

Recognition of Sarcasm in News Headlines based on Sentiment Analysis using NLP



Rasika Bose (2038)

Akshay Mahajan (2022)

Guide – Dr. Akanksha Kashikar

INDEX

1. Motivation	2
1.1 Data Description	2
2. Related Works	3
3. Methodology	5
3.1 Data Pre- processing	6
3.2 Feature Extraction	7
3.3 Classification algorithms	10
4. Analysis Report	12
4.1 Data Pre-processing and Visualisation	12
4.2 Classification using Machine Learning Models	18
4.2.1 Linear Support Vector Machine	18
4.2.2 Gaussian Naïve Bayes'	20
4.2.3 Logistic Regression	21
4.2.4 Random Forest Classifier	21
4.2.5 Recurrent Neural Network	23
5. Conclusion	27
6. Appendix	28
7. References	37

Recognition of Sarcasm in News Headlines based on Sentiment Analysis using NLP

Rasika Bose (2038), Akshay Mahajan (2022)

Guide – Dr. Akanksha Kashikar

1. Motivation

In the last decade there has been a boom in opinionated textual data both in online resources like social media and in dissemination of information. Sentiment Analysis is used to analyse such opinionated texts which will provide us insights about the emotion or thought process of the author. One of the many challenges faced by sentiment analysis is sarcasm or irony detection. Sarcasm is a complex act of communication that allows speakers the opportunity to express sentiment rich opinions in an implicit way. For example we came across a very funny headline from The Baltimore Sun newspaper “Cows lose their jobs as milk prices drop”. It’s always heart breaking when a cow loses a job but it is not actually the emotion the editor of the article was trying to express.

Although sarcasm is widely used, it is challenging not only for computers but also for humans to detect promptly. This makes the detection of sarcasm a crucial task. Automated sarcasm detection can be seen as a text classification problem. Text data is one of the simplest forms of data. Machine learning algorithms only process numerical data. So the need for feature extraction arises. Here comes in the use of feature extraction using Natural Language Processing (NLP). NLP is a branch of computer science that involves the analysis of human language in speech and text. NLP acts as a translator of sorts between humans and machines. Humans can speak or write normally and NLP translates the language into a form that a machine or algorithm can understand. Though this is a dataset driven project the main motivation for our project is to delve deep into the understanding of text classification methods for sarcasm detection (& sentiment analysis) and understanding and implementing the feature extraction methods of NLP.

It is important to extract useful information from any forms of data, especially unstructured forms like text data. Feature extraction and the proper representation of text for classification purposes is an important factor that affects the accuracy of classification. In this project we plan to explore the use of the Count Vectorizer, TF-IDF and Word2Vec on different supervised learning algorithms and the accuracy of these models will be measured and compared.

1.1. Data description

The Headlines dataset consists of about 28000 text data points where each observation is categorised into two groups, viz., Sarcastic and Non Sarcastic. We are provided with three fields in the dataset described as following:

- `is_sarcastic`: 1 if the record is sarcastic, otherwise 0

- `headline`: the headline of the news article
- `article_link`: link to the original news article. Useful in collecting supplementary data

The data is provided in JSON (JavaScript Object Notation) format which we have converted to pandas object format using Python. The dataset after dropping the `article_link` field looks as shown in Figure 1.

	headline	is_sarcastic
0	former versace store clerk sues over secret 'b...	0
1	the 'roseanne' revival catches up to our thorn...	0
2	mom starting to fear son's web series closest ...	1
3	boehner just wants wife to listen, not come up...	1
4	j.k. rowling wishes snake happy birthday in th...	0
...
26704	american politics in moral free-fall	0
26705	america's best 20 hikes	0
26706	reparations and obama	0
26707	israeli ban targeting boycott supporters raise...	0
26708	gourmet gifts for the foodie 2014	0

26709 rows × 2 columns

Figure 1

This News Headlines dataset for Sarcasm Detection is collected from two new websites. *TheOnion* aims at producing sarcastic versions of current events and they collected all the headlines from News in Brief and News in Photos categories (which are sarcastic). They also collected real (and non-sarcastic) news headlines from *HuffPost*.

The link for the dataset is provided below: <https://www.kaggle.com/rmisra/news-headlines-dataset-for-sarcasm-detection>

2.Related works

2.1. Data preprocessing

Text extraction describes the process of taking words from text data and transforming them into a numerical feature set that is useful for a machine learning classifier [Sindhu et al, 2014]. Extracting features is useful for all analyses involving text data in all domains. In the work detailed in Sindhu et al, 2014 as well as Tungthamthiti et al, 2018 the impact of pre-processing is discussed and reviewed. The common **data preprocessing** techniques used by both the research papers were: 1)**Tokenization** 2) **Removal of stop words** 3)**Stemming and lemmatization** 4)**P-O-S tagging**.

2.2. Feature extraction

There are several techniques to extract features from text and in Sindhu et al, 2014 we can see that sarcasm is hinted in different ways. Here the classes of sarcastic texts are provided below:

Class No	Feature	Example
Class 1	Co-existence of positive and negative	It is not sweet at all, it is called delicate sweetness.
Class 2	Positive sentence followed by a negative sentence and vice versa	Sweet and delicious oranges. That's why they are rotting in a delivery box.
Class 3	A dilemma in the sentence	I was thinking whether to buy the product because price is (good/expensive).
Class 4	Positive phrase followed by a negative phrase and vice versa	Delivery was good but the product was not.
Class 5	Comparison between bad and worse meaning in the situation	I regret that I bought this product, but it's better than losing money again.
Class 6	Comparison with a better product	I know more delicious oranges that are sold nearby
Class 7	It implies a negative meaning to the target product in the review	If this were a disposable product, I would get satisfaction about it.
Class 8	No specific positive and negative point.	I can use it so so.

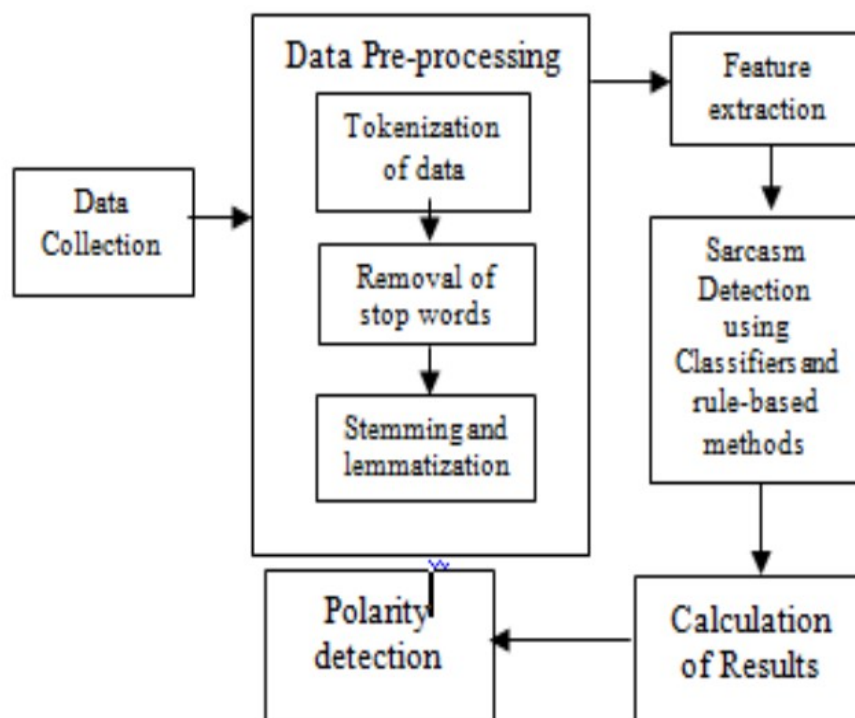
The most common methods of feature extraction used in both Sindhu et al, 2014 and Tungthamthiti et al, 2018 are **1)Bag of Words / Count Vectorizer, 2)Term Frequency – Inverse Document Frequency (TF-IDF) and 3)Word2Vec 4)N-gram**. These methods will be further discussed in methodology.

