Class: TE-1 **Batch:** L1

Submitted By:

- Harsh Dhawale 31124
- Vinayak Jamadar 31137
- Sanket Jhavar 31138

Title: Twitter Sentiment Analysis System

Problem Statement: To Build a Twitter Sentiment Analysis System

Date: 14-5-23

Objectives:

- Twitter Sentiment Analysis Dataset
- Text Processing
  A. Cleaning of raw text
  B. Tokenization
  C. Stemming
- Word Embedding Techniques
  A. Bag of Words
  B. Term Frequency — Inverse Document Frequency
  C. Word2Vec
- Model
- Performance Metrics
- Results
- Summary

Theory:

Twitter Sentiment Analysis Dataset

Let’s start with our Twitter data. We will use the open-source Twitter Tweets Data for Sentiment Analysis dataset. It contains 163000 tweets.

The target variable for this dataset is ‘**category’**, which maps negative tweets to -1, positive tweets to 1 and anything else to 0. Think of the target variable as what you’re trying to predict. For our machine learning problem, we'll train a classification model on this data so it can predict the class of any new tweets we give it.

Text Processing

Data usually comes from a variety of different sources and is often in a variety of different formats. For this reason, cleaning your raw data is an essential part of preparing your dataset. However, cleaning is not a simple process, as text data often contain redundant and/or repetitive words. This is especially true in Twitter sentiment analysis, so processing our text data is the first step towards our solution.

The fundamental steps involved in text processing are:

A. Cleaning of Raw Data
B. Tokenization
C. Stemming

***A.* Cleaning of Raw Data**

This phase involves the deletion of words or characters that do not add value to the meaning of the text. Some of the standard cleaning steps are below:

- Lowering case
- Removal of mentions
- Removal of special characters
- Removal of stopwords
- Removal of hyperlinks
- Removal of numbers
- Removal of whitespaces

Lowering Case

Lowering the case of text is essential for the following reasons:
The words, ‘Tweet’, ‘TWEET’, and ‘tweet’ all add the same value to a sentence.
Lowering the case of all the words helps to reduce the dimensions by decreasing the size of the vocabulary.

```
def to_lower(word):
    result = word.lower()
    return result
```

Removal of mentions

Mentions are very common in tweets. However, as they don’t add value for interpreting the sentiment of a tweet, we can remove them. Mentions always come in the form of ‘@mention’, so we can remove strings that start with ‘@’.

To achieve this on the entire dataset, we use the function below.

```
def remove_mentions(word):
   result = re.sub(r"@\S+", "", word)
   return result
```

Removal of stopwords

Stopwords are commonly occurring words in a language, such as ‘the’, ‘a’, ‘an’, ‘is’ etc. We can remove them here because they won’t provide any valuable information for our Twitter data analysis.

```
#split the sentence and check if stopwords present in it, if not
 stop word then append it to another temp list till.
#And clear the temp list after use
def remove_stopwords(text):
    temp = []
    for i in text.split():
        if i not in stopwords.words('english'):
            temp.append(i)
    result = temp[:]
    temp.clear()
    return result
```

Removal of hyperlinks

Now we can remove URLs from the data. It’s not uncommon for tweets to contain URLs, but we won’t need to analyze them for our task.

```
def remove_hyperlink(word):
   return re.sub(r"http\S+", "", word)
```

***B.* Tokenization**

Tokenization is the process of splitting text into smaller chunks, called tokens. Each token is an input to the machine learning algorithm as a feature. NLTK (Natural Language Toolkit) provides a utility function for tokenizing data.

