**SENTIMENT ANALYSIS**

Submitted in Partial fulfilment of the requirements of

the degree of

Bachelor of Engineering

By

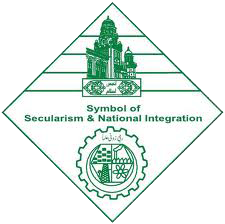
Mushrifah Hasan

Khan Anam

Maghrabi Umama

Guide:

Asst. Prof. Waheeda Dhokley



(Computer Department)

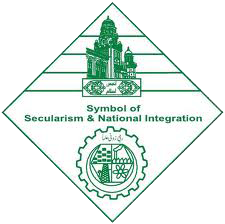
**Anjuman-i-Islam’s**

**M.H. Saboo Siddik College of Engineering**

**2018-2019**

**M.H. SABOO SIDDIK COLLEGE OF ENGINEERING**

**8, Saboo Siddik Road Byculla, Mumbai – 400 008**



This is to certify that

**Roll No.                               Name**

CS116006                             Mushrifah Hasan

                             CS116013     Khan Anam

 CS116018                         Maghrabi Umama

of Third Year (Semester VI) degree course in computer Engineering have completed the specified project report on

**“Sentiment Analysis”**

As a partial fulfillment of the project work in a satisfactory manner as per the rules of the curriculum laid by the University of Mumbai, during the Academic Year January 2019 – April 2019

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_                                                        \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

       Project Guide                                                                   External Examiner

(Prof. Waheeda Dhokley)

**ABSTRACT**

We are using a set of manually annotated texts: movie reviews from IMDB(www.imdb.com) – a world known cinema portal for sentiment analysis. This data set contains 50,000 movie reviews scrapped from IMDB website where 25,000 of them are positive and the remaining 25,000 are negative reviews. Since it is scrapped, it contains many useless things which we are removing in pre-processing. For pre-processing our data, we are using Natural Language Toolkit (NLTK). We are using ‘*WordNetLemmatizer*’ and ‘*stopwords*’ imported from NLTK for lemmatizing and removing the stopwords respectively. After this, we are storing these reviews as processed\_reviews with their respective sentiment value(0-negative/1-positive). Out of the 50,000 reviews, we are using 35,000 if them for training and the rest for testing our model. CountVectorizer which is another function from sklearn.feature\_extraction that implements both tokenization and occurrence counting in a single class. The algorithm we are using for classification of positive and negative sentiments is Support Vector Machine(SVM). SGDClassifier from scikit-learn is used for classification. Now, we predict the value and apply different metric functions to check our accuracy.Using SVM model, we are getting an accuracy of 90% on test data set. We finally deployed our model using flask api with a web page containing a textbox for us to write a review which shows us whether our review is positive or negative.

**List of figures**

Figure 3.1.1 Flowchart …………………………………………………………………………………………………………….7

Figure 3.2.2.1 Train & Test flowchart…………………………………..……………………………………………………..8

Figure 3.3.1 Possible Hyperplanes…………………………………...…………………………………………………….10

Figure 3.3.2 Hyperplanes in 2D & 3D feature space………………………………………………………………..10

Figure 3.3.3 Support Vector………………………………….…………………………………………………………………11

Figure 4.2 Graph…………………………………..……………………………………………………………………………..15

Figure 4.2.2 Support Vector………………………………….………………………………………………………………..16

Figure 4.2.3 Input Screen…………..……………………….…………………………………………………………………16

Figure 4.2.4 Result Screen………..………………………….…………………………………………………………………16

**Table of contents**

Abstract…………………………………………………………………………………………………………………………….. 1

1. Introduction………………………………………………………………………………………………………………. 4
2. Literature Review………………………………………………………………………………………………………. 4
   1. Background and related work
      1. Determining Sentiment
      2. Machine Learning for Determining Sentiment
3. Proposed system and Design implementation………………………………………………………….. 6
   1. Flowchart
   2. Sentiment Analysis Algorithms
   3. What is Support Vector Machine?
   4. Stemming and Lemmatization
   5. Text feature extraction
   6. Sentiment Analysis Use Cases & Applications
4. Results………………………………………………………………………………………………………………………. 15
5. Future scope…………………………………………………………………………………………………………….. 17
6. Conclusion……………………………………………………………………………………………………………….. 17
7. Code…………………………………………………………………………………………………………………………. 18
8. References………………………………………………………………………………………………………………… 27

**1.0 Introduction**

Sentiment analysis is the automated process of understanding an opinion about a given subject from written or spoken language. In a world where we generate 2.5 quintillion bytes of data every day, sentiment analysis has become a key tool for making sense of that data. This has allowed companies to get key insights and automate all kind of processes.