2.3)Sarcasm detection

The detection of sarcasm is very crucial in the domain of sentiment analysis and opinion mining. Different machine learning algorithms (both supervised and semi supervised) have been applied to various problems Sindhu et al, 2014. After data preprocessing and feature extraction supervised learning models like Naïve Bayes Classifier, Support Vector Machine, decision tree, random forest, gradient boosting, ensemble classifiers are used and their accuracy for each of the models is compared Sindhu et al , 2014. In Tunghamthiti et al, 2018 we see a new method for recognition of sarcasm in tweets. The methods used here are **sentiment analysis, concept level knowledge expansion, coherence of sentences and machine learning classifications**. On applying SVM classifier trained with N-gram features on the extracted feature vector in Tunghamthiti et al, 2018 they get an accuracy of 63.42% accuracy.

This project uses some of the methods described in the previous works.

3.Methodology



3.1 Data Pre-processing

The first step in the pre-processing pipeline is the conversion of all the text to lower case. This is done to achieve uniformity of words. Unwanted symbols like digits and newlines are also removed. After these, the texts are tokenized.

Tokenization – Tokenization is the process of tokenizing or splitting a string, text into a list of tokens. One can think of token as parts like a word is a token in a sentence, and a sentence is a token in a paragraph. For our project we will first tokenize each headline paragraphs into sentences and then tokenize each of the sentences into words. (Natural language Understanding by James Allen)

Eg- Input – “Never give up” after tokenization “Never”, “give”, “up”.

Stopwords – Stopwords are words that are commonly used in any language. They do not contribute any contextual meaning to the data and as such should be eliminated. Example of stop words are “is , I , am, a, an, the etc” . Removal of stopwords is followed by Stemming and lemmatization.

Eg- Input-“She is the president of India”, output after removing stop words – “president India”

Stemming and lemmatization - Stemming and lemmatization is the process of converting the words into their root words so that they can be analysed as a single item. For example- for the set of words “finally, final, finalized” stemming will give an output “fina” as the root word, which has no contextual meaning whereas lemmatization will give an output “final” which holds a contextual meaning and can be of more use in sentiment analysis.

Another example which shows semantic advantage of **lemmatization** over **stemming** is the words “historic, historical”. For this stemming gives an output “histori” but lemmatization gives “history”. (Allen, 2002)

3.2 Feature Extraction

Count Vectorizer/ Bag Of Words - The Count Vectorizer is a simple means used to tokenize and build vocabulary of known words. It is also used to encode new documents using the vocabulary that has been created. The product of the count vectorizer is an encoded vector which has the length of the entire vocabulary of the document and a count for the number of times each word appeared in the document. Count Vectorizer will be implemented using scikit-learn in python environment. (Allen, 2002)

Eg- The following models a text document using bag-of-words/ Count Vectorizer. Here are two simple text documents:

- (1) John likes to watch movies. Mary likes movies too.
- (2) Mary also likes to watch football games.

Based on these two text documents, a list is constructed as follows for each document:

"John","likes","to","watch","movies","Mary","likes","movies","too"

"Mary","also","likes","to","watch","football","games"

Representing each bag-of-words as a [JSON object](#), and attributing to the respective [JavaScript](#) variable:

```
BoW1 = {"John":1,"likes":2,"to":1,"watch":1,"movies":2,"Mary":1,"too":1};  
BoW2 = {"Mary":1,"also":1,"likes":1,"to":1,"watch":1,"football":1,"games":1};
```

Each key is the word, and each value is the number of occurrences of that word in the given text document.

The order of elements is free, so, for

example `{"too":1,"Mary":1,"movies":2,"John":1,"watch":1,"likes":2,"to":1}` is also equivalent to *BoW1*. It is also what we expect from a strict *JSON object* representation.

Note: if another document is like a union of these two,

(3) John likes to watch movies. Mary likes movies too. Mary also likes to watch football games.

its JavaScript representation will be:

BoW3 =

```
{"John":1,"likes":3,"to":2,"watch":2,"movies":2,"Mary":2,"too":1,"also":1,"football":1,"games":1};
```

So, as we see in the [bag of words algebra](#), the "union" of two documents in the bags-of-words representation is, formally, the [disjoint union](#), summing the multiplicities of each element.

Term Frequency – Inverse Document Frequency (TF-IDF)

TF-IDF is a popular and well recognized method for the evaluation of the importance of a word in a document. Term Frequency measures the frequency of a term in a document. For a variety of documents with different lengths, the probability of a word occurring more times in a longer document is present. Due to this fact, normalization is required.

$$TF(t) = \frac{N}{T}$$

Where **N = Number of times term t appears in a document and**

T= Total number of terms in the document

The Inverse Document Frequency is the measure of importance of a term. This is done to reduce the influence of words e.g. 'of', 'the' that could be used multiple times in a document but have no real meaning and importance in context. (no semantic meaning)

$$IDF(t) = \log_e \frac{TD}{ND}$$

where **TD = Total number of documents and**

ND = Number of documents with term t in it.

For most traditional applications the bag-of-words model was used. However, words that occur frequently in the document were given significant weights in the Bag of Words model

and this affects the accuracy of these results. The use of TF-IDF is introduced to solve these problems of the typical Bag Of Words Model. The IDF of an infrequent term due to the log applied is high and the IDF for common words is low . We are planning to use three levels of TF-IDF for this analysis. They are the Word-level, N-Gram and Character Level TF-IDF. All these are implemented using the scikit-learn package of python. (Allen, 2002)

Word2Vec

Word embedding is one of the most popular representations of document vocabulary. It is capable of capturing context of a word in a document, semantic and syntactic similarity, relation with other words, etc.

What are word embeddings exactly? Loosely speaking, they are vector representations of a particular word. Having said this, what follows is how do we generate them? More importantly, how do they capture the context?

Word2Vec is one of the most popular techniques to learn word embeddings using shallow neural network. It was developed by Tomas Mikolov in 2013 at Google.

Let's tackle this part by part.

Why do we need them?

Consider the following similar sentences: *Have a good day* and *Have a great day*. They hardly have different meaning. If we construct an exhaustive vocabulary (let's call it V), it would have $V = \{\text{Have, a, good, great, day}\}$.

Now, let us create a one-hot encoded vector for each of these words in V. Length of our one-hot encoded vector would be equal to the size of V (=5). We would have a vector of zeros except for the element at the index representing the corresponding word in the vocabulary. That particular element would be one. The encodings below would explain this better.

Have = $[1,0,0,0,0]^t$; a= $[0,1,0,0,0]^t$; good= $[0,0,1,0,0]$; great= $[0,0,0,1,0]$; day= $[0,0,0,0,1]^t$

If we try to visualize these encodings, we can think of a 5 dimensional space, where each word occupies one of the dimensions and has nothing to do with the rest (no projection along the other dimensions). This means ‘good’ and ‘great’ are as different as ‘day’ and ‘have’, which is not true.

Our objective is to have words with similar context occupy close spatial positions. Mathematically, the cosine of the angle between such vectors should be close to 1, i.e. angle close to 0. (Karani,2018)

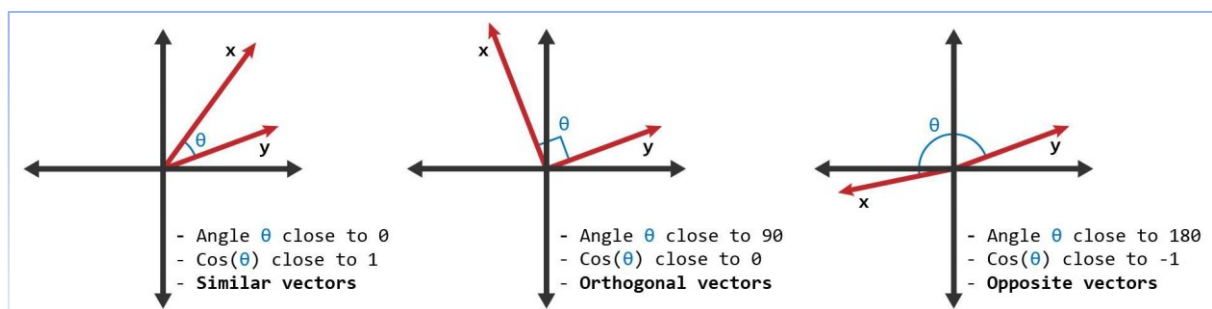


Figure 2

3.3 Classification algorithms

We will be applying multiple classification algorithms on the text to vector converted dataset.