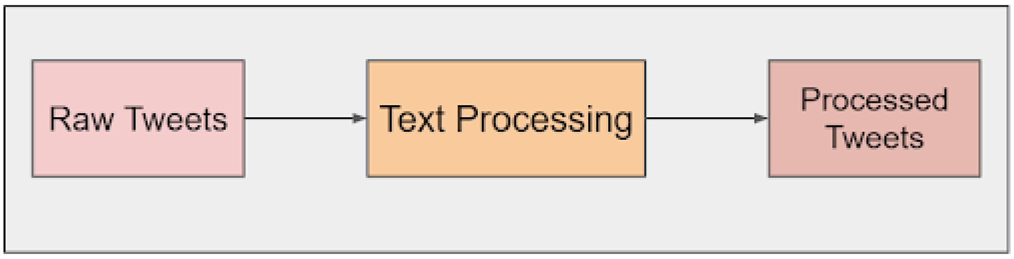
```
from nltk.tokenize import word_tokenize
tweets_data['tweet'] = tweets_data['tweet'].apply(word_tokenize)
```

***C.* Stemming**

Stemming is the process of removing and replacing suffixes from a token to obtain the root or base form of the word. This is called a ‘stem’. For example, the stem for the words, ‘satisfied’, ‘satisfaction’, and ‘satisfying’ is ‘satisfy’ and all of these imply the same feeling.

Porter stemmer is a widely used stemming technique. nltk.stem provides the utility function to stem ‘PorterStemmer’

```
#the words like "asked", "asking", "ask" will be stem to "ask" o
nly
def stem_words(text):
    temp = []
    for i in text:
        temp.append(ps.stem(i))
    result = temp[:]
    temp.clear()
    return result
```



Overview of Text Processing. (Image by author)

Word Embedding Techniques

There is a huge amount of data in text format. Analyzing text data is an extremely complex task for a machine as it's difficult for a machine to understand the semantics behind the text. At this stage, we're going to process our text data into a machine-understandable format using word embedding.

> *Word Embedding is simply converting data in a text format to numerical values (or vectors) so we can give these vectors as input to a machine, and analyze the data using the concepts of algebra.*

However, it's important to note that when we perform this transformation there could be data loss. The key then is to maintain an equilibrium between conversion and retaining data.

Here are two commonly used terminologies when it comes to this step.

- Each text data point is called a Document
- An entire set of documents is called a Corpus

Text processing can be done using the following techniques:

1. Bag of Words
2. TF-IDF
3. Word2Vec

Next, let’s explore each of the above techniques in more detail, then decide which to use for our Twitter sentiment analysis model.

A. Bag of Words

Bag of Words does a simple transformation of the document to a vector by using a dictionary of unique words. This is done in just two steps, outlined below.

Construction of Dictionary

Create a dictionary of all the unique words in the data corpus in a vector form. Let the number of unique words in the corpus be, ‘d’. So each word is a dimension and hence this dictionary vector is a d-dimension vector.

Construction of Vectors

For each document, say, r_i we create a vector, say, v_i.

This v_i which has d-dimensions can be constructed in two ways:

1. For each document, the v_i is constructed in accordance with the dictionary such that each word in the dictionary is reproduced by the number of times that word is present in the document.
2. For each document, the v_i is constructed in accordance with the dictionary such that each word in the dictionary is reproduced as:

- 1 if the word exists in the document or
- 0 if the word doesn’t exist in the document

This type is known as a Binary Bag of Words.

Now we have vectors for each document and a dictionary with a set of unique words from the data corpus. These vectors can be analyzed by,

- plotting in d-dimension space or

- calculating distance between vectors to get the similarity (*the closer the vectors are, the more similar they are*)

B. Term Frequency — Inverse Document Frequency

There are three elements here: word, document, corpus. Term Frequency — Inverse Document Frequency, or TF-IDF for short, uses the relationship between these elements to convert text data into vectors.

