# FOM HOCHSCHULE

**Essen , Germany – 45127**

**2019-2021**


**A PROJECT REPORT**
**ON**

## "Sentiment Analyzer (Using IMDB movie review)"

**MASTER OF SCIENCE**

**IN**

**BIG DATA AND BUSINESS ANALYTICS**

Submitted By
**SHIVANEE SURESHBHAI PANCHAL**
ROLL NO : **DBSOM19BD04029**

Under the Guidance of
**Prof. Srinatha D.K.**

# FOM HOCHSCHULE
## Essen, Germany - 45127


### DEPARTMENT OF IT MANAGEMENT

## BIG DATA AND BUSINESS ANALYTICS

### CERTIFICATE

This is certify that the project work titled **"Sentiment Analyzer (Using IMDB movie review)"** carried out by **Mrs. Shivanee Sureshbhai Panchal (Roll no : DBSOM19BD04029)** in partial fulfillment of the requirements for the award of **Master of Science** in **Big Data and Business Analytics,** FOM Hochschule , Essen during the year 2020-2021.It is certified that all corrections/suggestions indicated for internal assessment have been incorporated in the report. The project report has been approved as it satisfies the academic requirements in respect of project work prescribed for the said degree.

***Signature of the guide***

**(Prof. SHRINATHA D. K.)**

# PREFACE

It gives me immense pleasure to write this report on Analysis of Semi- & Unstructured Data subject by addressing use of Semi and Unstructured Data in Sentiment Analysis. The term is evolved with the reviews and opinion of different people on different items, movies, products, etc. and its importance is growing exponentially in this data world. The report contains a detailed explanation and execution on a IMDB movie review and its Analytics.

# ABSTRACT

Sentiment analysis also term as refers to the use of natural language processing, text analysis and computational linguistics to identify and extract subjective information in source materials. In recent years, Opinion mining is a hotspot in the field of natural language processing, and it is also a challenging problem Sentiment analysis is widely applied to reviews and social media for a variety of applications, ranging from marketing to customer service. Movie reviews are an important way to gauge the performance of a movie. The objective of this project is to extract features from the product reviews and classify reviews into positive, negative. . In this project we aim to use Sentiment Analysis on a set of movie reviews which is given by reviewers and then try to understand what the overall reaction to the movie was according to them, i.e. if they liked the movie or they hated it. We aim to use the relationships of the words in the review to predict the overall polarity of the review.

**Keywords**: Movie Review Mining, Pre-processing, Sentiment Analysis, Stemming

# TABLE OF CONTENT

# TABLE OF FIGURES

# SYSTEM REQUIREMENTS

- Hardware Requirements:

⇒ Core i5/i7 processor
⇒ At least 8 GB RAM
⇒ At least 60 GB of Usable Hard Disk Space

- Software Requirements:

⇒ Python 3.x
⇒ Anaconda Distribution
⇒ NLTK Toolkit
⇒ UNIX/LINUX/MAC Operating System.

- **Dataset: IMDB review dataset**:

⇒ I am using a publicly available dataset consisting of 50,000 reviews from IMDB, allowing no more than 30 reviews per movie. The constructed dataset contains an equal number of positive and negative reviews, so random guessing yields 50% accuracy.

⇒ You can download the dataset from the following link:

⇒ http://ai.stanford.edu/~amaas/data/sentiment/
⇒ The dataset has three columns. The first column specifies a sequential number of the record. The second column has a text which is the review and the third column is a class denoting the sentiment of the reviewer. The class has value 1 if the sentiment is positive and 0 if the sentiment is negative.
⇒ 'Here Objective is to build a custom sentiment analyser that can classify sentiment (positive or negative) of reviewers out of their movie reviews.'