Sentiment Analysis also known as *Opinion Mining* is a field within Natural Language Processing (NLP) that builds systems that try to identify and extract opinions within text. Usually, besides identifying the opinion, these systems extract attributes of the expression e.g.:

*Polarity*: if the speaker express a *positive* or *negative* opinion,

*Subject*: the thing that is being talked about,

*Opinion holder*: the person, or entity that expresses the opinion.

Currently, sentiment analysis is a topic of great interest and development since it has many practical applications. Since publicly and privately available information over Internet is constantly growing, a large number of texts expressing opinions are available in review sites, forums, blogs, and social media.

With the help of sentiment analysis systems, this unstructured information could be automatically transformed into structured data of public opinions about products, services, brands, politics, or any topic that people can express opinions about. This data can be very useful for commercial applications like marketing analysis, public relations, product reviews, net promoter scoring, product feedback, and customer service.

**2.0 Literature review**

Unsupervised vector-based approaches to semantics can model rich lexical meanings, but they largely fail to capture sentiment information that is central to many word meanings and important for a wide range of NLP tasks.Word representations are a critical component of many natural language processing systems. It is common to represent words as indices in a vocabulary, but this fails to capture the rich relational structure of the lexicon. Vector-based models do much better in this regard. They encode continuous similarities between words as distance or angle between word vectors in a high-dimensional space. The general approach has proven useful in tasks such as word sense disambiguation, named entity recognition, part of speech tagging, and document retrieval[1][2].

**2.1 Background and related work**

Sentiment classification of reviews has been the focus of recent research. It has been attempted in different domains such as movie reviews, product reviews, and customer feedback reviews (Pang et al. 2002; Turney and Littman 2003; Pang and Lee 2004; Beineke, Hastie, and Vaithyanathan 2004; Gamon 2004). Much of the research until now has focused on training ML algorithms such as SVMs to classify reviews. Research has also been done on positive/negative term-counting methods and automatically determining if a term is positive or negative (Turney and Littman 2002).

**2.1.1 Determining Sentiment**

Research on predicting the semantic orientation of adjectives was initiated by Hatzivassiloglou and McKeown (1997). An unsupervised learning algorithm was used in Turney (2002) and Turney and Littman (2003) to determine the semantic orientation of individual terms. The algorithm started with seven known positive terms and seven known negative terms. The algorithm took a search term and used *AltaVista’s* NEAR operator to find how many documents have the search term near the seven positive terms and the seven 112 COMPUTATIONAL INTELLIGENCE negative terms. The difference in pointwise mutual information (PMI) score with the two sets was then used to determine the semantic orientation from pointwise mutual information (SO-PMI) score, which gives the degree to which each term is positive or negative (Turney and Littman 2002). The PMI score of two words w1 and w2 is given by the probability of the two words occurring together divided by the probabilities of each word in part:

The formula for the semantic orientation of a word can be expressed as:

,

where the positive and negative reference terms are

*pquery* = *good* OR *nice* OR *excellent* OR *positive* OR *fortunate* OR *correct* OR *superior*

*nquery* = *bad* OR *nasty* OR *poor* OR *negative* OR *unfortunate* OR *wrong* OR *inferior.*

OR and NEAR are operators offered by the AltaVista search engine (NEAR is no longer supported). By approximating the PMI values using number of hits returned by the search engine and ignoring the number of documents in the corpus (N), the formula becomes

The semantic orientation of bigrams can also be determined (Turney 2002). The semantic orientation of terms and phrases can be used to determine the sentiment of complete sentences and reviews. Four hundred ten reviews from epinions.com were taken and the accuracy of classifying the documents was found when computing the sentiment of phrases for different kinds of reviews. Results ranged from 84% for automobile reviews to as low as 66% for movie reviews (Turney 2002).[4]

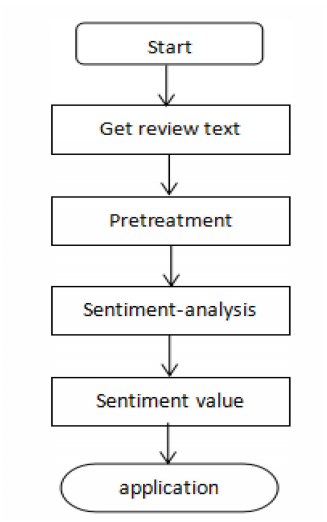
**2.1.2. Machine Learning for Determining Sentiment**

One of the most common methods of classifying documents into positive and negative terms is to train an ML algorithm to classify the documents. Several ML algorithms are compared in Pang et al. (2002) and Pang and Lee (2004), where it was found that SVMs generally gave better results than other classifiers. Unigrams, bigrams, part of speech information, and the position of the terms in the text were used as features; however, using only unigrams was found to give the best results. Bayesian belief networks have also been used to determine the sentiment of a document (Bai, Padman, and Airoldi 2004).