> *Term Frequency refers to the relationship between a word and a document. Whereas Inverse Document Frequency refers to the relationship between a word and the corpus.*

Calculating Term Frequency

Term frequency is the probability of the word w_j in the document r_i. It is calculated as below.

$$Term\ Frequency = \frac{Number\ of\ times\ a\ word\ is\ present\ in\ the\ review}{Total\ number\ of\ words\ in\ that\ review}$$

The mathematical formula for calculating TF. (Image by author)

> *High Term Frequency of a word in a review implies the word is frequent in that review. Low Term Frequency of a word in a review implies the word is rare in that review.*

Calculating Inverse Document Frequency

Inverse Document Frequency (IDF) says how frequently a word occurs in the entire corpus. This is calculated as below.

$$Inverse\ Document\ Frequency = Log(\frac{Total\ number\ of\ documents\ in\ the\ corpus}{Number\ of\ documents\ which\ contain\ that\ word})$$

The mathematical formula for calculating IDF. (Image by author)

> *Low Inverse Document Frequency implies the word is frequent in the corpus. High Inverse Document Frequency implies the word is rare in the corpus.*

We use logarithm instead of simple inverse ratio because of scaling. Term Frequency is a probability and ranges between 0 and 1. The inverse ratio of this can take values from 0 to infinity and can bias the IDF. Using log to solve this problem is one simple and highly accepted reason.

*TF-IDF of a word in the review = TF(word, review) * IDF(word, corpus).*

In the vector form of each document, we have this TF-IDF of each word. Converting a document into a vector, using TF-IDF values is called TF-IDF vectorization.

TF-IDF vectorization gives high importance to words which are:

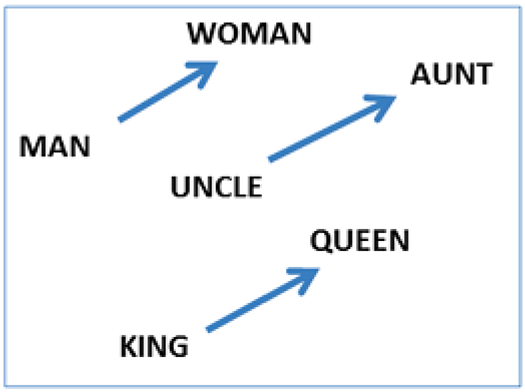
- frequent in a document (from TF)
- rare in the corpus (from IDF)

C. Word2Vec

In Bag of Words and TF-IDF, we convert *sentences into vectors*. But in Word2Vec, we convert *words into vectors*. Hence the name, word2vec!

Word2Vec takes as its input a large corpus of text and produces a vector space, typically of several hundred dimensions, with each unique word in the corpus assigned to a corresponding vector in the space. The positioning of word vectors is done in such a way that words with common contexts in the corpus are located closer in space.

For example, the vector from man to woman is parallel to the king to the queen, etc.



Example of vectors and their representations using Word2vec. (Image by author)

Model Fitting

Here we have used Naïve Bayes Classifier method.

Naive Bayes classifier is a popular machine learning algorithm used for classification tasks. It is based on Bayes' theorem, which is a probability theory that relates the probability of a hypothesis to the evidence. The algorithm is called "naive" because it makes the assumption that all features are independent of each other, which is often not true in real-world scenarios.
The Naive Bayes classifier is widely used for text classification, spam filtering, sentiment analysis, and many other applications. The algorithm is based on the Bayes' theorem, which can be written as:
$P(A|B) = P(B|A) * P(A) / P(B)$
where A and B are events, $P(A|B)$ is the probability of event A given that event B has occurred, $P(B|A)$ is the probability of event B given that event A has occurred, $P(A)$ is the prior probability of event A, and $P(B)$ is the prior probability of event B.

```
#import Naive bases major classes to check the accuracy
from sklearn.naive_bayes import GaussianNB, MultinomialNB, Ber
noulliNB
```

```
#Create object of each class
gObj = GaussianNB()
mObj = MultinomialNB()
bObj = BernoulliNB()
```

1. *Gaussian Naive Bayes Classifier:*

Suppose we have a dataset of patients with diabetes and non-diabetes conditions, and we want to classify a new patient based on their blood glucose and blood pressure levels. The data for each patient consists of two continuous features: blood glucose level and blood pressure level. We can use a Gaussian Naive Bayes classifier to classify the new patient based on their blood glucose and blood pressure levels.