# ARCHITECTURE

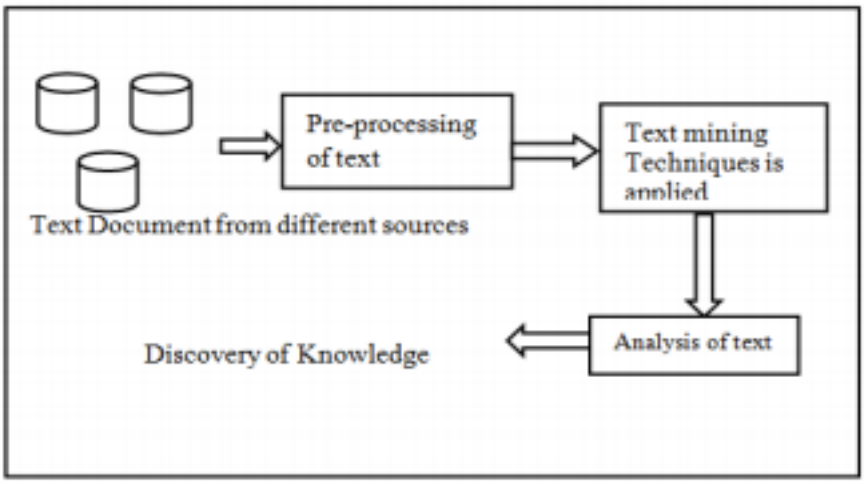


**Figure 1.** Text Mining Process

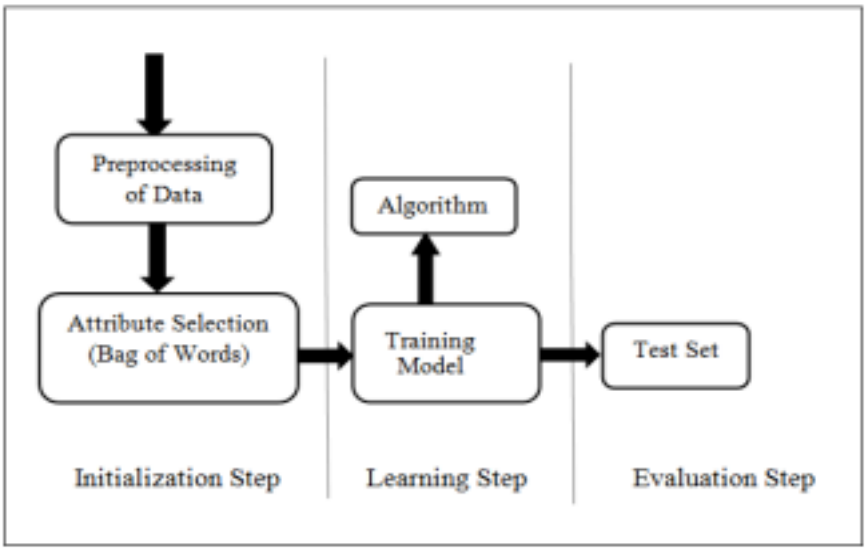


**Fig. 2.** Steps for training a classifier for sentiment analysis

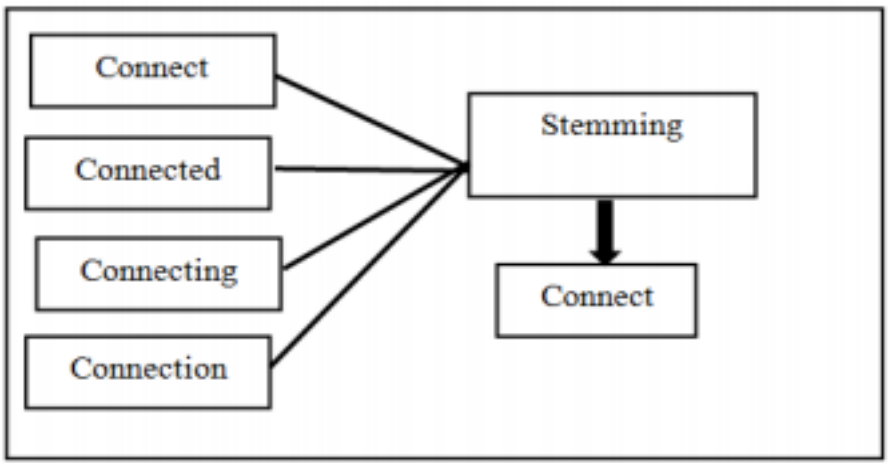


**Fig 3** – Stemming Process

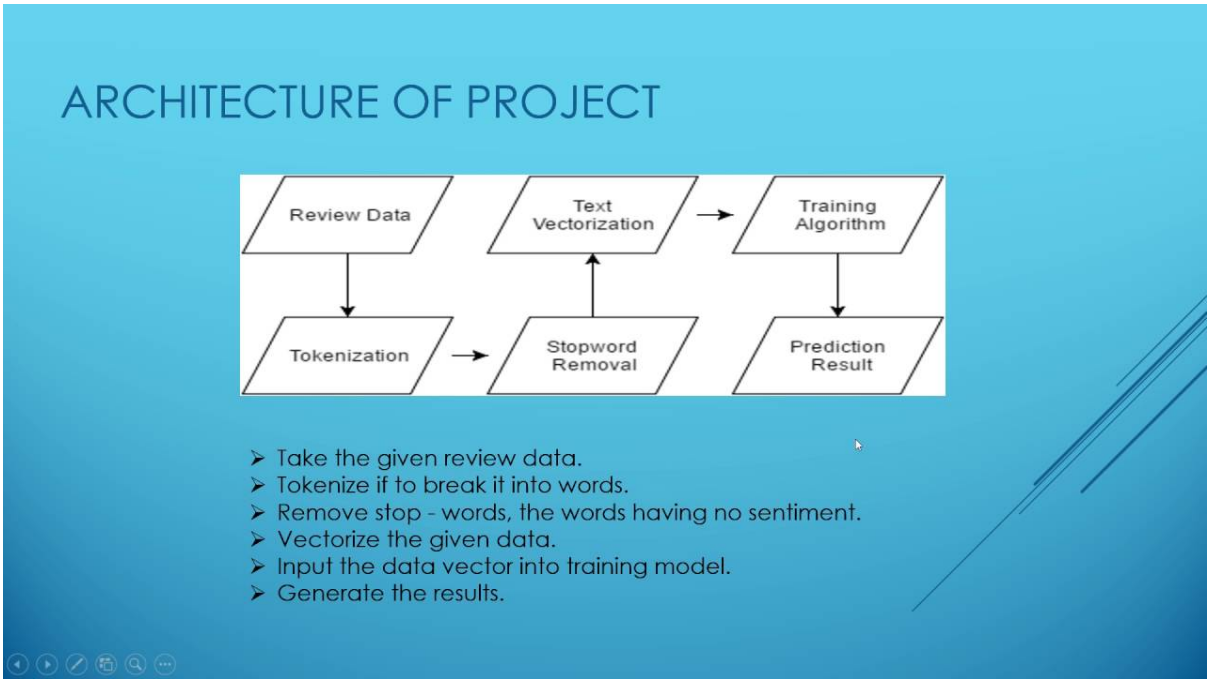


Figure 4. Architecture of Project

# ALGORITHM AND FORMULATION

- **Step 1:** Download and Combine Movie Reviews

If you haven't yet, go to IMDb Reviews and click on "Large Movie Review Dataset v1.0". Once that is complete you'll have a file called aclImdb_v1.tar.gz in your downloads folder.

- **Step 2**: Read into Python

- **Step 3:** Clean and Pre-process

The raw text is pretty messy for these reviews so before we can do any analytics we need to clean things up.

- **Step 4:** Vectorization

In order for this data to make sense to our machine learning algorithm we'll need to convert each review to a numeric representation, which we call *vectorization*.

1. Sentiment sentence extraction & pos tagging:

Tokenization of reviews after removal of STOP words which mean nothing related to sentiment is the basic requirement for POS tagging. After proper removal of STOP words like "am, is, are, the, but" and so on the remaining sentences are converted in tokens. These tokens take part in POS tagging

In natural language processing, part-of-speech (POS) taggers have been developed to classify words based on their parts of speech. For sentiment analysis, a POS tagger is very useful because of the following two reasons: 1) Words like nouns and pronouns usually do not contain any sentiment. It is able to filter out such words with the help of a POS tagger; 2) A POS tagger can also be used to distinguish words that can be used in different parts of speech.

2. Process of transforming data into numeric vectors:

1. The first step is to select the data which needs to be modelled for training the analyser.
2. stop words that are present in almost all documents such as a, an, the, is etc, must be removed from training data. Since they occur in almost every sentence they won't be useful at all for extracting sentiment information from the text
3. Further Data Pre-processing is done using regex by removing special characters, digits from our documents.
4. We use term-frequency to represent each term in our vector space. We determine term-frequency which is nothing more than a measure of how many times the terms present in our vocabulary. We can specify many parameters in Tf-Idf such as Max_df (maximum Document frequency), Lowercase(for converting all characters to lowercase), max_features (specifies No of terms used in building vocabulary).
5. we can define the term-frequency as a counting function:

$$\mathrm{tf}(t, d) = \sum_{x \in d} \mathrm{fr}(x, t)$$

- **Step 5:** Build Classifier

About Machine Learning Algorithm Used:

The machine Learning algorithm used is: Stochastic Gradient Descent Classifier from *Scikit-Learn*

Why this Particular Algorithm ? why not other commonly used algorithms such Naive-bayes, Decision tree or Logistic Regression.?

The main reason for using SGD Classifier for training our data model is that our training data is very large hence it is sparse, the classifiers in this module easily scale to problems with more than 10^5 training examples and more than 10^5 features..

The estimator implements regularized linear models with stochastic gradient descent (SGD) learning: the gradient of the loss is estimated each sample at a time and the model is updated along the way with a decreasing strength schedule (aka learning rate).

This results in a better accuracy as the data backpropagates to reduce error.

- **Step 6:** Train Final Model

Now we should train a model using the entire training set and evaluate our accuracy on the 25k test reviews.

- **Step 7:** Testing our Data Model:

We use scikit-learn library and calculate accuracy precision recall and f1 results for the testing data which is the other 25000 reviews. This helps us in determining how good our model has trained for Sentimental Analysis.

Also we print the confusion Matrix to determine True Positive, False Positive, True Negative and False Negative.

We can Visualize our Confusion Matrix using Matplotlib Library of Python for better Analytics of our results.

# Chapter 1

## INTRODUCTION

Movie reviews are an important way to gauge the performance of a movie. While providing a numerical/stars rating to a movie tells us about the success or failure of a movie quantitatively, a collection of movie reviews is what gives us a deeper qualitative insight on different aspects of the movie. A textual movie review tells us about the strong and weak points of the movie and deeper analysis of a movie review can tell us if the movie in general meets the expectations of the reviewer. Sentiment Analysis [1] is a major subject in machine learning which aims to extract subjective information from the textual reviews. The field of sentiment of analysis is closely tied to natural language processing and text mining. It can be used to determine the attitude of the reviewer with respect to various topics or the overall polarity of review. Using sentiment analysis, we can find the state of mind of the reviewer while providing the review and understand if the person was "happy", "sad", "angry" and so on. In this project we aim to use Sentiment Analysis on a set of movie reviews given by reviewers and try to understand what their overall reaction to the movie was, i.e. if they liked the movie or they hated it. We aim to utilize the relationships of the words in the review to predict the overall polarity of the review.

From a user's perspective, people are able to post their own content through various social media, such as forums, micro-blogs, or online social networking sites. From a researcher's perspective, many social media sites release their application programming interfaces (APIs), prompting data collection and analysis by researchers and developers. However, those types of online data have several flaws that potentially hinder the process of sentiment analysis. The first flaw is that since people can freely post their own content, the quality of their opinions cannot be guaranteed. he second flaw is that ground truth of such online data is not always available. A ground truth is more like a tag of a certain opinion, indicating whether the opinion is positive or negative or neutral.

**"**It is a quite boring movie……. but the scenes were good enough."

The given line is a movie review that states that "it" (the movie) is quite boring but the scenes were good. Understanding such sentiments require multiple tasks.

Hence, SENTIMENTAL ANALYSIS is a kind of text classification based on *Sentimental Orientation* (SO) of opinion they contain.

Sentiment analysis of product reviews has recently become very popular in text mining and computational linguistics research.

 Firstly, evaluative terms expressing opinions must be extracted from the review.

 Secondly, the SO, or the polarity, of the opinions must be determined.

 Thirdly, the opinion strength, or the intensity, of an opinion should also be determined.

 Finally, the review is classified with respect to sentiment classes, such as Positive and Negative, based on the SO of the opinions it contains.

# Chapter 2

## LITERATURE SURVEY

The original work [3] on this dataset was done by researchers at Stanford University wherein they used unsupervised learning to cluster the words with close semantics and created word vectors. They ran various classification models on these word vectors to understand the polarity of the reviews. This approach is particularly useful in cases when the data has rich sentiment content and is prone to subjectivity in the semantic affinity of the words and their intended meanings. Apart from the above, a lot of work has been done by Bo Pang [7] and Peter Turnkey [8] towards polarity detection of movie reviews and product reviews. They have also worked on creating a multi-class classification of the review and predicting the reviewer rating of the movie/product.

These works discussed the use of Random Forest classifier and SVMs for the classification of reviews and also on the use of various feature extraction techniques. One major point to be noted in these project was exclusion of a neutral category in classification under the assumption that neutral texts lie close to the boundary of the binary classifiers and are disproportionately hard to classify.

There are many sentiment analyses tools and software existing today that are available for free or under commercial license. With the advent of microblogging, sentiment analysis is being widely used to analyse the general public sentiments and draw inferences out of these. One famous application was use of Twitter to understand the political sentiment of the people in context of German Federal elections [9].

A max-entropy POS tagger is used in order to classify the words of the sentence, an additional python program to speed up the process. The negation words like no, not, and more are included in the adverbs whereas Negation of Adjective and Negation of Verb are specially used to identify the phrases.

The following are the various classification models which are selected for categorization: Naïve Bayesian, Random Forest, Logistic Regression and Support Vector Machine.

For feature selection, Pang and Lee suggested to remove objective sentences by extracting subjective ones. They proposed a text-categorization technique that is able to identify subjective content using minimum cut. Gann et al. selected 6,799 tokens based on Twitter data, where each token is assigned a sentiment score, namely TSI (Total Sentiment Index), featuring itself as a positive token or a negative token. Specifically, a TSI for a certain token is computed as:

$$TSI = \frac{p - \frac{tp}{tn} \times n}{p + \frac{tp}{tn} * n}$$

where p is the number of times a token appears in positive tweets and *n* is the number of times a token appears in negative tweets is $\frac{tp}{tn}$ the ratio of total number of positive tweets over total number of negative tweets.