Sentiment classification has also been done on customer feedback reviews (Gamon 2004). A variety of features were used in SVMs in an attempt to divide the data set not only into positive and negative, but to give rankings of 1, 2, 3, or 4, where 1 means “not satisfied” and 4 means “very satisfied.” The proposed system was fairly good at distinguishing classes 1 and 4, with about 78% accuracy. Separating classes 1 and 2 from 3 and 4 was more difficult and was only 69% accurate. These results were achieved when using the top 2,000 features selected by log-likelihood ratios[4].

**3.0 Proposed system and design implementation**

**3.1 Flowchart**



*Fig 3.1.1:* Flowchart

**3.2 Sentiment Analysis Algorithms**

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

1. Rule-based systems that perform sentiment analysis based on a set of manually crafted rules.
2. Automatic systems that rely on machine learning techniques to learn from data.
3. Hybrid systems that combine both rule based and automatic approaches.

**3.2.1. Rule-based Approaches**

Usually, rule-based approaches define a set of rules in some kind of scripting language that identify subjectivity, polarity, or the subject of an opinion.The rules may use a variety of inputs, such as the following:

Classic NLP techniques like *stemming*, *tokenization*, *part of speech tagging* and *parsing*. Other resources, such as lexicons (i.e. lists of words and expressions).

A basic example of a rule-based implementation would be the following:

Define two lists of polarized words (e.g. negative words such as *bad*, *worst*, *ugly*, etc and positive words such as *good*, *best*, *beautiful*, etc).

Given a text:

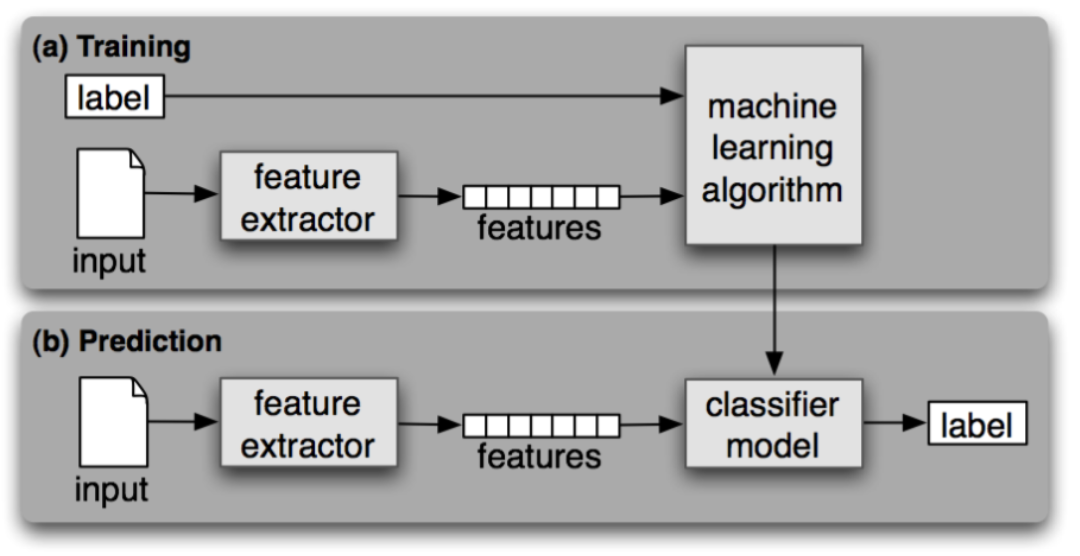
* Count the number of positive words that appear in the text.
* Count the number of negative words that appear in the text.
* If the number of positive word appearances is greater than the number of negative word appearances return a positive sentiment, conversely, return a negative sentiment. Otherwise, return neutral.

This system is very naïve since it doesn't take into account how words are combined in a sequence. A more advanced processing can be made, but these systems get very complex quickly. They can be very hard to maintain as new rules may be needed to add support for new expressions and vocabulary. Besides, adding new rules may have undesired outcomes as a result of the interaction with previous rules. As a result, these systems require important investments in manually tuning and maintaining the rules.