2. *Multinomial Naive Bayes Classifier:*

Suppose we want to classify news articles into categories such as politics, sports, and entertainment. We can represent each news article as a bag of words, where each feature represents the frequency of a particular word in the article. We can then use a Multinomial Naive Bayes classifier to classify the news article based on its bag-of-words representation.

3. *Bernoulli Naive Bayes Classifier:*

Suppose we want to classify emails as spam or not spam. We can represent each email as a bag of words, where each feature represents whether a particular word appears in the email or not. We can then use a Bernoulli Naive Bayes classifier to classify the email based on its bag-of-words representation.

System Architecture:

The system architecture for Twitter sentiment analysis typically involves several components that work together to analyze and classify the sentiment of tweets. Here's a high-level overview of a typical architecture:

Data Collection: The first step is to collect the relevant Twitter data. This can be done by utilizing the Twitter API or by accessing publicly available datasets. The data collection process involves gathering tweets based on specific keywords, user profiles, or other criteria.

Preprocessing: Once the data is collected, it needs to be preprocessed to remove noise and irrelevant information. Preprocessing steps may include tokenization (splitting text into words or tokens), removing stop words (commonly occurring words like "and," "the," etc.), stemming or lemmatization (reducing words to their base form), and handling mentions, hashtags, URLs, and other special characters.

Feature Extraction: In this step, relevant features are extracted from the preprocessed text to represent the tweets. Commonly used features include bag-of-words or TF-IDF (Term Frequency-Inverse Document Frequency) representations, n-grams (contiguous sequences of words), word embeddings (vector representations of words), and sentiment-specific features like emoticons or capitalization patterns.

Sentiment Classification: The extracted features are then used as input to a sentiment classification model. This model can be based on various algorithms, such as Naive Bayes, Support Vector Machines (SVM), Random Forests, or more advanced techniques like deep learning models (e.g., recurrent neural networks or transformers). The model is trained on labeled data, where each tweet is associated with a sentiment label (e.g., positive, negative, or neutral).

Model Training: The sentiment classification model is trained using labeled data. The labeled data is typically annotated manually, where human annotators assign sentiment labels to a subset of tweets. The model learns from this labeled data to generalize patterns and make predictions on new, unseen tweets.

Model Evaluation: The trained sentiment classification model is evaluated using evaluation metrics such as accuracy, precision, recall, and F1 score. This step helps assess the model's performance and determine if further improvements or fine-tuning are needed.

Real-Time Prediction: Once the sentiment classification model is trained and evaluated, it can be deployed to make real-time predictions on new, incoming tweets. The deployed model takes preprocessed tweet data as input, applies the same preprocessing steps as during training, and predicts the sentiment label (positive, negative, or neutral) for each tweet.

Visualization and Analysis: The final step involves visualizing and analyzing the results. The sentiment analysis system can generate reports, visualizations, or dashboards to present the sentiment distribution, trending topics, or other relevant insights derived from the analyzed tweets.