The benefit of social media to know public opinions and extract their emotions are considered by authors in [2] and explained how twitter gives advantage politically during elections. Further, the concept of the hashtag is used for text

classification as it conveys emotion in few words. They suggested how previous research work suffered from lack of training set and misses some features of target data. They opted two stage approach for their framework- first preparing training data from twitter using mining conveying relevant features and then propounding the Supervised Learning Model to predict the results of Elections held in USA in 2016. After collecting and pre-processing the tweets, training data set was created first by manual labelling of hashtags and forming clusters, next by using online Sentimental Analyzer VADER which outputs the polarity in percentage. This approach reduced the number of tweets or training set and further they applied Support Vector Machine and Naive Bayes classification algorithm to determine the polarity of tweets. Multistage Classification approach was used where an entity classifier receives general class of tweets and categorise them with respect to individual candidates for comparison. The metric they used to determine the winner

Sentiment Analysis by researchers Imran et al. exploited the technology 'Apache Spark' for fast streaming of tweets and presented the approach Stream Sensing to handle real time data in unstructured and noisy form. They conducted the approach on twitter data to find some useful and interesting trends which further can be generalized to any real-time text stream. Unsupervised learning approach is used to locate interesting patterns and trends from tweets processed on Apache Spark. Inspired by the approach described by Zhu et al. and Li et al. for mining data by selecting time window, authors opted for sliding window method for capturing the live streams of tweets.

The common approach found in almost all relevant research works constitutes data collection using Twitter API, pre-processing of data, filtering of data then approaches in feature extraction, classification and pattern analysis makes the distinction. Authors used sliding window of 5 minutes during data collection and further created Term Document Matrix(TDM) for feature extraction. The pattern analysis was carried out by using the score of TF-IDF for finding most important keywords as explained by Wu et al. The trending topic or hashtag is fed and tweets relevant to it are filtered to form TDM and computing the weights of TF-IDF to find most important words is the key idea of this sentiment analysis. Parallel computation of TDM, TF-IDF score and determining top 5 keywords generated from TDM in each minute as the sliding window moves are one of the highlighting features of this research work. Thus, it leverages the fast computation power of Apache Spark.

# Chapter 3

## PROPOSED METHODOLOGY

This section describes the different pre-processing modules that have been used in our project. All of them are built in Python. The pipeline of the project is organized in the following way. In order to perform sentiment analysis, data have to be prepared in order to obtain a data set – namely, the training set. The training sets are subject to pre-processing techniques mentioned in the work. Then, such a data set is involved in the learning step, which uses Machine Learning [ML] algorithm and yields a trained classifier. After training the classifier it has to be tested on a different data set – namely the test set. Figure 2 shows the various steps for training a classifier to perform sentiment analysis

**Data collection :**

**DATASET : IMDB review dataset**:

You are to use a publicly available dataset consisting of 50,000 reviews from IMDB, allowing no more than 30 reviews per movie. The constructed dataset contains an equal number of positive and negative reviews, so random guessing yields 50% accuracy.

You can download the dataset from the following link:

http://ai.stanford.edu/~amaas/data/sentiment/

The dataset has three columns. The first column specifies a sequential number of the record. The second column has a text which is the review and the third column is a class denoting the sentiment of the reviewer. The class has value 1 if the sentiment is positive and 0 if the sentiment is negative.

'Our objective is to build a custom sentiment analyser that can classify sentiment (positive or negative) of reviewers out of their movie reviews.'

Our data set looks like this.

```
In [3]: print(df.shape)
(50001, 2)

In [4]: df.head(10)

Out[4]:
   sentiment                                             reviews
0  sentiment                                              review
1          1  Based on an actual story, John Boorman shows t...
2          1  This is a gem. As a Film Four production - the...
3          1  I really like this show. It has drama, romance...
4          1  This is the best 3-D experience Disney has at ...
5          1  Of the Korean movies I've seen, only three had...
6          1  this movie is funny funny funny my favorite qu...
7          1  I'm just starting to explore the so far wonder...
8          1  There is no need for me to repeat the synopsis...
9          1  I got this movie with my BBC "Jane Austen Coll...
```

Figure 5. Head of the dataset – overview of data

**Pre-processing Basic Operation and Cleaning:**

This first module manages basic cleaning operations, which consist in removing unimportant or disturbing elements for the next phases of analysis and in the normalization of some misspelled words. In order to provide only significant information, in general a clean review should not contain URLs and hashtags (i.e. #happy). Furthermore, tabs and line breaks should be replaced with a blank and quotation marks with apexes. After this step, all the punctuation is removed, except for apexes, because they are part of grammar constructs such as the genitive. The next operation is to remove the vowels repeated in sequence at least three times, because by doing so the words are normalized: for example, two words written in a different way (i.e. cooooool and cool) will become equals. The last step is to convert many types of emoticons into tags that express their sentiment (i.e. :) → smile happy). The list of emoticons is taken from Wikipedia. Finally, all the text is converted to lower case, and extra blank spaces are removed. All the operations in this module are executed to try to make the text uniform. This is important because during the classification process, features are chosen only when they exceed a certain frequency in the data set. Therefore, after the basic pre-processing operations, having different words written in the same way helps the classification.

**Dictionary:**

Here we make use of the external python library PyEnchant, which provides a set of functions for the detection and correction of misspelled words using a dictionary. It also allows us to substitute slang with its formal meaning (i.e., l8 → late), using a list. It also allows us to replace insults with the tag "bad word". The reason for the use of these functions is the same as for the basic pre-processing operation, i.e. to reduce the noise in text and improve the overall classification performances.

**Negation:**

Dealing with negations (like "not good") is a critical step in Sentiment Analysis. A negation word can influence the tone of all the words around it, and ignoring negations is one of the main causes of misclassification. In this phase, all negative constructs (can't, don't, isn't, never etc.) are replaced with "not". This technique helps classifier model to be enriched with a lot of negation constructs that would otherwise be excluded due to their low frequency as detailed in Fig. 2, Machine Learning algorithms need to work on data that is properly processed. This phase is a fundamental step in order for the whole system to obtain good results. Normally it includes methods for data cleaning and feature extraction and selection. A good overview of the steps and the most known algorithms for each step is explained in [13].

# Chapter 4

## OVERVIEW OF NLP TERMS

**NLP (Natural language processing)** is a component of AI that helps computers understand, manipulate and interpret human language as it is spoken. It is an intersection of computer science, artificial intelligence, and computational linguistics.

**Sentiment analysis** refers to the analysis to systematically identify, extract, quantify, and study affective states and subjective information from any form of data such as reviews and survey responses, online and social media responses, etc. It uses natural language processing, text analysis, computational linguistics, and biometrics for the analysis. Sentiment analysis is also known as opinion mining or emotion AI.

**Lemmatisation** refers to the reduction of a group of words into their respective lemma or dictionary form. POS (Parts of Speech) i.e., the meaning of the word in the nearby sentences, etc. are taken into account before reducing the word to its lemma.

**Tokenization** refers to the segmentation of running text into sentences and words. With tokenization, the text is cut into pieces called tokens along with throwing away certain characters, such as punctuation.

**Information Extraction** is the process of automatically extracting structured information from unstructured and/or semi-structured sources.

**Named Entity Recognition (NER)** is the process by which text is located and classified into predefined categories such as the names of people, address, organizations, places, monetary values, gender, etc.

**Bag of Words** is a commonly used model while Text Classification. With BOW, a piece of text such as a sentence or a document is represented as a bag. BOW simplifies the contents of a selection of text. It omits grammar and word order but is interested in the number of occurrences of words within the text. The frequency of occurrence of each word in the bag is used for training a classifier.

For example: Consider the sample text as follows:

"Hello, Hello, Hello, Hello," said Ankita.

"There, there," said Rekha. "There,"

The resulting bag of words representation as a dictionary:

```
{
  ' Hello ': 4,
  'said': 2,
  'Ankita': 1,
  'there': 3,
  'Rekha': 1
}
```

**Information retrieval** is the process by which the most appropriate information is accessed or retrieved from text based on a particular query. This is done by using context-based indexing or metadata.
For example: Search engines like Google Search.

**Stop Words:**
Stop words are a division of natural language. The reason that stop-words should be removed from a text is that they make the text look heavier. For it we have various stop word removing techniques [5] were referred. Removing stop words helps in reducing the dimensionality of term space. The most common words in text documents are articles, prepositions, etc that do not give the meaning to the documents. These are the words which are treated as stop words. Example: the, in, a, an, with, etc. It is important to avoid having these words within the classifier model, because they can lead to a less accurate classification.

**Stemming:**
Stemmers remove morphological affixes from words, leaving only the word stem. For example, the words connect, connected, connecting, connections all can be stemmed to the word “connect”. The purpose of this method is to remove various suffixes, to have accurately matching stems, to save time and memory space. This is illustrated in Fig. 3 Translation of morphological forms of a word to its stem is done assuming that the words are semantically related. There are two points are considered while using a stemmer:
- Words that do not have the same meaning should be kept separate
- Morphological forms of a word are assumed to have the same base meaning and hence it should be mapped to the same stem

These two rules are good and sufficient in language processing applications. For our Project we will be using Snowball Stemmer with the help of NLTK toolkit. It is an evolution of the original Porter Stemmer algorithm and is computationally fast and more efficient.