**3.2.2. Automatic Approaches**

Automatic methods, contrary to rule-based systems, don't rely on manually crafted rules, but on machine learning techniques. The sentiment analysis task is usually modeled as a classification problem where a classifier is fed with a text and returns the corresponding category, e.g. positive, negative, or neutral (in case polarity analysis is being performed).

Said machine learning classifier can usually be implemented with the following steps and components:



*fig 3.2.2.1:* Train & Test Flowchart

**a.) The Training and Prediction Processes**

In the training process (a), our model learns to associate a particular input (i.e. a text) to the corresponding output (tag) based on the test samples used for training. The feature extractor transfers the text input into a feature vector. Pairs of feature vectors and tags (e.g. *positive*, *negative*, or *neutral*) are fed into the machine learning algorithm to generate a model.

In the prediction process (b), the feature extractor is used to transform unseen text inputs into feature vectors. These feature vectors are then fed into the model, which generates predicted tags (again, *positive*, *negative*, or *neutral*).

**b.) Feature Extraction from Text**

The first step in a machine learning text classifier is to transform the text into a numerical representation, usually a vector. Usually, each component of the vector represents the frequency of a word or expression in a predefined dictionary (e.g. a lexicon of polarized words). This process is known as feature extraction or text vectorization and the classical approach has been bag-of-words or [bag-of-ngrams](https://www.quora.com/What-is-the-difference-between-bag-of-words-and-bag-of-n-grams) with their frequency.

More recently, new feature extraction techniques have been applied based on word embeddings (also known as *word vectors*). This kind of representations makes it possible for words with similar meaning to have a similar representation, which can improve the performance of classifiers.

**c.) Classification Algorithms**

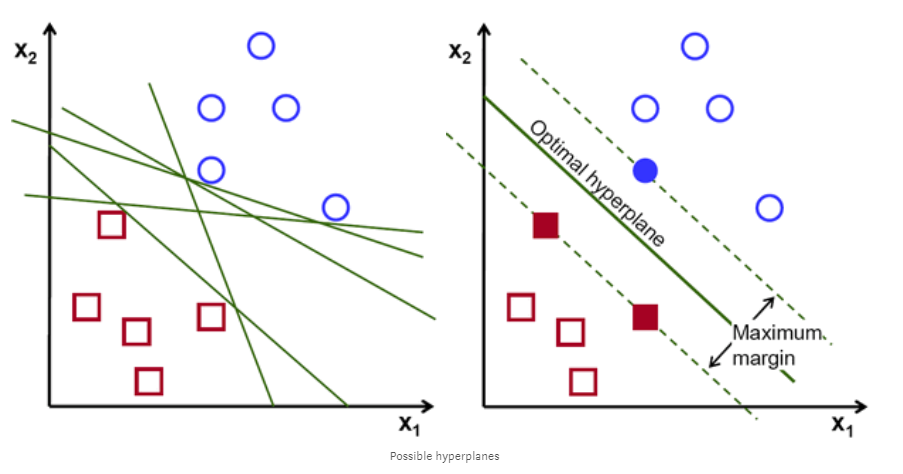
The classification step usually involves a statistical model like Naïve Bayes, Logistic Regression, Support Vector Machines, or Neural Networks:

1. Naïve Bayes: a family of probabilistic algorithms that uses Bayes Theorem to predict the category of a text.
2. Linear Regression: a very well-known algorithm in statistics used to predict some value (Y) given a set of features (X).
3. Support Vector Machines: a non-probabilistic model which uses a representation of text examples as points in a multidimensional space. These examples are mapped so that the examples of the different categories (sentiments) belong to distinct regions of that space.. Then, new texts are mapped onto that same space and predicted to belong to a category based on which region they fall into.

If you or your company have not used sentiment analysis before, then you’ll see some improvement really quickly. For typical use cases, such as ticket routing, brand monitoring, and VoC analysis (see below), this means you will save a lot of time and money -which you are likely to be investing in in-house manual work nowadays,- save your teams some frustration, and increase your (or your company’s) productivity.

**3.3 What is Support Vector Machine?**

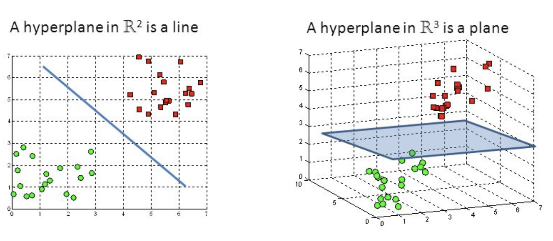
The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space(N — the number of features) that distinctly classifies the data points.