A couple of classification algorithms we have chosen for this dataset to train and validate with the output are:

- i. **Naïve Bayes:** The Naive Bayes is a powerful algorithm used for classifying data based on probabilities. This algorithm is based on the Bayes theorem in statistics. The classification of the data is done using various probabilities. It is fast and scalable but is not without disadvantages as it assumes independence among predictors. It works well for small data sets and has been known to give great results. The Naive Bayes Classifier employed in the analysis is built using the scikit-learn package available in python.
- ii. **Random Forest:** Random forest is a Supervised Machine Learning Algorithm that is used widely in Classification and Regression problems. It builds decision trees on different samples and takes their majority vote for classification

and average in case of regression. One of the most important features of the Random Forest Algorithm is that it can handle the data set containing continuous variables as in the case of regression and categorical variables as in the case of classification. It performs better results for classification problems. (James et al, 2013)

- iii. **Decision Tree:** Decision tree is the most interpretable and popular tool for classification and prediction. A Decision tree is a flowchart like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label.
- iv. **Support Vector Machines (SVM):** The Support Vector Machine is efficient for both classification and regression. It is known to give good results for classification. We will check if the classes are separated by boundaries found by the algorithm.
- v. **Logistic Regression:** Logistic Regression is a popular classification algorithm. It belongs to the class of Generalized Linear Models. Its loss function is the sigmoid function which minimizes the results to be a value between 0 and 1. The Logistic Regression Model will be implemented for this analysis using the function available in the scikit-learn package.
- vi. **Convolutional Neural Networks (CNN):** CNNs are very commonly used in text classification problems due to their success and great results. This has contributed to the decision to use CNN for this aspect of the experiment.

Polarity Detection: During this process, the polarity of the headline or statement is identified and labelled as sarcastic or not sarcastic. This polarity detection is done by identifying positive and negative words in the same sentence. There are in total 8 classes of sarcastic texts, which have been provided in Section 2.2.

4. Analysis Report

1. DATA PRE-PROCESSING AND VISUALISATION –

- The data is provided in JSON (JavaScript Object Notation) format which we have converted to pandas object format using Python. Then we drop the article_link field as it does not hold any relevant information for the analysis. The dataset in pandas object format will appear as shown in **Figure 3**:

	headline	is_sarcastic
0	former versace store clerk sues over secret 'b...	0
1	the 'roseanne' revival catches up to our thorn...	0
2	mom starting to fear son's web series closest ...	1
3	boehner just wants wife to listen, not come up...	1
4	j.k. rowling wishes snape happy birthday in th...	0
...
26704	american politics in moral free-fall	0
26705	america's best 20 hikes	0
26706	reparations and obama	0
26707	israeli ban targeting boycott supporters raise...	0
26708	gourmet gifts for the foodie 2014	0

26709 rows x 2 columns

Figure 3

- Then we check for the **presence of missing values** in the dataset. We get no missing values for numeric attributes of the dataset.
- We plot a basic **count plot** of the is_sarcastic feature.

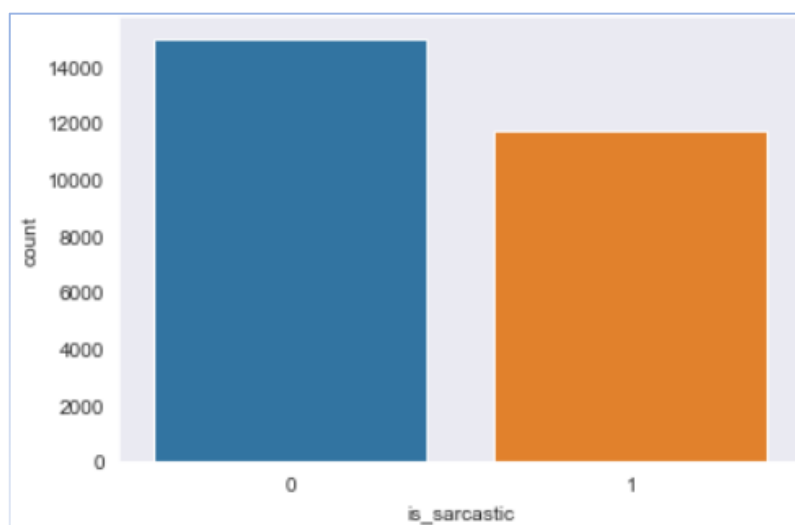
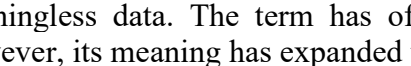
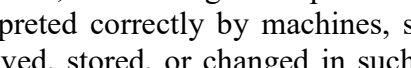


Figure 4

So, we can see that (from Figure 4) the dataset is more or less balanced.

- **Removal of stopwords and punctuations**

Stopwords are the words which do not add much meaning to a sentence. They can safely be ignored without sacrificing the meaning of the sentence. For example, the words like the, he, have etc.

- We also clean the dataset feature Headlines by removing **noisy texts**. **Noisy data** is meaningless data. The term has often been used as a synonym for corrupt data. However, its meaning has expanded to include any data that cannot be understood and interpreted correctly by machines, such as unstructured text. Any data that has been received, stored, or changed in such a manner that it cannot be read or used by the program that originally created it can be described as noisy. Example in our context the words like “hey”.

- **WORDCLOUD** - **Word Cloud** or **Tag Clouds** is a visualization technique for texts that are natively used for visualizing the tags or keywords from the websites. These keywords typically are single words that depict the context of the webpage the word cloud is being made from. These words are clustered together to form a **Word Cloud**.


Each word in this cloud has a variable font size and colour tone. Thus, this representation helps to determine words of prominence. **A bigger font size of a word portrays that it is more prominent relative to other words in the cluster.** Word Cloud can be built in varying shapes and sizes based on the creators' vision. The number of words plays an important role while creating a Word Cloud. More number of words does not always mean a better Word Cloud as it becomes cluttered and difficult to read. A Word Cloud must always be semantically meaningful and must represent what it is meant for.

Although, there are different ways by which Word Clouds can be created but the most widely used type is by using the **Frequency of Words** in our corpus. And thus, we will be creating our Word Cloud for both **not sarcastic and sarcastic texts** separately by using the **Frequency type**.

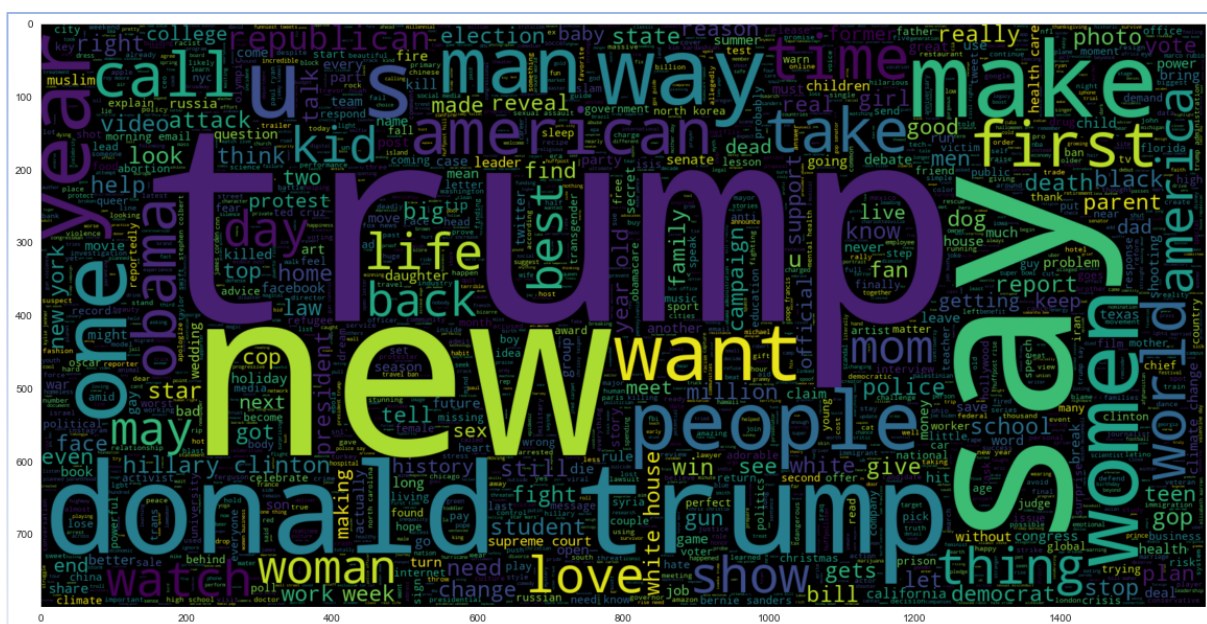


Figure 5 - WORDCLOUD FOR TEXT THAT IS NOT SARCASTIC (LABEL - 0)

From **Figure 7** we **cannot** see much of a **difference** in the **distribution of number of characters** in **Sarcastic versus Not Sarcastic texts**. So the character length does not help much in our further analysis of whether a data point is Sarcastic or Not sarcastic.