For example, A stemming algorithm reduces the words “plays”, “played”, “playing” ,” player” to the root word, “play”.

Similarly, the stemming algorithm reduces the words “presenting”, “presentation”, “presents” ,” presented” to the root word, “present”.

**POS Tagging:**

The Part-Of-Speech of a word is a linguistic category which is defined by what is known as its syntactic or morphological behavior. Common POS categories in English grammar are: noun, verb, adjective, adverb, pronoun, preposition, conjunction, and interjection. POS tagging is the task of labeling (or tagging) each word in a sentence with its appropriate part of speech. POS tagging is an important phase of opinion mining, it is essential to determine the features and opinion words from the reviews. POS tagging can be done either manually or with the help of POS tagger tool. POS tagging of the reviews by human is time consuming. POS tagger is used to tag all the words of reviews. Stanford tagger is used to tag each word in an online review sentence. Every one sentence in customer reviews are tagged and stored in text file

# Chapter 5

## OVERVIEW OF NLTK AND NLP WITH PYTHON

NLTK is one of the leading platforms for working with human language data and Python, the module NLTK is used for natural language processing. NLTK is literally an acronym for Natural Language Toolkit.

sudo pip3 install nltk

Installation is not complete after these commands. Open python and type:

import nltk

nltk.download()

Download all the required packages which may take a while, the bar on the bottom shows the progress.

**Tokenizing sentences**

A document or data can be split into sentences using the method **sent_tokenize()**

In [1]:

```
from nltk.tokenize import sent_tokenize

data = "All work and no play makes jack dull boy. All work and no play makes jack a dull boy."
print(sent_tokenize(data))
['All work and no play makes jack dull boy.', 'All work and no play makes jack a dull boy.']
```

### Tokenize words

Similar way, a sentence or data can be split into words using the method **word_tokenize()**

In [2]:

```
from nltk.tokenize import word_tokenize

data = "All work and no play makes jack a dull boy, all work and no play"
print(word_tokenize(data))
['All', 'work', 'and', 'no', 'play', 'makes', 'jack', 'a', 'dull', 'boy', ',', 'all', 'work', 'and', 'no', 'play']
```

In [3]:

```
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords

data = "All work and no play makes jack a dull boy, all work and no play"
words = word_tokenize(data)
```

```
stop_words = set(stopwords.words('english'))
print(stop_words)
```

```
{'at', 'here', 'once', 'or', 'them', 'doing', 'own', 'in', 'haven', 'further', 'for', 'if', 'than', 'aren', 'where', 'be', 'being', 'll', 'what', 'while', 'through', 'nor', 'yours', 'only', 'm', 'not', 'against', 'will', 'an', 'did', 'she', 'were', 'but', 'isn', 'few', 'which', 'just', 'myself', 'more', 'd', 'and', 'themselves', 'to', 'needn', 're', 'yourself', 'such', 'your', 'it', 'a', 'off', 'been', 'her', 'now', 'the', 'had', 'we', 'down', 'him', 'above', 'mightn', 'ma', 'on', 'no', 'ain', 'hasn', 'ours', 'both', 's', 'you', 'am', 'weren', 'don', 'shouldn', 'our', 'me', 'does', 'has', 'of', 'doesn', 't', 'before', 'theirs', 'each', 'over', 'out', 'most', 'yourselves', 'shan', 'because', 'have', 'into', 'during', 'too', 'itself', 'very', 'didn', 'this', 'himself', 'same', 'up', 'any', 'when', 'do', 'hadn', 'i', 'are', 'whom', 'other', 'again', 'with', 'won', 'couldn', 'these', 'his', 'why', 'then', 'o', 'hers', 'my', 'their', 'from', 'how', 'is', 'ourselves', 'mustn', 'its', 'should', 'he', 'as', 'wouldn', 'was', 'there', 'herself', 'they', 'y', 'wasn', 'that', 'under', 'so', 'some', 've', 'between', 'by', 'those', 'having', 'all', 'below', 'can', 'after', 'who', 'about', 'until'}
```

In [4]:

```
filtered_word = []

for wrd in words:
    if wrd not in stop_words:
        filtered_word.append(wrd)

print(filtered_word)
```

```
['All', 'work', 'play', 'makes', 'jack', 'dull', 'boy', ',', 'work', 'play']
```

## NLTK - stemming

A word stem is part of a word. It is sort of a normalization idea, but linguistic. For example, the stem of the word **waiting** is **wait**.

In [5]:

```
from nltk.stem import PorterStemmer
from nltk.tokenize import sent_tokenize, word_tokenize

words = ["game","gaming","gamed","games"]
ps = PorterStemmer()

for word in words:
    print(ps.stem(word))
```

```
game
game
game
game
```

## NLTK - speech of tagging (POS)

Given a sentence or paragraph, it can label words such as verbs, nouns and so on.

In [6]:

```
import nltk
from nltk.tokenize import PunktSentenceTokenizer

document = 'Whether you\'re new to programming or an experienced developer, it\'s easy to learn and use Python.'
```

```
sentences = nltk.sent_tokenize(document)
for sent in sentences:
    print(nltk.pos_tag(nltk.word_tokenize(sent)))
```

```
[('Whether', 'IN'), ('you', 'PRP'), ("'re", 'VBP'), ('new', 'JJ'), ('to', 'TO'), ('programming', 'VBG'), ('or', 'CC'), ('an', 'DT'), ('experienced', 'JJ'), ('developer', 'NN'), (',', ','), ('it', 'PRP'), ("'s", 'VBZ'), ('easy', 'JJ'), ('to', 'TO'), ('learn', 'VB'), ('and', 'CC'), ('use', 'VB'), ('Python', 'NNP'), ('.', '.')]
```

We can filter this data based on the type of word, especially if we are looking for nouns.

In [7]:

```
import nltk
from nltk.corpus import state_union
from nltk.tokenize import PunktSentenceTokenizer

document = 'Two military helicopters and six boats of the National Disaster Response Force were sent in to rescue as many as 900 people stranded on board a passenger train about 60 km out of Mumbai on Saturday after heavy rain paralysed the city and its surrounding areas.'
sentences = nltk.sent_tokenize(document)

data = []
for sent in sentences:
    data = data + nltk.pos_tag(nltk.word_tokenize(sent))

#print(data)

for word in data:
    if word[1] in ['NN', 'NNP', 'NNS']:
        print(word)
```

```
('helicopters', 'NNS')
('boats', 'NNS')
('National', 'NNP')
('Disaster', 'NNP')
('Response', 'NNP')
('Force', 'NNP')
('people', 'NNS')
('board', 'NN')
('passenger', 'NN')
('train', 'NN')
('km', 'NNS')
('Mumbai', 'NNP')
('Saturday', 'NNP')
('rain', 'NN')
('city', 'NN')
('areas', 'NNS')
```

# Chapter 6

## SENTIMENT ANALYSER

**Build a sentiment analyser**

There are many methods and algorithms to implement sentiment analysis systems, which can be classified as:

Rule-based systems – perform sentiment analysis based on a set of rules, prepared by experts. Modelled-based systems – rely on machine learning techniques to learn from data. Apart from these two, there is a hybrid systems that combine both rule based and modelled based approaches.

Rule-based Systems

Usually, rule-based systems run on a set of rules that identify subjectivity, polarity etc.

We can summarize a basic rule-based systems in following steps:

· Define two lists of polarized words negative words and positive words. Some examples of negative words are awful, bad, worst, etc. and some positive words are beautiful, good, best, etc.

· Given a text:

o Count the number of positive words in the text.

o Count the number of negative words in the text.

If the number of positive words is greater than the number of negative words then return a positive sentiment, conversely, return a negative sentiment. Otherwise, return neutral.

This system is very immature since it doesn't take into account how words are combined in a sequence. The above system can be made sophisticated by introducing intensity score of each words in the lists. For example, intensity score of ‘best’ is higher than ‘good’. And instead of comparing counts, one can use aggregated score.

The rule-based systems are very hard to maintain as new rules may be needed to add support for new expressions and vocabulary. As a result, these systems require important investments in manually tuning and maintaining the rules.

Modelled-based Systems

Modelled-based systems are based on machine learning techniques. The sentiment analysis task is usually modelled as a classification problem where a classifier returns the corresponding category, e.g. positive, negative, or neutral for an input text.

Building sentiment classifiers may be tedious task depending on what level of sentiment analysis the model will be going to perform.

A simple sentiment analyser that classifies a text either positive, negative, or neutral categories, can be implemented with the following steps:

The Training Processes

· Training data preparation: it is important to create diversified training data to train a model. Remember, all machine learning procedures perform well with input similar to training data. That is why, need to create a comprehensive list of text. Each text must be tagged with any from a predefined categories like positive, negative or neutral. The entire dataset is split randomly into two sets called training set and test set. Training set consists of around 75% of entire data. Remaining data belongs to test set.

· Train the model to build a classifier: Pick a classification procedure based on statistical model (Naïve Bayes, Logistic Regression, CART, Discriminant Analysis or Support Vector Machines ) or based on Artificial Neural Networks, then train the model with training set.

· Validate the classifier: Use test set to validate the model. Test set contains text which are not part of the training and all text are tagged. It is comparison between actual tag and tag generated by classifier.