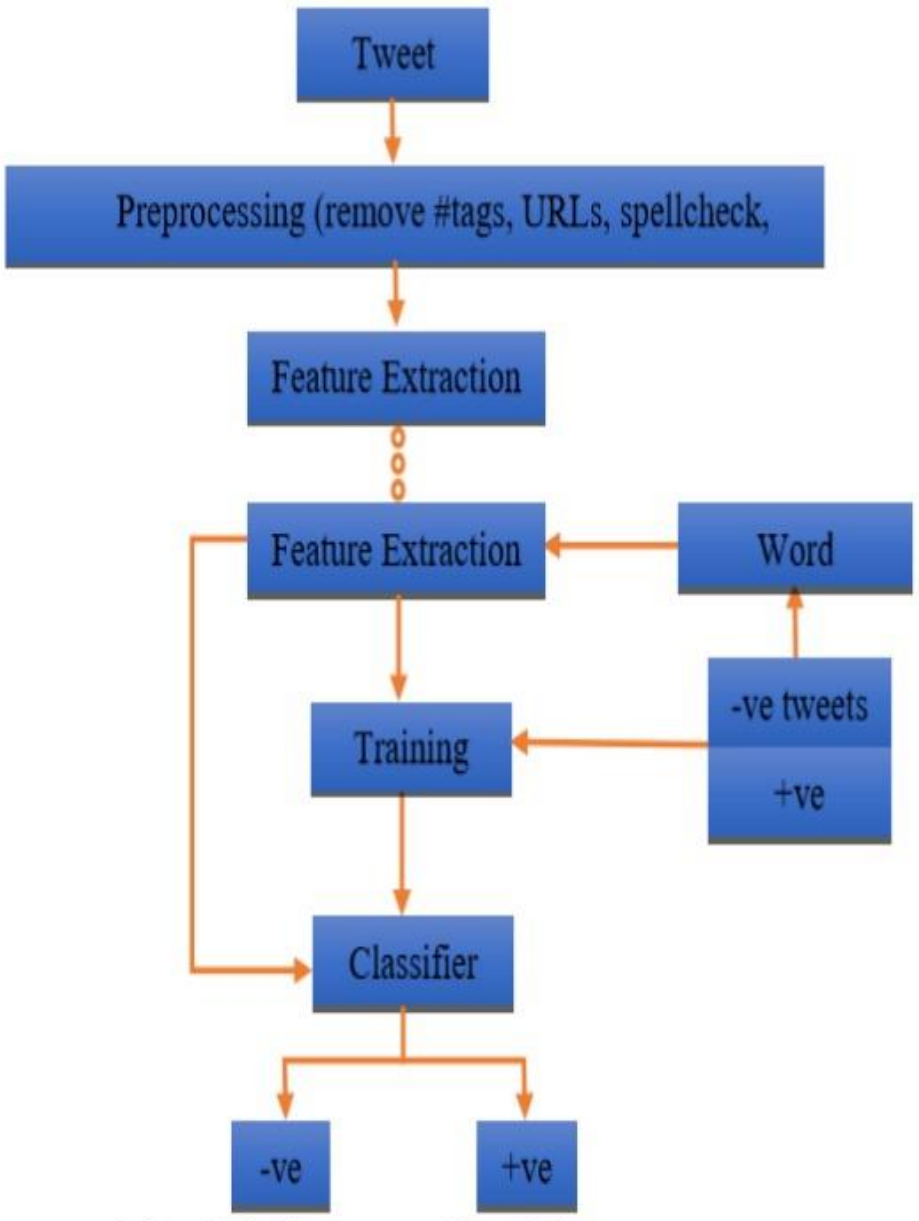


Fig 1.1 System Architecture

Methodology:

Introduction: Start by providing an introduction to the purpose and scope of the sentiment analysis report. Explain why sentiment analysis is relevant for understanding public opinion on Twitter and outline the objectives of the analysis.

Data Collection: Describe the data collection process, including the timeframe, sample size, and any specific criteria used to gather the tweets. Mention if you used the Twitter API, publicly available datasets, or a combination of both.

Preprocessing: Explain the preprocessing steps applied to the collected tweets. Describe the techniques used for tokenization, stop word removal, stemming/lemmatization, and handling special characters like mentions, hashtags, and URLs.

Feature Extraction: Discuss the features extracted from the preprocessed tweets. Explain the methods used, such as bag-of-words, TF-IDF, n-grams, word embeddings, and sentiment-specific features. Highlight any specific considerations or modifications made to suit the sentiment analysis task.

Sentiment Classification Model: Present the sentiment classification model used for predicting the sentiment of the tweets. Explain the algorithm employed (e.g., Naive Bayes, SVM, Random Forest, deep learning models), and provide justification for the choice based on factors like performance, scalability, and interpretability.