*Fig 3.3.1:* Possible hyperplanes

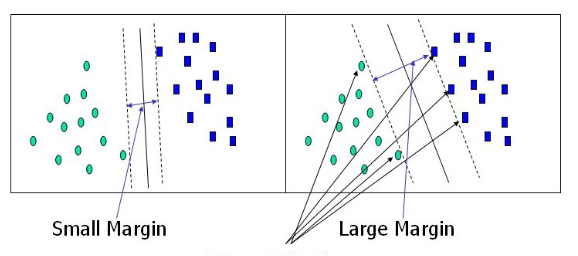
To separate the two classes of data points, there are many possible hyperplanes that could be chosen. Our objective is to find a plane that has the maximum margin, i.e the maximum distance between data points of both classes. Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence.

**Hyperplanes and Support Vectors**



*Fig 3.3.2:* Hyperplanes in 2D and 3D feature space

Hyperplanes are decision boundaries that help classify the data points. Data points falling on either side of the hyperplane can be attributed to different classes. Also, the dimension of the hyperplane depends upon the number of features. If the number of input features is 2, then the hyperplane is just a line. If the number of input features is 3, then the hyperplane becomes a two-dimensional plane. It becomes difficult to imagine when the number of features exceeds 3.



*Fig 3.3.3*: Support Vectors

Support vectors are data points that are closer to the hyperplane and influence the position and orientation of the hyperplane. Using these support vectors, we maximize the margin of the classifier. Deleting the support vectors will change the position of the hyperplane. These are the points that help us build our SVM.

**3.4 Stemming and Lemmatization**

Stemming and Lemmatization are **Text Normalization** (or sometimes called **Word Normalization**) techniques in the field of **Natural Language Processing** that are used to prepare text, words, and documents for further processing. Stemming and Lemmatization have been studied, and algorithms have been developed in Computer Science since the 1960's. In this tutorial you will learn about Stemming and Lemmatization in a practical approach covering the background, some famous algorithms, applications of Stemming and Lemmatization, and how to stem and lemmatize words, sentences and documents using the Python nltk package which is the Natural Language Tool Kit package provided by Python for Natural Language Processing tasks.

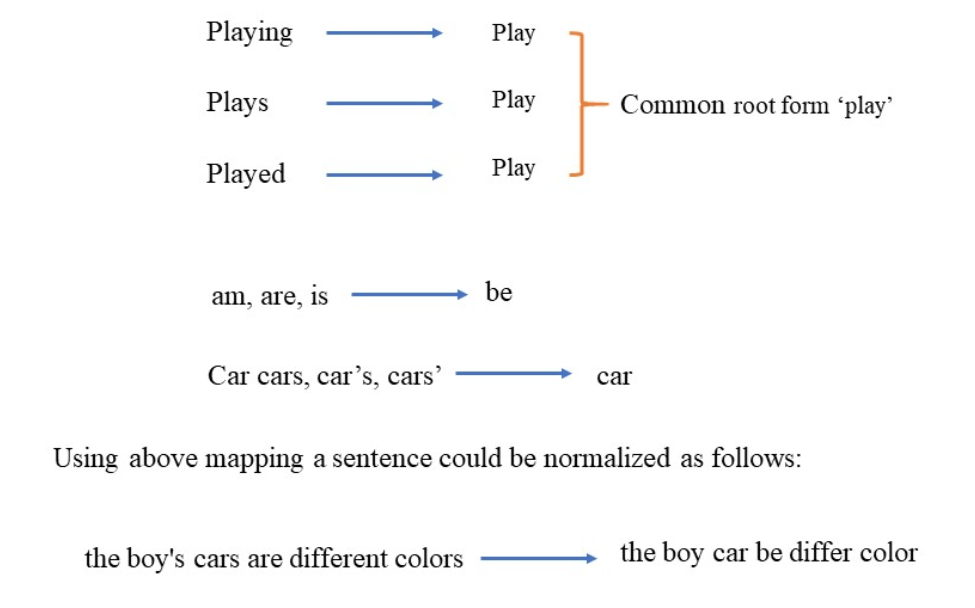
1. Background
2. Stemming with Python nltk package
   * What is Python nltk Package?
3. Lemmatization with Python nltk package
4. Applications of Stemming and Lemmatization

**3.4.1 Background**

Languages we speak and write are made up of several words often derived from one another. When a language contains words that are derived from another word as their use in the speech changes is called **Inflected Language**.