# Chapter 7

# BUILDING CUSTOM SENTIMENT ANALYSER : TF-IDF

Here, we have taken a portion of the IMDB review dataset with 1,000 data. We will use 500 reviews as training data and remaining 500 reviews as the test data. The dataset is available here. First, we will apply TF-IDF on the training data to convert the text into numeric vector. Let us see step by step process. Let's upload the data in pandas DataFrame object. A randomly chosen 1000 reviews are separated and store in a file.

```
import pandas as pd
df =
pd.read_csv("https://spotleai.sgp1.digitaloceanspaces.com/course/data/movie_review_data_1000.csv",
encoding='utf-8')
print(df.head(10))
```

The following output shows the first 10 observations from the dataset. Let us separate all *review text* from data frame to a list.

| | Review | Sentiment |
|---|---|---|
| 0 | JESSICA: A GHOST STORY is as the name implies ... | 0 |
| 1 | After seeing Arthur on TV numerous times I lau... | 1 |
| 2 | Set in Venice mainly on the Lido, Visconti's "... | 1 |
| 3 | Morris and Reva Applebaum had been the toast o... | 1 |
| 4 | Based on fact, this is the story of a teenager... | 1 |
| 5 | Not to be confused with the Resse Witherspoon ... | 1 |
| 6 | When I attended college in the early 70s, it w... | 0 |
| 7 | About 15 minutes in, my wife was already wanti... | 0 |
| 8 | I was disappointed with the recent (2000) Amer... | 1 |
| 9 | Katherine Heigl, Marley Shelton, Denise Richar... | 1 |

Figure 5. Head of the dataset – overview of data

```
review = list(df['review'].values)
```

sklearn has a method called TfidfVectorizer to Convert a collection of raw documents to a matrix of TF-IDF features. TfidfVectorizer first tokenizes documents, learns the vocabulary, inverses document frequency weightage, and allows to encode new documents. The following piece of code does the same on 1000 review text.

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer()
review_vectors = tfidf.fit_transform(review)
print(review_vectors.shape)
```

(1000, 18506)

review_vectors holds 18506 numeric features for each of the 1000 reviews. Let's save the transformed features along with sentiment information to a file, so that the file can be used for further analysis.

```
df_sentiment = pd.DataFrame(review_vectors.todense(), columns=['col_'+ str(i+1) for i in range(review_vectors.shape[1])])
df_sentiment['sentiment'] = df['sentiment']
df_sentiment.to_csv('review_1000_vectors.csv', index = False)
```

```
print(df_sentiment.tail())
```

The following is the output, that shows the last 5 reviews in the file. The file contains the TF-IDF score generated for all the words that are there in 1000 reviews. This is the numeric vector input variable converted from the text.

```
     col_1 col_2 col_3 col_4 col_5   col_6 col_7 col_8 col_9 col_10 \
995  0.0  0.0  0.0  0.0  0.0 0.000000  0.0  0.0  0.0   0.0
996  0.0  0.0  0.0  0.0  0.0 0.000000  0.0  0.0  0.0   0.0
997  0.0  0.0  0.0  0.0  0.0 0.000000  0.0  0.0  0.0   0.0
998  0.0  0.0  0.0  0.0  0.0 0.000000  0.0  0.0  0.0   0.0
999  0.0  0.0  0.0  0.0  0.0 0.046999  0.0  0.0  0.0   0.0

     ... col_18498 col_18499 col_18500 col_18501 col_18502 col_18503 \
995 ...    0.0    0.0    0.0    0.0    0.0    0.0
996 ...    0.0    0.0    0.0    0.0    0.0    0.0
```

```
997 ...    0.0    0.0    0.0    0.0    0.0    0.0

998 ...    0.0    0.0    0.0    0.0    0.0    0.0

999 ...    0.0    0.0    0.0    0.0    0.0    0.0


     col_18504 col_18505 col_18506 sentiment

995    0.0    0.0    0.0     1

996    0.0    0.0    0.0     1

997    0.0    0.0    0.0     0

998    0.0    0.0    0.0     1

999    0.0    0.0    0.0     0


[5 rows x 18507 columns]
```

We will use Naïve Bayes Classifier on this and study the accuracy of the classifier. We will use the first 500 data to train the model and last 500 data to validate the model. But before that let us learn how Doc2Vec is used to convert the same text into numeric vector here.

- **<u>Stochastic Gradient Descent (SGD):</u>**

The word '*stochastic*' means a system or a process that is linked with a random probability. Hence, in Stochastic Gradient Descent, a few samples are selected randomly instead of the whole data set for each iteration. In Gradient Descent, there is a term called "batch" which denotes the total number of samples from a dataset that is used for calculating the gradient for each iteration. In typical Gradient Descent optimization, like Batch Gradient Descent, the batch is taken to be the whole dataset. Although, using the whole dataset is really useful for getting to the minima in a less noisy or less random manner, but the problem arises when our datasets get really huge.
Suppose, you have a million samples in your dataset, so if you use a typical Gradient Descent optimization technique, you will have to use all of the one million samples for completing one iteration while performing the Gradient Descent, and it has to be done for every iteration until the minima is reached. Hence, it becomes computationally very expensive to perform.

This problem is solved by Stochastic Gradient Descent. In SGD, it uses only a single sample, i.e., a batch size of one, to perform each iteration. The sample is randomly shuffled and selected for performing the iteration.

$$for\ i\ in\ range\ (m):$$

$$\theta_j = \theta_j - \alpha\,(\hat{y}^i - y^i)\,x_j^i$$

Figure 6. Formula used for SDG Algorithm

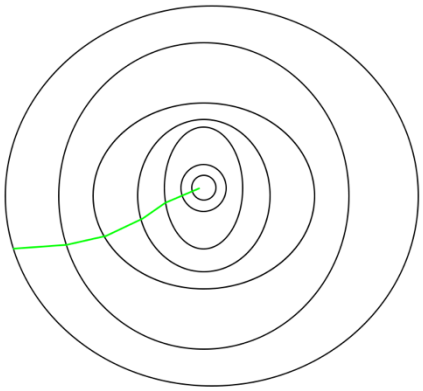

Figure 7. Path taken by Batch Gradient Descent.

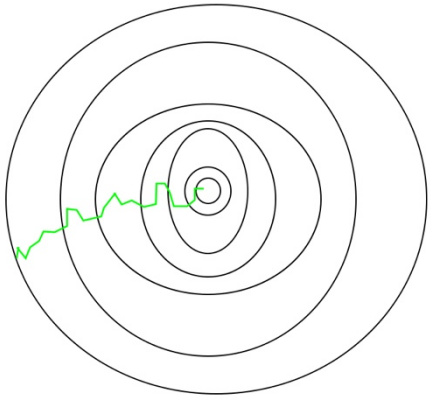

Figure 8. Path taken by Stochastic Gradient Descent

One thing to be noted is that, as SGD is generally noisier than typical Gradient Descent, it usually took a higher number of iterations to reach the minima, because of its randomness in its descent. Even though it requires a higher number of iterations to reach the minima than typical Gradient Descent, it is still computationally much less expensive than typical Gradient Descent. Hence, in most scenarios, SGD is preferred over Batch Gradient Descent for optimizing a learning algorithm.

# Chapter 8

# BUILDING CUSTOM SENTIMENT ANALYZER: DOCEMENT EMBEDDING

Document Embeddings

Document embedding is an extension of word embedding. Word embedding is the collective name for a set of language modelling and feature learning techniques in natural language processing (NLP) where words or phrases from the vocabulary are mapped to vectors of real numbers. Conceptually it involves a mathematical embedding from a space with one dimension per word to a continuous vector space with a much lower dimension. Document Embeddings are the texts converted into numbers and there may be different numerical representations of the same text. Document vectors are constructed by vector of words in the documents.

Here, we have taken a portion of the IMDB review dataset with 1,000 data. The dataset is available here. First, we will apply document embeddings on the data to convert the text into numeric vector. Let us see step by step process. First, we will upload the data in pandas DataFrame object. We have randomly chosen 1000 reviews; separated and stored in a file.

```
import pandas as pd
df =
pd.read_csv("https://spotleai.sgp1.digitaloceanspaces.com/course/data/movie_review_data_1000.csv",
encoding='utf-8')
print(df.head(10))
```

The following output shows the first 10 observations from the dataset.

| | Review | Sentiment |
|---|---|---|
| 0 | JESSICA: A GHOST STORY is as the name implies ... | 0 |
| 1 | After seeing Arthur on TV numerous times I lau... | 1 |
| 2 | Set in Venice mainly on the Lido, Visconti's "... | 1 |
| 3 | Morris and Reva Applebaum had been the toast o... | 1 |
| 4 | Based on fact, this is the story of a teenager... | 1 |
| 5 | Not to be confused with the Resse Witherspoon ... | 1 |
| 6 | When I attended college in the early 70s, it w... | 0 |
| 7 | About 15 minutes in, my wife was already wanti... | 0 |
| 8 | I was disappointed with the recent (2000) Amer... | 1 |
| 9 | Katherine Heigl, Marley Shelton, Denise Richar... | 1 |

Figure 5. Head of the dataset – overview of data

Let's split the dataset into training dataset and test dataset. The training dataset contains first 500 reviews and respective sentiments. The test dataset contains last 500 reviews and respective sentiments.