- **Plotting the number of words in text**

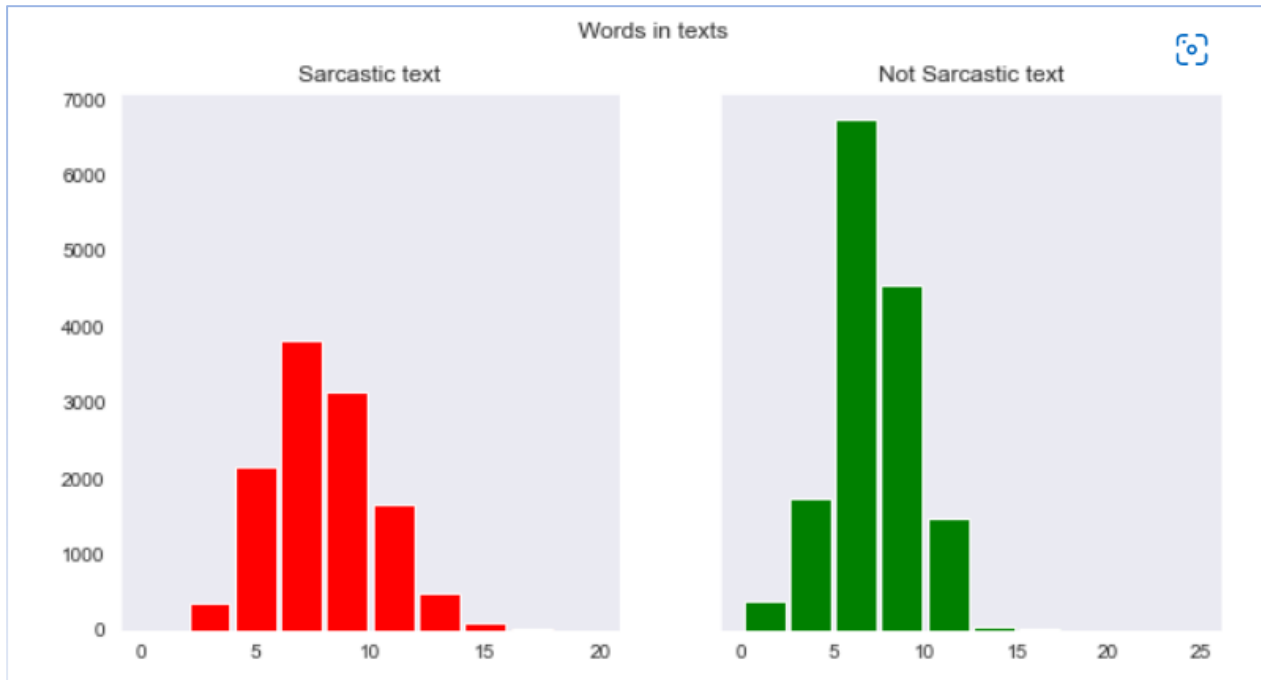


Figure 8 - Number of words in Sarcastic texts vs Not Sarcastic texts.

From **Figure 8** we **cannot** see a significant **difference** in the **distribution of number of words** in **Sarcastic versus Not Sarcastic texts**. So the word length of a text does not help much in our further analysis - whether a data point is Sarcastic or Not sarcastic.

- **Plotting average word length in a text**

From **Figure 9** we **cannot** see a significant **difference** in the **distribution of average word length** in **Sarcastic versus Not Sarcastic texts**. The **median** average word length is about **6** for both Sarcastic and Not Sarcastic Headlines So the average word length of a text does not help much in our further analysis ,i.e. whether a data point is Sarcastic or Not sarcastic.

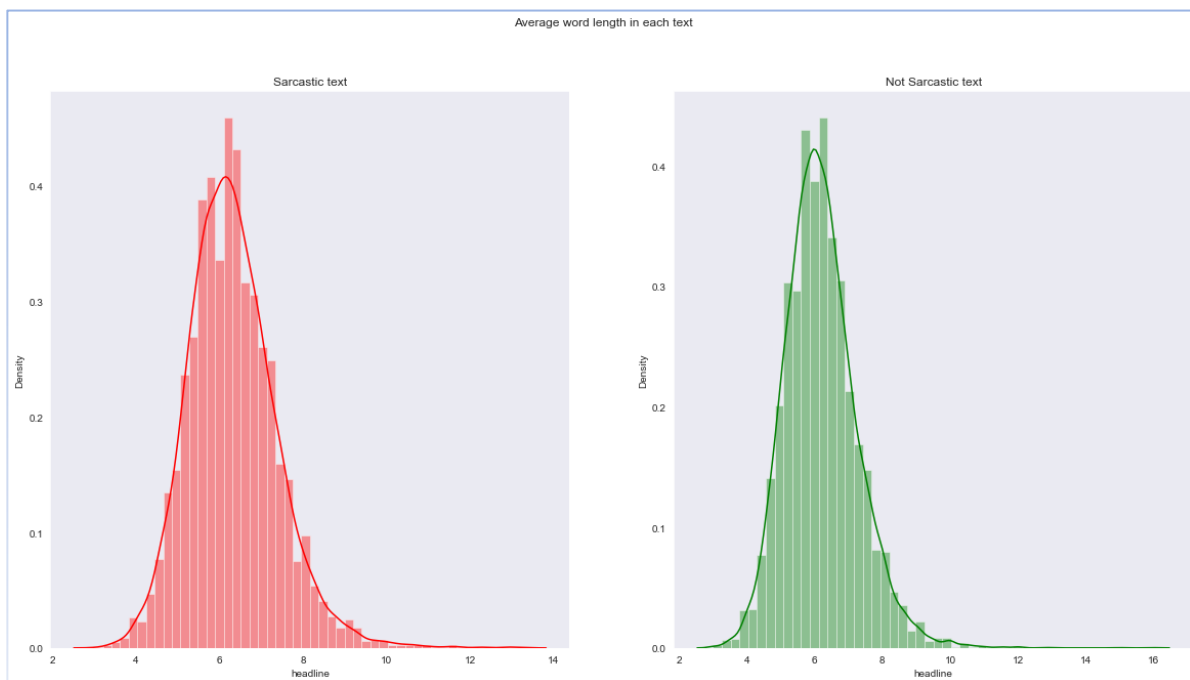


Figure 9 - Average word length in each text: Sarcastic vs Not Sarcastic

- **Stemming** - Stemming is a technique used to extract the base form of the words by removing affixes from them. It is just like cutting down the branches of a tree to its stems. For example, the stem of the words eating, eats, eaten is eat. Stemming is required for performing further analysis and gets semantic meaning of words using **word embedding techniques**.
- After replacing special symbols and digits in headline column with space (using regular Expression in NLP) we perform stemming. The output we get is given in **Figure 10**.

```
0    former versac store clerk sue secret black cod...
1    roseann reviv catch thorni polit mood better wors
2    mom start fear son s web seri closest thing gr...
3    boehner want wife listen come altern debt redu...
4          j k rowl wish snape happi birthday magic way
Name: headline, dtype: object
```

Figure 10 - First five entries of the headline column after Stemming.