*"In grammar, inflection is the modification of a word to express different grammatical categories such as tense, case, voice, aspect, person, number, gender, and mood. An inflection expresses one or more grammatical categories with a****prefix, suffix or infix****, or another internal modification such as a vowel change" [Wikipedia]*

The degree of inflection may be higher or lower in a language. As you have read the definition of inflection with respect to grammar, you can understand that an inflected word(s) will have a common root form. Let's look at a few examples,



Above examples must have helped you understand the concept of normalization of text, although normalization of text is not restricted to only written document but to speech as well. Stemming and Lemmatization helps us to achieve the root forms (sometimes called synonyms in search context) of inflected (derived) words. *Stemming is different to Lemmatization in the approach it uses to produce root forms of words and the word produced.*

Stemming and Lemmatization are widely used in **tagging systems, indexing, SEOs, Web search results, and information retrieval**. For example, searching for *fish* on Google will also result in *fishes*, *fishing* as *fish* is the stem of both words. Later in this tutorial, you will go through some of the significant uses of Stemming and Lemmatization in applications.

**3.4.2 Stemming with Python nltk package**

*"Stemming is the process of reducing inflection in words to their root forms such as mapping a group of words to the same stem even if the stem itself is not a valid word in the Language."*

Stem (root) is the part of the word to which you add inflectional (changing/deriving) affixes such as (-ed,-ize, -s,-de,mis). So stemming a word or sentence may result in words that are not actual words. Stems are created by removing the suffixes or prefixes used with a word.

*Information: Removing suffixes from a word is called Suffix Stripping*

* **What is Python nltk package?**

Natural Language Toolkit (NLTK) is a Python library to make programs that work with natural language. It provides a user-friendly interface to datasets that are over 50 corpora and lexical resources such as WordNet Word repository. The library can perform different operations such as tokenizing, stemming, classification, parsing, tagging, and semantic reasoning.

You can also tell the stemmer to ignorestop-words.

***Stop words:*** *Stop Words are words which do not contain important significance to be used in Search Queries. Usually, these words are filtered out from search queries because they return a vast amount of unnecessary information. Each programming language will give its own list of stop words to use. Mostly they are words that are commonly used in the English language such as 'as, the, be, are' etc.*

**3.4.3 Lemmatization with Python nltk package**

Lemmatization, unlike Stemming, reduces the inflected words properly ensuring that the root word belongs to the language. In Lemmatization root word is called **Lemma**. A lemma (plural lemmas or lemmata) is the canonical form, dictionary form, or citation form of a set of words.

For example, runs, running, ran are all forms of the word run, therefore run is the lemma of all these words. Because lemmatization returns an actual word of the language, it is used where it is Python NLTK provides **WordNet Lemmatizer** that uses the WordNet Database to lookup lemmas of words necessary to get valid words.

**3.4.4 Applications of Stemming and Lemmatization**

Stemming and Lemmatization are itself form of NLP and widely used in Text mining. Text Mining is the process of analysis of texts written in natural language and extract high-quality information from text. It involves looking for interesting patterns in the text or to extract data from the text to be inserted into a database. Text mining tasks include *text categorization, text clustering, concept/entity extraction, production of granular taxonomies, sentiment analysis, document summarization, and entity relation modeling (i.e., learning relations between named entities).* Developers have to prepare text using lexical analysis, POS (Parts-of-speech) tagging, stemming and other Natural Language Processing techniques to gain useful information from text.

**3.5 Text feature extraction**

**The Bag of Words representation**

Text Analysis is a major application field for machine learning algorithms. However the raw data, a sequence of symbols cannot be fed directly to the algorithms themselves as most of them expect numerical feature vectors with a fixed size rather than the raw text documents with variable length.

In order to address this, scikit-learn provides utilities for the most common ways to extract numerical features from text content, namely:

* **tokenizing** strings and giving an integer id for each possible token, for instance by using white-spaces and punctuation as token separators.
* **counting** the occurrences of tokens in each document.
* **normalizing** and weighting with diminishing importance tokens that occur in the majority of samples / documents.

In this scheme, features and samples are defined as follows:

* each individual token occurrence frequency (normalized or not) is treated as a feature.
* the vector of all the token frequencies for a given **document** is considered a multivariate sample.

A corpus of documents can thus be represented by a matrix with one row per document and one column per token (e.g. word) occurring in the corpus.