```
X_train = list(df.loc[:499, 'review'].values)
X_test = list(df.loc[500:, 'review'].values)
```

Gensim library has Doc2Vec model for document embedding. The following python code illustrates Doc2Vec model.

```
from gensim.models.doc2vec import Doc2Vec, TaggedDocument
from nltk.tokenize import word_tokenize
```

```
tagged_data = [TaggedDocument(words=word_tokenize(_d.lower()), tags=[str(i)]) for i, _d in
enumerate(X_train)]
```

```
max_epochs = 50
vec_size = 100
alpha = 0.025

model = Doc2Vec(vector_size=vec_size,
        alpha=alpha,
        min_alpha=0.00025,
        min_count=1,
        dm =1)

model.build_vocab(tagged_data)

for epoch in range(max_epochs):
  model.train(tagged_data,
        total_examples=model.corpus_count,
        epochs=model.epochs)
  model.alpha -= 0.0002
  model.min_alpha = model.alpha

model.save("doc.vec")
print("Model Saved")
```

```
Model Saved
```

```
from gensim.models.doc2vec import Doc2Vec

model= Doc2Vec.load("doc.vec")
```

```
import numpy as np

narr = np.append([], [])
for text in X_train:
  txt = word_tokenize(text.lower())
  vect = model.infer_vector(txt)
  narr = np.append(narr, vect)
```

```
for text in X_test:
  txt = word_tokenize(text.lower())
  vect = model.infer_vector(txt)
  narr = np.append(narr, vect)
narr = np.resize(narr, (1000, vec_size))
narr.shape
```

Output is

(1000, 100)

```
df_trans = pd.DataFrame(narr, columns=['col_'+ str(i+1) for i in range(narr.shape[1])])
df_trans['sentiment'] = df['sentiment']
df_trans.to_csv('review_1000_doc2_vectors.csv', index=False)
df_trans.tail()
```

The following is the last five reviews of the file, that is, the embedding vectors generated for all the reviews that are there in the 1000 reviews. This is the numeric vector input variable converted from the text.

| | col_1 | col_2 | col_3 | col_4 | col_5 | col_6 | col_7 | col_8 | col_9 | col_10 | ... | col_92 | col_93 | col_94 | col_95 | col_96 | col_97 | col_98 | col_99 | col_100 | sentiment |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 995 | 0.132904 | -0.50509 | 0.457686 | -0.668819 | -0.490391 | 0.171646 | -0.206709 | 0.188175 | -0.016879 | -0.091713 | ... | 0.272456 | 0.604319 | 0.114643 | 0.524703 | -0.160662 | -0.077891 | -0.046372 | -0.019794 | 0.373469 | 1 |
| 996 | -0.689388 | -0.588956 | -0.232552 | -0.789492 | -0.053097 | 0.380332 | 0.797743 | -0.95297 | -0.799754 | 0.771582 | ... | 0.520564 | 1.804569 | -0.367148 | 1.04524 | 0.640017 | 0.366299 | 0.383579 | 0.202839 | 0.54003 | 1 |
| 997 | 0.064988 | -0.643349 | -0.497534 | -0.286019 | 1.189623 | 0.158624 | 0.527183 | 0.6059 | 0.197497 | -0.783868 | ... | 0.215794 | 1.09324 | -0.466265 | 0.797427 | 0.423468 | -1.614433 | -0.089704 | -0.233564 | -0.006417 | 0 |
| 998 | 0.210158 | 0.008131 | 0.061542 | -0.649481 | -0.314281 | -0.183619 | -0.466841 | 0.148559 | -1.037212 | 0.242247 | ... | 0.349194 | 0.30107 | 0.118117 | -0.349806 | 0.332339 | -0.591167 | -0.277603 | 0.218451 | 0.231355 | 1 |
| 999 | -1.08312 | 0.050585 | -0.319282 | -0.703648 | 0.888707 | 0.841766 | -0.00464 | -0.114184 | -0.295284 | 0.06537 | ... | 0.578929 | 0.371921 | -0.683825 | -0.056885 | 1.2088 | 0.755461 | 1.140789 | -0.676443 | 0.689438 | 0 |

Figure 9. Numeric vector input variable converted from the text.

5 rows × 101 columns

Let us now see how classification is done using the numeric vector. We will use the TF-IDF score we generated earlier.

# Chapter 9

# BUILDING CUSTOM SENTIMENT ANALYZER: TF-IDF WITH NAÏVE BAYES CLASSIFIER

TF-IDF Score followed by Naïve Bayes Classifier

We will apply Naïve Bayes Classifier on the dataset which has been generated from the TF-IDF Score. We will see the process step by step. First let's upload the dataset into a pandas DataFrame object.

```
import pandas as pd
df = pd.read_csv("review_1000_tfidf_vectors.csv")
```

The next step is to split the dataset into training data and test data. The training dataset contains first 500 rows and the test dataset contains remaining 500 rows.

```
y = df.pop('sentiment')
X = df.values

X_train = df.loc[:499].values
X_test = df.loc[500:].values
y_train = y.loc[:499:].values
y_test = y.loc[500:].values
```

Now, let's discuss few points on Naïve Bayes Classifiers.

MultinomialNB class in sklearn, implements the Naïve Bayes algorithm for multinomially distributed data, and is one of the most used versions of Naïve Bayes in text classification, where the data are typically represented as word vector counts, TF-IDF vectors. The distribution is parameterized by vectors xc=(xc1,…,xcn) for each class c, where n is the number of features, in our case, the size of the vocabulary and xci is the probability of feature i appearing in a sample belonging to class c.

The multinomial distribution normally requires integer feature counts. However, in practice, fractional counts also work. But all values must be non negative.

```
#Let's import MultinomialNB
from sklearn.naive_bayes import MultinomialNB
```

**Some important parameters in MultinomialNB:**

***alpha:*** *float, optional (default=1.0) - Additive (Laplace/Lidstone) smoothing parameter (0 for no smoothing).*

***fit_prior:*** *boolean, optional (default=True) - Whether to learn class prior probabilities or not. If false, a uniform prior will be used.*

***class_prior:*** *array-like, size (n_classes,), optional (default=None) - Prior probabilities of the classes. If specified the priors are not adjusted according to the data.*

```
model = MultinomialNB().fit(X_train, y_train)

from sklearn.metrics import accuracy_score
predicted = model.predict(X_test)
acc = round(accuracy_score(y_test, predicted)*100,2)
print("Accuracy of the model with test dataset: ", acc)
```

Accuracy of the model with test dataset is 49.2

As we know, we need a lot of data to build a classifier using ML algorithms. The dataset with 1000 reviews is just to demonstrate the flow of building your own sentiment analyzer. Accuracy will increase with the increase in number of reviews.

# Chapter 10

# BUILDING CUSTOM SENTIMENT ANALYZER: DOCUMENT EMBEDDINGS WITH DECISION TREE

Document Embeddings followed by Decision Tree Classifier

We will apply Decision Trees Classifier on the dataset that has been generated from the document embeddings as explained in Building Custom Sentiment Analyzer: Hands-on Part 2. We will see the process step by step. First let's upload the dataset into a pandas DataFrame object.

```
import pandas as pd
df = pd.read_csv("review_1000_doc2_vectors.csv")
```

The next step is to split the dataset into training data and test data. The training dataset contains first 500 rows and the test dataset contains remaining 500 rows.

```
y = df.pop('sentiment')
X = df.values
X_train = df.loc[:499].values
X_test = df.loc[500:].values
y_train = y.loc[:499:].values
y_test = y.loc[500:].values
```

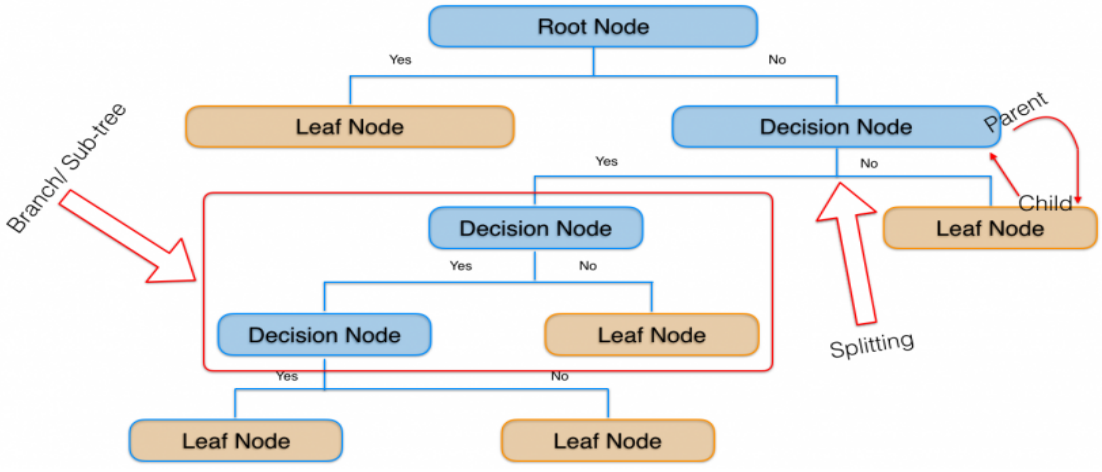


Figure 10. Decision Trees classifiers

Here is a recap of Decision Trees classifiers.

Decision Trees are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. For instance, in the example below, decision trees learn from data to approximate a non linear curve with a set of if-then-else decision rules. Deeper the tree, more complex is the decision rules, better is the fitment of the model.

Some advantages of decision trees are:

- Simple to understand and to interpret. Trees can be visualised.
- Requires little data preparation. Other techniques often require data normalisation, creation of dummy variables and removal of blank values. Note that, this module does not support missing values.
- The cost of using the tree (i.e. predicting data) is logarithmic in the number of data points used to train the tree.
- Able to handle both numerical and categorical data. Other techniques are usually specialised in analyzing datasets that have only one type of variable.
- Able to handle multi-output problems.
- Uses a white box model. If a given situation is observed in a model, the explanation for the condition is easily explained by boolean logic. By contrast, in a black box model (e.g. in an artificial neural network), results may be more difficult to interpret.
- Possible to validate a model using statistical tests. That makes it possible to account for the reliability of the model.
- Performs well even if its assumptions are somewhat violated by the true model from which the data were generated.

