative}}{\text{Total values}}}$$

Models	Confusion matrix	Accuracy
Logistic Regression	$\begin{bmatrix} 2487 & 542 \\ 634 & 2061 \end{bmatrix}$	79.45 %
Naïve Bayes	$\begin{bmatrix} 2560 & 469 \\ 653 & 2042 \end{bmatrix}$	80.39%
Random Forest	$\begin{bmatrix} 2545 & 484 \\ 976 & 1719 \end{bmatrix}$	74.49 %

Observation

- As we can observe that the accuracy rate is high in models like naïve bayes and logistic regression, while comparing to other models.
- Each and every model used here logistic regression,naïve bayes and random forest models are good at predicting the numerical outcomes.
- We select the model with less errors and high accuracy.

From the above observation we can conclude that **NAÏVE BAYES MODEL** is best for our data set.

Reference

1. www.wikipedia.com
2. https://www.pluralsight.com/
3. https://towardsdatascience.com/
4. https://www.kaggle.com/
5. https://www.analyticsvidhya.com/blog