We call **vectorization** the general process of turning a collection of text documents into numerical feature vectors. This specific strategy (tokenization, counting and normalization) is called the **Bag of Words** or “Bag of n-grams” representation. Documents are described by word occurrences while completely ignoring the relative position information of the words in the document

[**CountVectorizer**](https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html#sklearn.feature_extraction.text.CountVectorizer) implements both tokenization and occurrence counting in a single class:

>>> from sklearn.feature\_extraction.text import CountVectorizer

**3.6 Sentiment Analysis Use Cases & Applications**

In this section, we’ll take a dive into real life use cases, applications, and examples of the impact of all this can have on businesses, cities, and society – sentiment analysis in the wild, if you will.

Specifically, we’ll examine the use of sentiment analysis in the following:

* Social media monitoring
* Brand monitoring
* Voice of customer (VoC)
* Customer service
* Workforce analytics and voice of employee
* Product analytics
* Market research and analysis

**3.6.1 Sentiment Analysis in Social Media Monitoring**

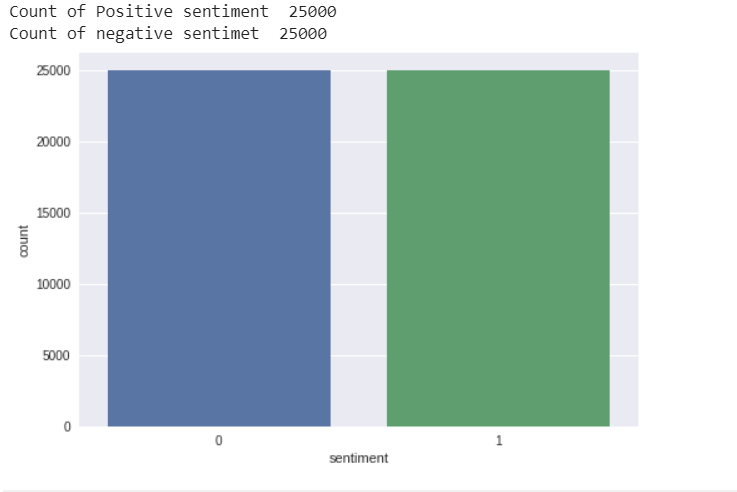
On the fateful evening of April 9th, 2017, United Airlines forcibly removed a passenger from an overbooked flight. The nightmare-ish incident was filmed by other passengers on their smartphones and posted immediately. One such video, posted to Facebook, was shared more than 87,000 times and viewed 6.8 million times by 6pm on Monday, just 24 hours later.

The fiasco was magnified horrifically by the company’s dismissive response. On Monday afternoon, they tweeted a statement from the CEO apologizing for “having to re-accommodate customers.” Cue public outrage –you can imagine the field day on Twitter.

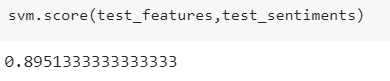
This is exactly the kind of PR catastrophe we’d all like to do happily without. This is also an excellent example of why we care not only about *if* people are talking about our brand, but *how* they’re talking about it. More mentions does *not* equal positive mentions.

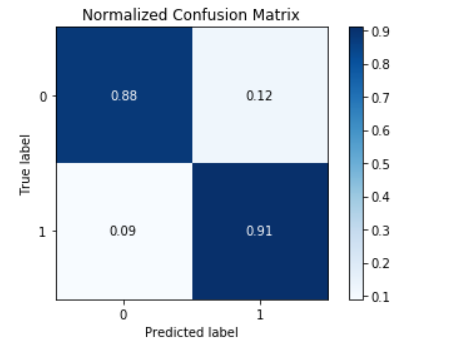
In today’s day and age, brands of all shapes and sizes have meaningful interactions with customers, leads, and even competition on social networks like Facebook, Twitter, and Instagram. Most marketing departments are already tuned into to online mentions as far as *volume* –they measure more chatter as more brand awareness. Nowadays, however, we can take a step deeper. By using sentiment analysis on social media, we can get incredible insights into the *quality* of conversation that’s happening around a brand.