```
#Let's import DecisionTreeClassifier
from sklearn.tree import DecisionTreeClassifier
```

Let's start building our decision tree classifier. Our objective is to build a decision tree based model, that will predict the sentiment variable (that means whether a review is positive or not) of the reviews based on the observations in 100 features obtained from Doc2vec.

```
model = DecisionTreeClassifier()
```

**Description of parameters**

- **criterion**: string, optional (default="gini"). The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain.

- **splitter**: string, optional (default="best"). The strategy used to choose the split at each node. Supported strategies are "best" to choose the best split and "random" to choose the best random split.
- **max_depth**: int or None, optional (default=None). The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.
- **min_samples_split**: int, float, optional (default=2). The minimum number of samples required to split an internal node:
- If int, then consider min_samples_split as the minimum number.
- If float, then min_samples_split is a fraction and ceil(min_samples_split * n_samples) are the minimum number of samples for each split.
- **min_samples_leaf**: int, float, optional (default=1). The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min_samples_leaf training samples in each of the left and right branches. This may have the effect of smoothing the model, especially in regression.
- If int, then consider min_samples_leaf as the minimum number.
- If float, then min_samples_leaf is a fraction and ceil(min_samples_leaf * n_samples) are the minimum number of samples for each node.
- **min_weight_fraction_leaf**: float, optional (default=0.). The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Samples have equal weight when sample_weight is not provided.
- **max_features**: int, float, string or None, optional (default=None). The number of features to consider when looking for the best split:
- If int, then consider max_features features at each split.
- If float, then max_features is a fraction and int(max_features * n_features) features are considered at each split.
- If "auto", then max_features=sqrt(n_features).
- If "sqrt", then max_features=sqrt(n_features).
- If "log2", then max_features=log2(n_features).
- If None, then max_features=n_features.
- **random_state**: int, RandomState instance or None, optional (default=None). If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator; If None, the random number generator is the RandomState instance used by np.random.
- **max_leaf_nodes**: int or None, optional (default=None). Grow a tree with max_leaf_nodes in best-first fashion. Best nodes are defined as relative reduction in impurity. If None then unlimited number of leaf nodes.
- **min_impurity_decrease**: float, optional (default=0.). A node will be split if this split induces a decrease of the impurity greater than or equal to this value.
- **min_impurity_split**: float, (default=1e-7). Threshold for early stopping in tree growth. A node will split if its impurity is above the threshold, otherwise it is a leaf. Use min_impurity_decrease instead.
- **class_weight**: dict, list of dicts, "balanced" or None, default=None. Weights associated with classes in the form {class_label: weight}.
- **presort**: bool, optional (default=False). Whether to presort the data to speed up the finding of best splits in fitting. For the default settings of a decision tree on large datasets, setting this to true may slow down the training process. When using either a smaller dataset or a restricted depth, this may speed up the training.

We will now fit the model with the training data.

```
model.fit(X_train, y_train)

DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
    max_features=None, 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, presort=False, random_state=None,
    splitter='best')
```

Our decision tree based classifier is ready. We will now validate our model with the test data. The test data will tell us how efficient our decision tree classifier is. We will calculate the predicted output for each subject in the test data. Since, we already know the actual outputs of the test dataset, we can find out the accuracy of this classification model by comparing actual outputs and predicted outputs.

```
from sklearn.metrics import accuracy_score
predicted = model.predict(X_test)
acc = round(accuracy_score(y_test, predicted)*100,2)
print("Accuracy of the model with test dataset: ", acc)
```

Accuracy of the model with test dataset: 51.0

As we know, we need a lot of data to build a classifier using ML algorithms. The dataset with 1000 reviews is just to demonstrate the flow of building your own sentiment analyzer. Accuracy will increase when we will consider more number of reviews.

# Chapter 11

# APPLICATIONS OF SENTIMENT ANALYSIS

## Applications

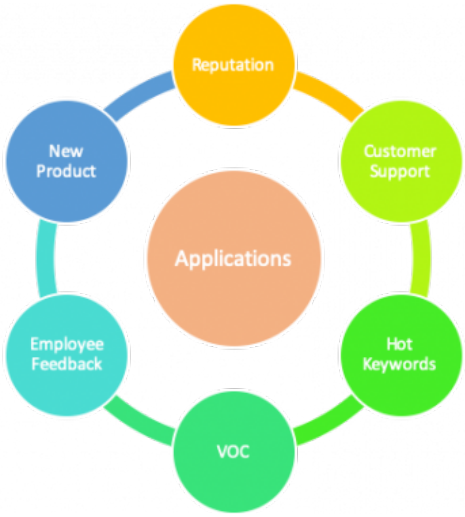


Figure 11. Applications of Sentiment Analysis

The applications for sentiment analysis are numerous. It is used in social media monitoring, customer reviews, survey responses, competitors, etc. However, it is also practical for use in business analytics and situations in which text needs to be analysed.

Sentiment analysis is in demand because of its efficiency. Thousands of text documents can be processed for sentiment in seconds, compared to the hours it would take a team of people to manually complete. Many businesses are incorporating sentiment analysis into their processes.

Reputation management – This is known as brand monitoring. We all know how much good reputation means these days when the majority of us check social media reviews as well as review sites before making a purchase decision. Most of time, when we visit a new restaurant, we do check its review beforehand. The same thing applies to buying stuff online, or researching tools I use daily at work as a marketer.

Negative reviews put people off and how you handle them can define your future as a business. You can ignore them at your own risk. But if you are aware of such opinions in the first place, you can take pre-emptive steps to restore customer faith. That's where social media monitoring combined with sentiment analysis comes in to help you manage your reputation, which eventually will keep your users happy.

Customer support - Social media are channels of communication with your customers. Whenever they're unhappy about something related to you, whether or not it's your fault, they will call you out on Facebook/Twitter/Instagram.

When you are not happy with a product or service you could first try to contact customer care, or just post the issues on social media tagging concerned departments. Nowadays we expect brands to respond on social media almost immediately. And if as a brand you're not quick enough, you may see users moving on to your competitors. Sentiment analysis helps categorising user complaints into right severity categories and respond accordingly.

Finding hot keywords – Opinion mining can help in discovering hot search keywords and trending topics. This feature can help the brand in their SEO (Search Engine Optimization) and social media marketing. This means that opinion mining will help brands craft their marketing and targeting strategies leveraging trending topics or search key words.

Voice of customer – Sentiment analysis of social media reviews, mentions and surveys helps broadcast the voice of customers to the brand, they are expressing their views about. This way the brand knows, exactly how the user feels about their services. The company can use this information in growing their market, advertisement targeting and building loyalty among its customers.

Employee feedback – Sentiment analysis can also be used to receive feedback from the employees of the company and analyse their emotions and attitude towards their job. And to determine whether they are satisfied with their job or not.

New product design – When a company releases a new product or service, it is released as a pilot or beta version. The monitoring of public feedback at this stage is very crucial. So, text mining from social media platforms and review sections helps brands tweak their product and make it market ready basis public opinion.

# Chapter 12

## RESULT AND SAMPLE OUTPUT

The ultimate outcome of this Training of Public reviews dataset is that, the machine is capable of judging whether an entered sentence bears positive response or negative response.

**Precision** (also called positive predictive value) is the fraction of relevant instances among the retrieved instances, while **Recall** (also known as sensitivity) is the fraction of relevant instances that have been retrieved over the total amount of relevant instances. Both precision and recall are therefore based on an understanding and measure of relevance.

$$\text{precision} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{retrieved documents}\}|}$$

$$\text{recall} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{relevant documents}\}|}$$

```
Model Accuracy:0.85
              precision    recall  f1-score   support

           0       0.85      0.85      0.85     12500
           1       0.85      0.85      0.85     12500

    accuracy                           0.85     25000
   macro avg       0.85      0.85      0.85     25000
weighted avg       0.85      0.85      0.85     25000