- **Lemmatization** - Lemmatisation in linguistics is the process of grouping together the inflected forms of a word so they can be analysed as a single item, identified by the word's lemma, or dictionary form.
- After replacing special symbols and digits in headline column with space (using regular Expression in NLP) we perform stemming. The output we get is given in **Figure 11**.

```
: 0    former versace store clerk sue secret black co...
  1    roseanne revival catch thorny political mood b...
  2    mom starting fear son s web series closest thi...
  3    boehner want wife listen come alternative debt...
  4    j k rowling wish snape happy birthday magical way
Name: headline, dtype: object
```

Figure 11 : First five entries of the headline column after lemmatization

- From further analysis using various machine learning algorithms for our dataset we see that both stemming and lemmatization give more or less similar accuracy for classification.
- The final data pre-processing step is to apply **Term Frequency and Inverse Document Frequency** to the feature (Headlines column). This **transforms text to feature vectors that can be used as input to estimator or prediction models**. It captures importance of terms in a text file in a vector format. Here we take the maximum features to perform TF-IDF as **5000**.
- Now the text is finally converted to a numeric feature matrix on which we can fit machine learning and deep learning algorithms for classification.

2. CLASSIFICATION USING MACHINE LEARNING MODELS –

The Machine Learning algorithms we used for classification and cross validation are:

- 1) **Linear support vector classifier**
- 2) **Gaussian Naive Bayes**
- 3) **Logistic Regression**
- 4) **Random Forest Classifier**

To check the accuracy of each of the classifier we make use of both **train test split** and **10 fold cross validation** technique. 10 fold cross validation is chosen as the size of the dataset is large.

Train-test split- The train-test split procedure is used to estimate the performance of machine learning algorithms when they are used to make predictions on data not used to train the model.

It is a fast and easy procedure to perform, the results of which allow us to compare the performance of machine learning algorithms for our classification or predictive modelling problem. Although simple to use and interpret, there are times when the procedure should not be used, such as when we have a small dataset and situations where additional configuration is required, such as when it is used for classification and the dataset is not balanced.

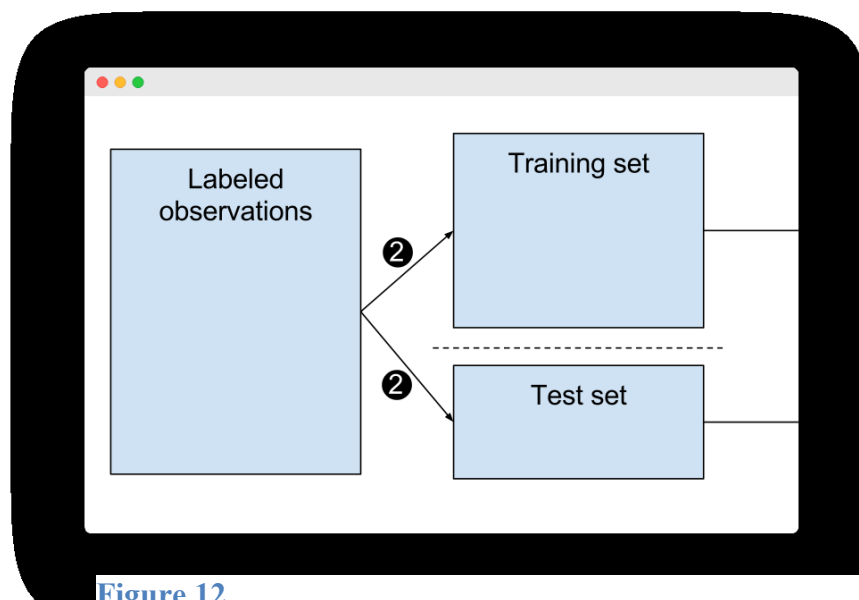


Figure 12

K-fold Cross Validation -

K-fold cross-validation approach divides the input dataset into K groups of samples of equal sizes. These samples are called **fold**s. For each learning set, the prediction function uses K-1 folds, and the rest of the folds are used for the test set. This approach is a very popular CV approach because it is easy to understand.

The steps for k-fold cross-validation are:

- Split the input dataset into K groups
- For each group:
 - Take one group as the reserve or test data set.
 - Use remaining groups as the training dataset
 - Fit the model on the training set and evaluate the performance of the model using the test set
 - Averaging the error

1. Linear support vector classifier

“Support Vector Machine” (SVM) is a **supervised machine learning** algorithm that can be used for both classification and regression challenges. However it is mostly used in classification problems. In the SVM algorithm, we plot each data item as a point in n-dimensional space (where n is a number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiates the two classes very well. We use here Linear Support Vector Classifier as we have a dataset which is linearly separable into **Sarcastic** and **Not Sarcastic**.

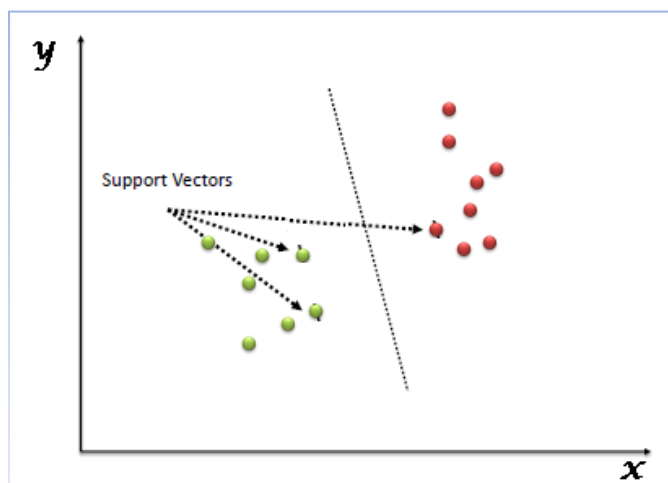


Figure 13 : Illustration of decision boundary separation in SVM

Analysis-

- We apply Linear support vector classifier in Python using `sklearn.svm` package using `LinearSVC` function.
- We use both train test split and 10 fold cross validation approach to compute accuracy. We get the following output:

Average 10 fold Cross Validation score: 0.78546

Testing score from train-test split: 0.79041

2. Gaussian Naïve Bayes

This classifier is employed when the predictor values are continuous and are expected to follow a Gaussian distribution. The Naive Bayes is a powerful algorithm used for classifying data based on probabilities. This algorithm is based on the Bayes theorem in statistics. The classification of the data is done using various probabilities. It is fast and scalable but is not without disadvantages as it assumes independence among predictors. It works well for small data sets and has been known to give great results.

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

Analysis-

- We apply Gaussian Naïve Bayes's classifier in Python using `sklearn.naive_bayes` package using `GaussianNB` function.
- We use both train test split and 10 fold cross validation approach to compute accuracy. We get the following output:

Average 10 fold Cross Validation score: 0.69456

Testing score from train-test split: 0.72230

3. Logistic Regression

Logistic regression is a supervised learning classification algorithm used to predict the probability of a target variable. The nature of target or dependent variable is dichotomous, which means there would be only two possible classes.

In simple words, the dependent variable is binary in nature having data coded as either 1 (stands for success/yes) or 0 (stands for failure/no).

Mathematically, a logistic regression model predicts $P(Y=1)$ as a function of X . It is one of the simplest ML algorithms that can be used for various classification problems such as spam detection, Diabetes prediction, cancer detection etc.

Analysis-

- We apply Logistic Regression classifier in Python using `sklearn.linear_model` package using **LogisticRegression** function.
- We use both train test split and 10 fold cross validation approach to compute accuracy. We get the following output:

Average 10 fold Cross Validation score: 0.78902

Testing score from train-test split: 0.78817

4. Random Forest Classifier

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of **ensemble learning**, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.

As the name suggests, **"Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset."** Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.

The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.

The Working process can be explained in the below steps and diagram:

Step-1: In random forest n number of random records are taken from the data set having k number of records.

Step-2: Individual decision trees are constructed for each sample.

Step-3: Each decision tree will generate an output.

Step-4: Final output is considered based on **Majority Voting or Averaging** for Classification and regression respectively.