Model Training and Evaluation: Describe the process of training the sentiment classification model using labeled data. Explain the annotation process and any considerations taken to ensure the quality and reliability of the labels. Present the evaluation metrics used (e.g., accuracy, precision, recall, F1 score) and provide the performance results of the trained model.

Results and Analysis: Present the results of the sentiment analysis. Provide an overview of the sentiment distribution, indicating the percentage of positive, negative, and neutral tweets. Include visualizations such as pie charts or bar graphs to illustrate the sentiment breakdown. Identify any notable trends or patterns observed in the data, such as popular topics associated with specific sentiments.

Insights and Interpretation: Provide insights and interpretation of the sentiment analysis results. Discuss the implications of the sentiment distribution and trends observed on Twitter. Connect the sentiment analysis findings to the broader context of the analyzed topic or domain, highlighting potential implications for businesses, public opinion, or decision-making processes.

Limitations and Future Work: Discuss the limitations of the sentiment analysis approach used and acknowledge any potential biases or shortcomings. Highlight areas for future improvement or research to enhance the accuracy or scope of the sentiment analysis. Address any constraints encountered during the analysis, such as data availability, computational resources, or time limitations.

Conclusion: Summarize the key findings of the sentiment analysis and reiterate the main points discussed throughout the report. Provide a concise conclusion that addresses the initial objectives of the analysis.

References: Include a list of references for any external sources, research papers, or frameworks used during the sentiment analysis process.

Results:

```
#accuracy score for Gaussian
print("Gaussian", accuracy_score(y_test,y_pred_gauss))
```

```
Gaussian 0.6777825500061357
```

```
#accuracy score for Multinomial
print("Multinomial", accuracy_score(y_test,y_pred_multinom))
```

```
Multinomial 0.6789483372192907
```

```
#accuracy score for BernoulliNB
print("BernoulliNB", accuracy_score(y_test,y_pred_berno))
```

```
BernoulliNB 0.695668180144803
```

```
#Crosschek - Output for positive
data = {
  "text": ["Discussion was very good"]
}
```

```
dfx = pd.DataFrame(data)
Xcheck = cv.transform(dfx['text']).toarray()
print(bObj.predict(Xcheck)[0])
print(mObj.predict(Xcheck)[0])
print(gObj.predict(Xcheck)[0])

1
1
1

#Crosschek - Output for Negative text
#Result : it works
data = {
  "text": ["The analysis is bad"]
}
dfx = pd.DataFrame(data)
Xcheck = cv.transform(dfx['text']).toarray()
print(bObj.predict(Xcheck)[0])
print(mObj.predict(Xcheck)[0])
print(gObj.predict(Xcheck)[0])

-1
-1
-1

#Crosschek - Output for neutral
#Result : Multinomial fails
data = {
  "text": ["we are coming"]
}
dfx = pd.DataFrame(data)
Xcheck = cv.transform(dfx['text']).toarray()
print(bObj.predict(Xcheck)[0])
print(mObj.predict(Xcheck)[0])
print(gObj.predict(Xcheck)[0])

0
1
0


#Crosschek - Output for neutral
#Result : None worked
data = {
  "text": ["The analysis is not bad nor good"]
}
dfx = pd.DataFrame(data)
Xcheck = cv.transform(dfx['text']).toarray()
print(bObj.predict(Xcheck)[0])
print(mObj.predict(Xcheck)[0])
print(gObj.predict(Xcheck)[0])
```

```
1
1
-1
```

Also, keep in mind that these results are based on our training data. When applying a sentiment analysis model to real-world data, we still have to actively monitor the model's performance over time.

Conclusion:

In this Mini Project, we learned various text processing and word embedding techniques and implemented a Twitter sentiment analysis classification model on processed data. Hopefully, this will give you an idea of how these social media data analytics systems work, and the sort of work required to prepare and deploy them.