[[10672  1828]
 [ 1913 10587]]
Model Accuracy:0.85
```

Figure 12. Model Accuracy and Confusion Matrix for SGDC

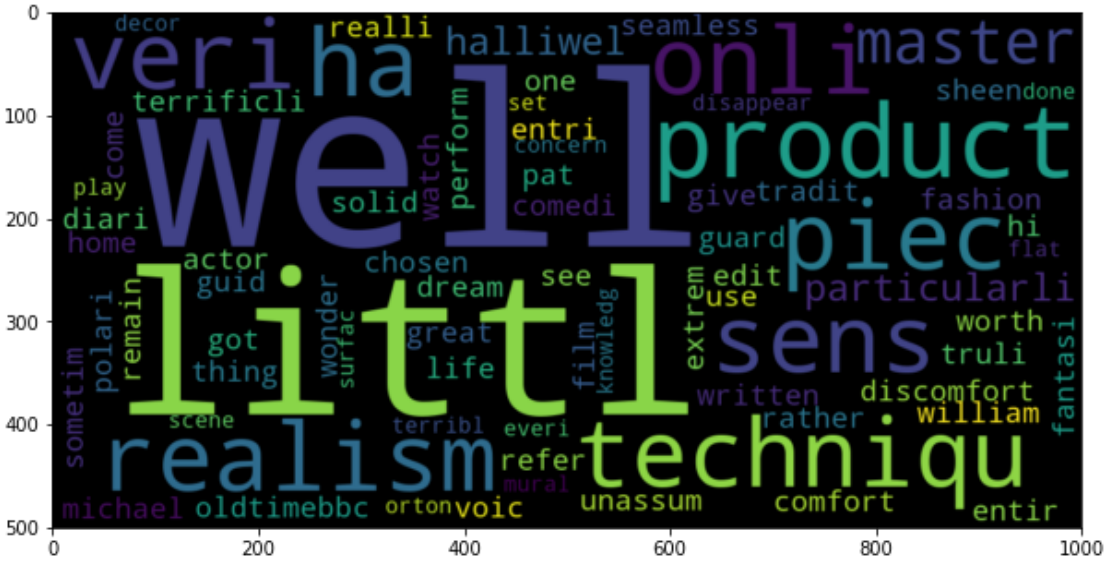


Figure 13. Word cloud for positive review words

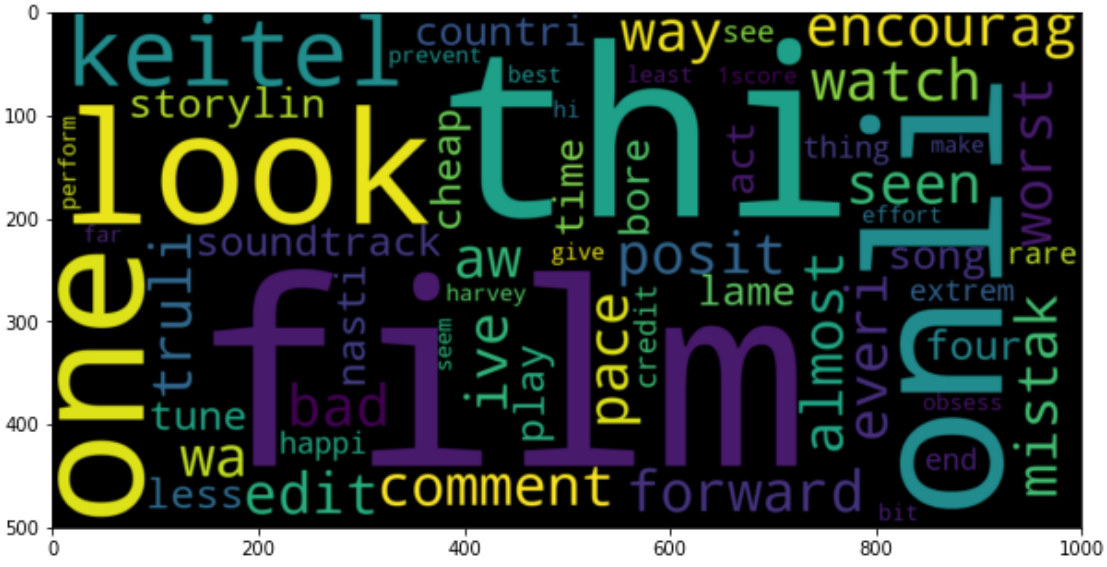


Figure 14. Word cloud for negative review words

## Confusion Matrix

A **confusion matrix** is a matrix (table) that can be used to measure the performance of an machine learning algorithm, usually a supervised learning one. Each row of the confusion matrix represents the instances of an actual class and each column represents the instances of a predicted class

The fields of the matrix mean the following:

| | | predicted | |
|---|---|---|---|
| actual | | negative | positive |
| | negative | TN<br>True positive | FP<br>False Positive |
| | positive | FN<br>False negative | TP<br>True positive |

Figure 15. Field of the confusion matrix

We can define now some important performance measures used in machine learning:

**Accuracy**:

$$AC=(TN+TP)/(TN+FP+FN+TP)$$

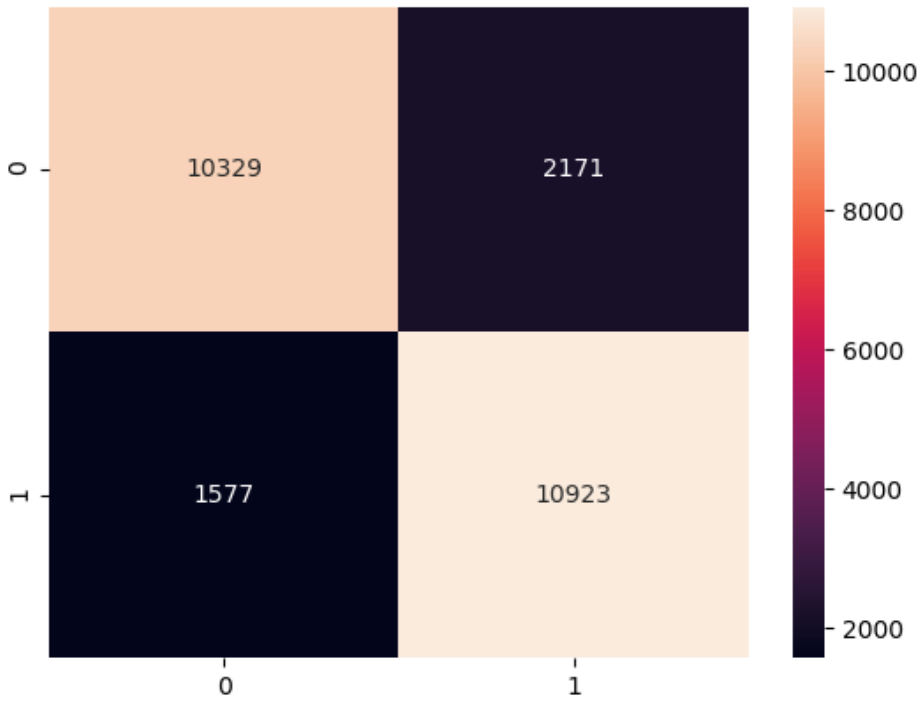


Figure 16. Confusion Matrix taken after execution of SGDC

# Chapter 12

## CONCLUSION

Text pre-processing is an important phase in all relevant applications of data mining. In Sentiment Analysis, in particular, it is cited in almost all available research works. The Implementation of the project was carried out on data of movie reviews. Sentiment Analysis and Opinion mining has become a fascinating research area due to the availability of a huge volume of user-generated content in review sites, forums and blogs. Sentiment Analysis has applications in a variety of fields ranging from market research to decision making to advertising. With the help of Sentiment Analysis, companies can estimate the extent of product acceptance and can devise strategies to improve their product.

Sentiment analysis is an emerging research area in text mining and computational linguistics, and has attracted considerable research attention in the past few years. Future research shall explore sophisticated methods for opinion and product feature extraction, as well as new classification models that can address the ordered labels property in rating inference. Applications that utilize results from sentiment analysis is also expected to emerge in the near future.

## FUTURE WORK

Try to apply different models and features to increase the accuracy score. As suggested by guide implement split of 75% and 25% dataset for train and test set respectively. Also try to create more improvement on visualisation part. Then, next will be try to catch the theme of the movie i.e. if movie is action , comedy , thriller , crime or imagination one.

## REFERENCES :

[1] Dr. Khalid N. Alhayyan & Dr. Imran Ahmad "Discovering and Analysing Important Real-Time Trends in Noisy Twitter Stream"
[2] J. Ramteke, S. Shah, D. Godhia, and A. Shaikh, "Election result prediction using Twitter sentiment analysis," in Inventive Computation Technologies (ICICT),
[3] Andrew L Mass, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng and Christopher Potts (2011). Learning Word Vectors for Sentiment Analysis
[4] Sentiment Analysis – Wikipedia – https://en.wikipedia.org/wiki/Sentiment_analysis
[5] Internet Movie Database – http://www.imdb.com/
[6] NLTK Stopwords Corpus: http://www.nltk.org/book/ch02.html
[7] Pang, Bo; Lee, Lillian; Vaithyanathan, Shivakumar (2002). "Thumbs up? Sentiment Classification using Machine Learning Techniques". Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP).
[8] Turney, Peter (2002). "Thumbs Up or Thumbs Down? Semantic Orientation Applied to Unsupervised Classification of Reviews". Proceedings of the Association for Computational Linguistics.
[9] Tumasjan, Andranik; O.Sprenger, Timm; G.Sandner, Philipp; M.Welpe, Isabell (2010). "Predicting Elections with Twitter: What 140 Characters Reveal about Political Sentiment". "Proceedings of the Fourth International AAAI Conference on Weblogs and Social Media"
[10] Spotle.ai