Example: Suppose there is a dataset that contains multiple fruit images. So, this dataset is given to the Random forest classifier. The dataset is divided into subsets and given to each decision tree. During the training phase, each decision tree produces a prediction result, and when a new data point occurs, then based on the majority of results, the Random Forest classifier predicts the final decision. Consider the below image:

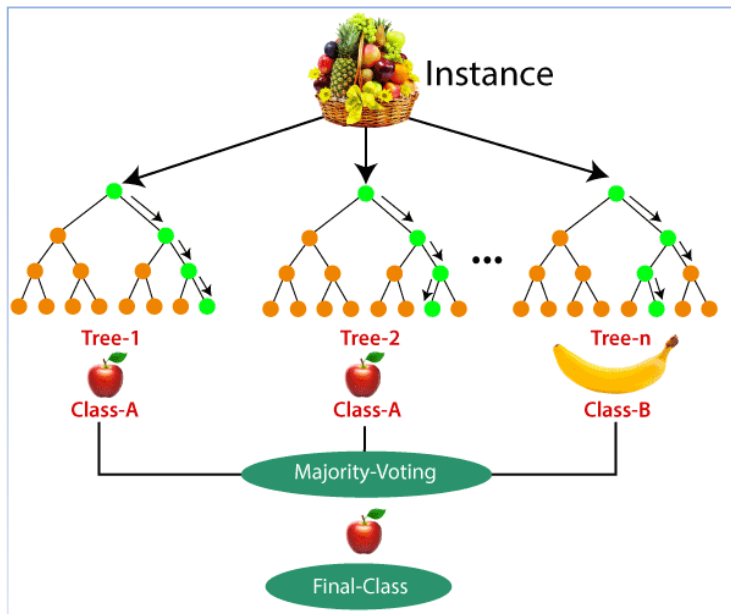


Figure 14: Diagrammatic representation of random forest for the given example.

Analysis-

- We apply Random Forest Classifier in Python using **sklearn.ensemble** package using **RandomForestClassifier** function.
- We consider **25 trees** at each iteration of the Random Forest algorithm.
- We use both train test split and 10 fold cross validation approach to compute accuracy. We get the following output:

Average 10 fold Cross Validation score: 0.74708

Testing score from train-test split: 0.74251

5. Recurrent Neural Network

What is Neural Network?

Neural Networks are set of algorithms which closely resemble the human brain and are designed to recognize patterns. They interpret sensory data through a machine perception, labelling or clustering raw input. They can recognize numerical patterns, contained in vectors, into which all real-world data (images, sound, text or time series), must be translated. Artificial neural networks are composed of a large number of highly interconnected processing elements (neuron) working together to solve a problem.

An ANN usually involves a large number of processors operating in parallel and arranged in tiers. The first tier receives the raw input information — analogous to optic nerves in human visual processing. Each successive tier receives the output from the tier preceding it, rather than from the raw input — in the same way neurons further from the optic nerve receive signals from those closer to it. The last tier produces the output of the system.

What is Recurrent Neural Network (RNN)?

Recurrent Neural Network is a generalization of feedforward neural network that has an internal memory. RNN is recurrent in nature as it performs the same function for every input of data while the output of the current input depends on the past one computation. After producing the output, it is copied and sent back into the recurrent network. For making a decision, it considers the current input and the output that it has learned from the previous input.

Unlike feedforward neural networks, RNNs can use their internal state (memory) to process sequences of inputs. This makes them applicable to tasks such as unsegmented, connected handwriting recognition or speech recognition. In other neural networks, all the inputs are independent of each other. But in RNN, all the inputs are related to each other. (Mittal,2019)

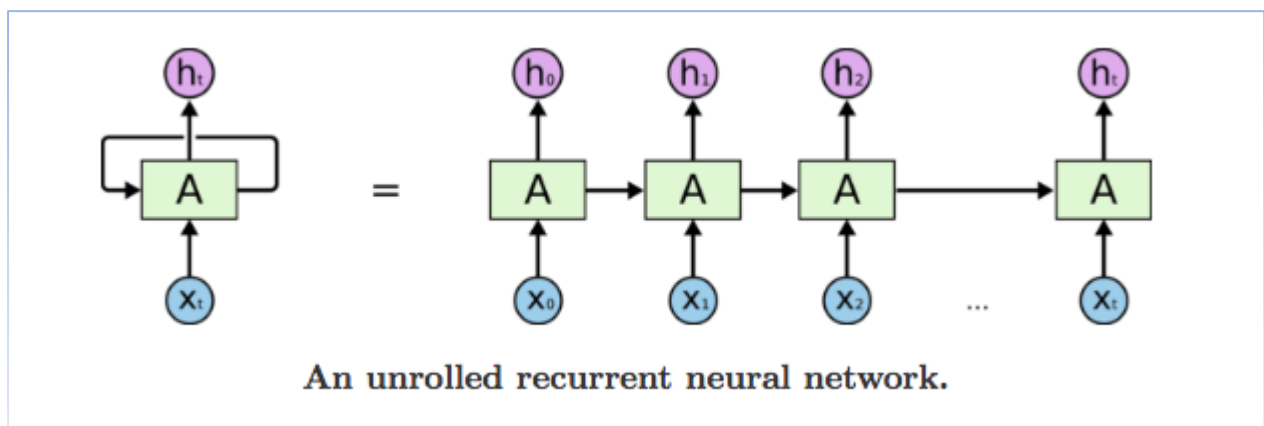


Figure 15: Illustration of how RNN works

First, it takes the $X(0)$ from the sequence of input and then it outputs $h(0)$ which together with $X(1)$ is the input for the next step. So, the $h(0)$ and $X(1)$ is the input for the next step.

Similarly, $h(1)$ from the next is the input with $X(2)$ for the next step and so on. This way, it keeps remembering the context while training.

The formula for the current state is

$$h_t = f(h_{t-1}, x_t)$$

Applying Activation Function:

$$h_t = \tanh (W_{hh}h_{t-1} + W_{hx}x_t)$$

W is weight, h is the single hidden vector, W_{hh} is the weight at previous hidden state, W_{hx} is the weight at current input state, \tanh is the Activation function, that implements a Non-linearity that squashes the activations to the range $[-1, 1]$

Output:

$$y_t = W_{hy}h_t$$

Y_t is the output state. W_{hy} is the weight at the output state.

Advantages of Recurrent Neural Network

1. RNN can model sequence of data so that each sample can be assumed to be dependent on previous ones
2. Recurrent neural network are even used with convolutional layers to extend the effective pixel neighbourhood.

Disadvantages of Recurrent Neural Network

1. Gradient vanishing and exploding problems.
2. Training an RNN is a very difficult task.
3. It cannot process very long sequences if using \tanh or relu (The rectified linear activation function or ReLU for short is a piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero) as an activation function.

What is Long Short Term Memory (LSTM)?

Long Short-Term Memory (LSTM) networks are a modified version of recurrent neural networks, which makes it easier to remember past data in memory. The vanishing gradient problem of RNN is resolved here. LSTM is well-suited to classify, process and predict time series given time lags of unknown duration. It trains the model by using back-propagation. In an LSTM network, three gates are present: (Mittal,2019)

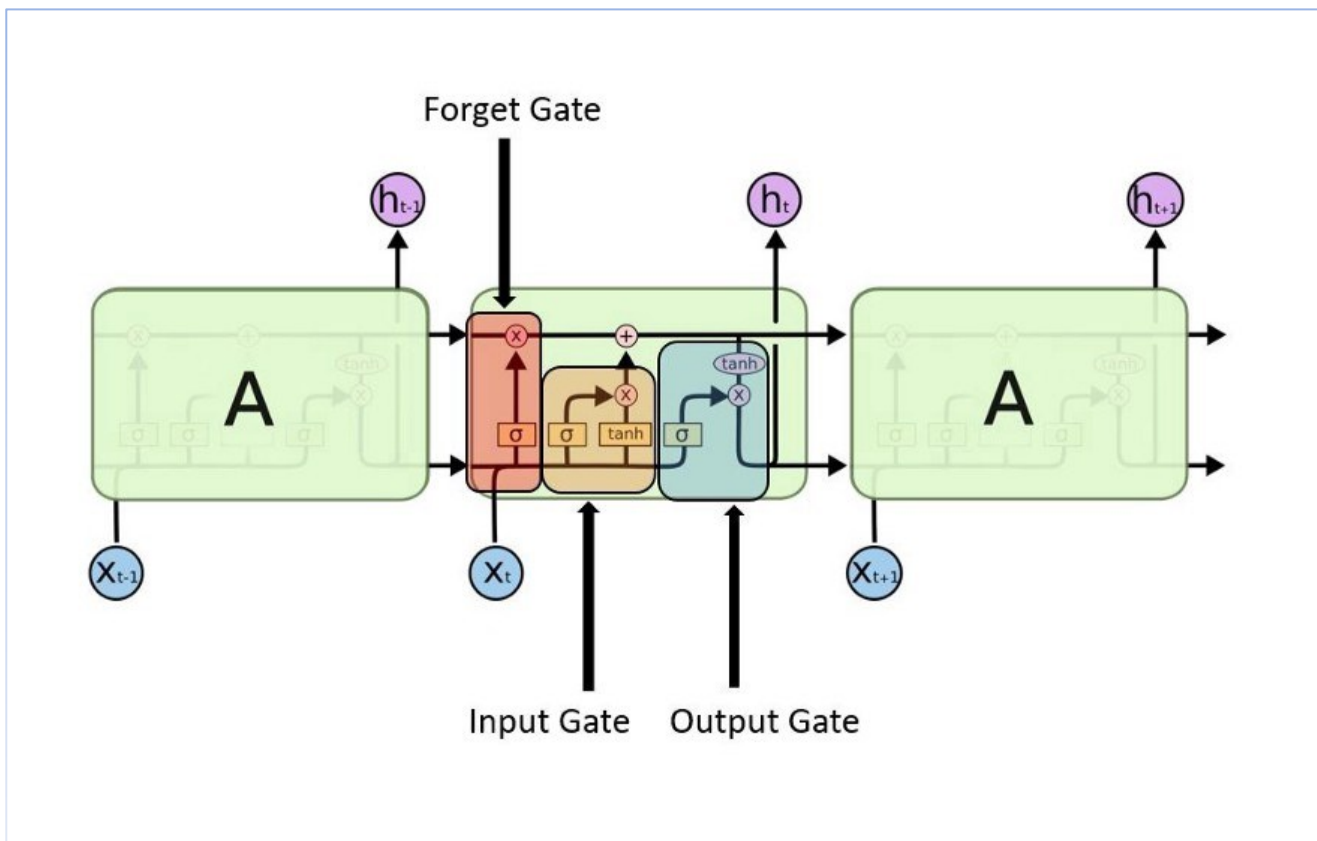


Figure 16: LSTM working illustration (Mittal,2019)

LSTM gates

1. **Input gate** — discover which value from input should be used to modify the memory. **Sigmoid** function decides which values to let through **0,1**. and **tanh** function gives weightage to the values which are passed deciding their level of importance ranging from -1 to 1.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

Input gate

2. Forget gate — discover what details to be discarded from the block. It is decided by the **sigmoid function**. it looks at the previous state(**ht-1**) and the content input(**Xt**) and outputs a number between **0(omit this)** and **1(keep this)** for each number in the cell state **Ct-1**.

Forget gate

$$f_t = \sigma (W_f \cdot [h_{t-1}, x_t] + b_f)$$

3. Output gate — the input and the memory of the block is used to decide the output. **Sigmoid** function decides which values to let through **0,1**. and **tanh** function gives weightage to the values which are passed deciding their level of importance ranging from **-1** to **1** and multiplied with output of **Sigmoid**.

$$o_t = \sigma (W_o [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh (C_t)$$

Analysis-

- We use Word2Vec algorithm for data preprocessing to apply the LSTM to get the embeddings vector.

Accuracy of the model on Testing Data using train test split is - 0.78971

Accuracy of the model on Testing Data using 10 fold cross validation is - 0.88726

3. CONCLUSION –

<i>Serial No.</i>	<i>Model used</i>	<i>10 Fold CV accuracy</i>	<i>Testing accuracy</i>
1	Linear Support Vector Classifier	0.78546	0.79041
2	Gaussian Naïve Bayes Classifier	0.69456	0.72230
3	Logistic Regression	0.78902	0.78817
4	Random Forest Classifier	0.74708	0.74251
5	LSTM(modified RNN)	0.88726	0.78971

From the above table we can see that LSTM model provides the best fit under 10 Fold Cross Validation with 88.72% accuracy followed by Logistic regression with 78.90% accuracy.

The lowest accuracy of classification of a text into sarcastic and not sarcastic is provided by Gaussian Naïve Bayes Classifier.

APPENDIX

CODE

1. Importing the required libraries and packages

```
import json
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
import nltk
nltk.download('stopwords')
from sklearn.preprocessing import LabelBinarizer
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
!pip install wordcloud
from wordcloud import WordCloud, STOPWORDS
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize, sent_tokenize
from bs4 import BeautifulSoup
import re, string, unicodedata
from keras.preprocessing import text, sequence
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.model_selection import train_test_split
from string import punctuation
import keras
from keras.models import Sequential
from keras.layers import Dense, Embedding, LSTM, Dropout, Bidirectional, GRU
import tensorflow as tf
import sklearn as sk
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.svm import LinearSVC
from sklearn.model_selection import cross_val_score, KFold
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
```

```
from nltk.stem.porter import PorterStemmer
```

2. Reading the dataset and dropping the link

```
data = pd.read_json("Sarcasm_Headlines_Dataset.json", lines = True)
data = data.drop('article_link',axis=1)
```

3. Exploratory data analysis and data preprocessing

- Checking for NA values

```
data.isna().sum()
```

- Checking for Balance in the dataset

```
sns.set_style("dark")
```

```
sns.countplot(data.is_sarcastic)
```

- Removing stop words , punctuations , html links and de-noising the dataset

```
stop = set(stopwords.words('english'))
punctuation = list(string.punctuation)
stop.update(punctuation)
def strip_html(text):
    soup = BeautifulSoup(text, "html.parser")
    return soup.get_text()

#Removing the square brackets
def remove_between_square_brackets(text):
    return re.sub('[^]*', '', text)
# Removing URL's
def remove_between_square_brackets(text):
    return re.sub(r'http\S+', '', text)
#Removing the stopwords from text
def remove_stopwords(text):
    final_text = []
    for i in text.split():
        if i.strip().lower() not in stop:
            final_text.append(i.strip())
    return " ".join(final_text)
#Removing the noisy text
def denoise_text(text):
    text = strip_html(text)
    text = remove_between_square_brackets(text)
```

```

text = remove_stopwords(text)
return text
#Apply function on review column
data['headline']=data['headline'].apply(denoise_text)

```

- WordCloud for text that is not sarcastic

```

plt.figure(figsize = (20,20)) # Text that is Not Sarcastic

wc = WordCloud(max_words = 2000 , width = 1600 , height = 800).generate("
".join(data[data.is_sarcastic == 0].headline))

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

```

- WordCloud for text that is sarcastic

```

plt.figure(figsize = (20,20)) # Text that is Sarcastic

wc = WordCloud(max_words = 2000 , width = 1600 , height = 800).generate("
".join(data[data.is_sarcastic == 1].headline))

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

```

- Characters in texts for sarcastic vs not sarcastic plot

```

fig,(ax1,ax2)=plt.subplots(1,2,figsize=(10,5),sharey=True)

text_len=data[data['is_sarcastic']==1]['headline'].str.len()

ax1.hist(text_len,color='red',range=(0,150),rwidth=0.9)

ax1.set_title('Sarcastic text')

text_len=data[data['is_sarcastic']==0]['headline'].str.len()

ax2.hist(text_len,color='green',range=(0,150),rwidth=0.9)

ax2.set_title('Not Sarcastic text')

fig.suptitle('Characters in texts')

plt.show()

```

- Words in texts for sarcastic vs not sarcastic plot

```

fig,(ax1,ax2)=plt.subplots(1,2,figsize=(10,5),sharey=True)

text_len=data[data['is_sarcastic']==1]['headline'].str.split().map(lambda x: len(x))

ax1.hist(text_len,color='red',range=(0,20),rwidth=0.9)

```

```

ax1.set_title('Sarcastic text')

text_len=data[data['is_sarcastic']==0]['headline'].str.split().map(lambda x: len(x))

ax2.hist(text_len,color='green',range=(0,25),rwidth=0.9)

ax2.set_title('Not Sarcastic text')

fig.suptitle('Words in texts')

plt.show()

```

- Average word length in each text for sarcastic vs not sarcastic

```

fig,(ax1,ax2)=plt.subplots(1,2,figsize=(20,10))

word=data[data['is_sarcastic']==1]['headline'].str.split().apply(lambda x : [len(i)
for i in x])

sns.distplot(word.map(lambda x: np.mean(x)),ax=ax1,color='red')

ax1.set_title('Sarcastic text')

word=data[data['is_sarcastic']==0]['headline'].str.split().apply(lambda x : [len(i)
for i in x])

sns.distplot(word.map(lambda x: np.mean(x)),ax=ax2,color='green')

ax2.set_title('Not Sarcastic text')

fig.suptitle('Average word length in each text')

```

- Replacing special symbols and digits in headline column

```
data['headline'] = data['headline'].apply(lambda s : re.sub('[^a-zA-Z]', ' ', s))
```

- Getting features and labels

```

features = data['headline']

labels = data['is_sarcastic']

```

- Stemming

```

ps = PorterStemmer()

features = features.apply(lambda x: x.split())

features = features.apply(lambda x : ' '.join([ps.stem(word) for word in x]))

```


○ Lemmatization

#Lemmatizing our data

```
nltk.download('wordnet')
```

```
from nltk.stem import WordNetLemmatizer
```

Create WordNetLemmatizer object

```
wnl = WordNetLemmatizer()
```

```
features= features.apply(lambda x: x.split())
```

```
features= features.apply(lambda x : ' '.join([wnl.lemmatize(word) for word in x]))
```

○ TF-IDF Vectorizer

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
tv = TfidfVectorizer(max_features = 5000)
```

```
features = list(features)
```

```
features = tv.fit_transform(features).toarray()
```

○ Train-Test split and 10 fold cross validation

```
features_train, features_test, labels_train, labels_test = train_test_split(features,  
labels, test_size = .05, random_state = 0)
```

○ Word2Vec for Recurrent Neural Network

```
words = []
```

```
for i in data.headline.values:
```

```
    words.append(i.split())
```

```
words[:5]
```

```
import gensim
```

#Dimension of vectors we are generating

```
EMBEDDING_DIM = 200
```

#Creating Word Vectors by Word2Vec Method

```

w2v_model = gensim.models.Word2Vec(sentences = words ,
size=EMBEDDING_DIM , window = 5 , min_count = 1)

#vocab size

len(w2v_model.wv.vocab)

#We have now represented each of 38071 words by a 100dim vector

tokenizer = text.Tokenizer(num_words=35000)

tokenizer.fit_on_texts(words)

tokenized_train = tokenizer.texts_to_sequences(words)

x = sequence.pad_sequences(tokenized_train, maxlen = 20)

# Adding 1 because of reserved 0 index

# Embedding Layer creates one more vector for "UNKNOWN" words, or padded words (0s). This Vector is filled with zeros.

# Thus our vocab size increases by 1

vocab_size = len(tokenizer.word_index) +1


# Function to create weight matrix from word2vec gensim model

def get_weight_matrix(model, vocab):

    # total vocabulary size plus 0 for unknown words

    vocab_size = len(vocab) + 1

    # define weight matrix dimensions with all 0

    weight_matrix = np.zeros((vocab_size, EMBEDDING_DIM))

    # step vocab, store vectors using the Tokenizer's integer mapping

    for word, i in vocab.items():

        weight_matrix[i] = model[word]

    return weight_matrix

#Getting embedding vectors from word2vec and usings it as weights of non-trainable keras embedding layer

embedding_vectors = get_weight_matrix(w2v_model, tokenizer.word_index)

```

4. Model Fitting

○ Linear Support Vector Classifier

```
lsvc = LinearSVC()
# training the model
lsvc.fit(features_train, labels_train)
score=cross_val_score(lsvc,features,labels,cv=kf)
print("Cross Validation Scores are {}".format(score))
print("Average Cross Validation score :{}".format(score.mean()))
# getting the score of train and test data
print(lsvc.score(features_train, labels_train))
print(lsvc.score(features_test, labels_test))
```

○ Gaussian Naïve Bayes

```
gnb = GaussianNB()
gnb.fit(features_train, labels_train)
score1=cross_val_score(gnb,features,labels,cv=kf)
print("Cross Validation Scores are {}".format(score1))
print("Average Cross Validation score :{}".format(score1.mean()))
print(gnb.score(features_train, labels_train))
print(gnb.score(features_test, labels_test))
```

○ Logistic Regression

```
lr = LogisticRegression()
lr.fit(features_train, labels_train)
score2=cross_val_score(lr,features,labels,cv=kf)
print("Cross Validation Scores are {}".format(score2))
print("Average Cross Validation score :{}".format(score2.mean()))
print(lr.score(features_train, labels_train))
print(lr.score(features_test, labels_test))
```

○ Random Forest Classifier

```
rfc = RandomForestClassifier(n_estimators = 10, random_state = 0)
rfc.fit(features_train, labels_train)
score3=cross_val_score(rfc,features,labels,cv=kf)
print("Cross Validation Scores are {}".format(score3))
```

```
print("Average Cross Validation score :{}".format(score3.mean()))
print(rfc.score(features_train, labels_train))
print(rfc.score(features_test, labels_test))
```

- LSTM using Word2Vec

#Defining Neural Network

```
model = Sequential()
```

#Non-trainable embedding layer

```
model.add(Embedding(vocab_size, output_dim=EMBEDDING_DIM,
weights=[embedding_vectors], input_length=20, trainable=True))
```

#LSTM

```
model.add(Bidirectional(LSTM(units=128 , recurrent_dropout = 0.3 , dropout =
0.3,return_sequences = True)))
```

```
model.add(Bidirectional(GRU(units=32 , recurrent_dropout = 0.1 , dropout = 0.1)))
```

```
model.add(Dense(1, activation='sigmoid'))
```

```
model.compile(optimizer=tf.keras.optimizers.Adam(lr=0.01),
loss='binary_crossentropy', metrics=['acc'])
```

```
del embedding_vectors
```

```
model.summary()
```

#For train test split

```
x_train, x_test, y_train, y_test = train_test_split(x, data.is_sarcastic , test_size = 0.3 ,
random_state = 0)
```

```
history = model.fit(x_train, y_train, batch_size = 128 , validation_data = (x_test,y_test) ,
epochs = 3)
```

```
print("Accuracy of the model on Training Data is - " ,
model.evaluate(x_train,y_train)[1]*100)
```

```
print("Accuracy of the model on Testing Data is - " ,
model.evaluate(x_test,y_test)[1]*100)
```

#For 10 fold cross validation

```
kf=KFold(10)
```

```
fold=0
```

```
y=data.is_sarcastic
```

```
for train, test in kf.split(x,y):
```

```
    fold+=1
```

```
    x_train=x[train]
```

```
    y_train=y[train]
```

```
    x_test=x[test]
```

```
    y_test=y[test]
```

```
history = model.fit(x_train, y_train, batch_size = 128 , validation_data = (x_test,y_test) ,  
epochs = 3)
```

```
print("Accuracy of the model on Training Data for 10 fold CV is - " ,  
model.evaluate(x_train,y_train)[1]*100)
```

```
print("Accuracy of the model on Testing Data for 10 fold CV is - " ,  
model.evaluate(x_test,y_test)[1]*100)
```

REFERENCES

- James G., Witten D., Hastie T. and Tibshirani R.(2013) *An Introduction to Statistical Learning*
- Sindhu C., Vadivu G. and Rao V. M. (2018).*A comprehensive study on sarcasm detection techniques in sentiment analysis*, International Journal of Pure and Applied Mathematics Volume 118, No. 22, 433-442 - <https://acadpubl.eu/hub/2018-118-22/articles/22a/63.pdf>
- Tungthamthiti P., Shirai K and Mohd M.(2014).*Recognition of Sarcasm in Tweets Based on Concept Level Sentiment Analysis and Supervised Learning Approaches* - PACLIC 28 - <https://www.aclweb.org/anthology/Y14-1047.pdf>
- Karani Dhruvil, (2018). *Introduction to Word Embeddings and Word2Vec* – Towards Data Science – <https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa>
- Mittal Aditi, (2019). *Understanding RNN and LSTM* – Medium – <https://aditi-mittal.medium.com/understanding-rnn-and-lstm-f7cdf6dfc14e>
- Allen James. (2002) *Natural Language Understanding*.

END