**4.0 Results**

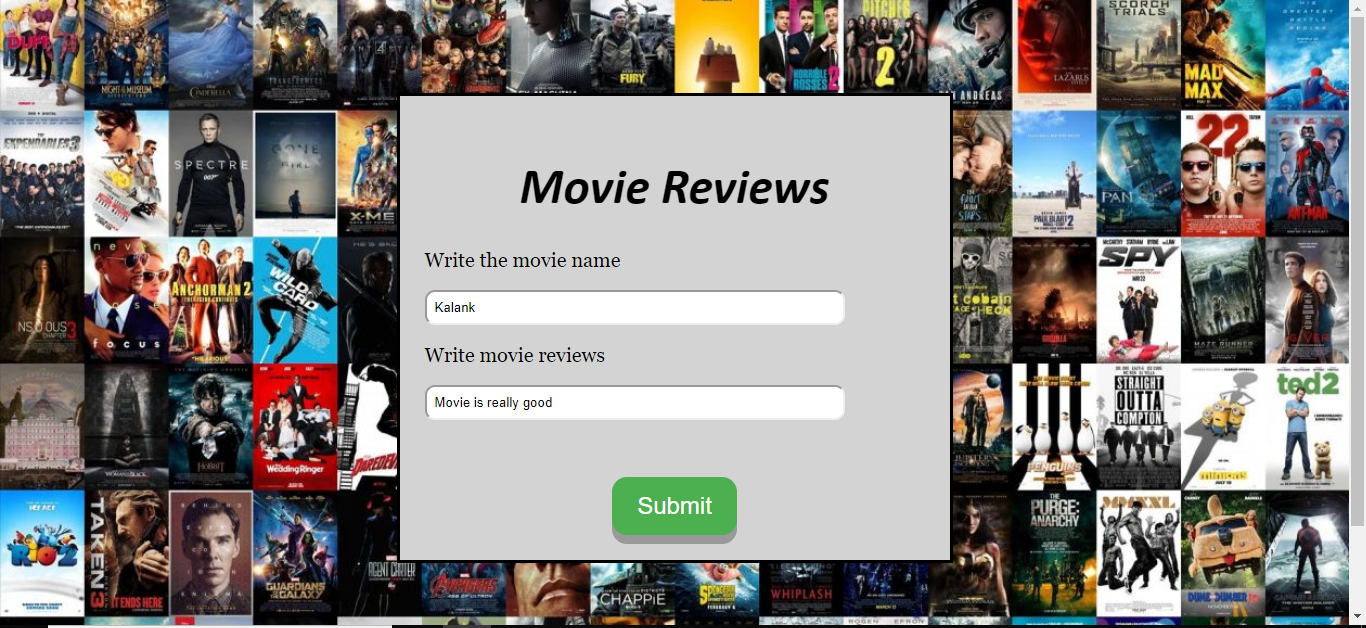
****

*fig 4.1: Graph*

****

****

*fig 4.2.2:* Confusion Matrix



*fig 4.2.3*: Input Screen



*fig 4.2.4*: Result screen

**5.0 Future scope**

To extend our model we aim for it to detect even neutral sentiments instead of just positive and negative sentiment given as our dataset consists of only these two sentiments i.e. positive and negative sentiment. Also to further improve the accuracy of model we would love to use neural networks i.e. RNN or LSTM for further better prediction results. Also we can use our model as an api which can be used by “dialog flow” and convert it into a chatbot application which can be used as a facebook messenger chatbot as a service and to keep the server up and running we will use heroku.

 **6.0 Conclusion**

The Sentiment analysis is a extraction of desired information from source material. We had done the sentiment analysis of movie reviews to understand the reaction of audience. This will help to improve the mistake and performance in future. We implement the different algorithm to achieve our goal. The naive bayes algorithm gave us the accuracy of about 73% which was very less so we moved to SVM. SVM SGD classifier deals with text analysis very effectively, so we get the accuracy of our model 90%.

**7.0 Code**

**1. Extractor.py file**

# -\*- coding: utf-8 -\*-

"""

Created on Mon Mar 25 21:14:25 2019

@author: mushrifah

"""

import os

import pandas as pd

import numpy as np

labels = {'pos': 'positive', 'neg': 'negative'}

dataset = pd.DataFrame()

for directory in ('test', 'train'):

for sentiment in ('pos', 'neg'):

path =r'F:/aclImdb/{}/{}'.format(directory, sentiment)

for review\_file in os.listdir(path):

with open(os.path.join(path, review\_file), 'r',encoding="utf8") as input\_file:

review = input\_file.read()

dataset = dataset.append([[review, labels[sentiment]]],

ignore\_index=True)

dataset.columns = ['review', 'sentiment']

indices = dataset.index.tolist()

np.random.shuffle(indices)

indices = np.array(indices)

dataset = dataset.reindex(index=indices)

dataset.to\_csv('movie\_reviews.csv', index=False,encoding="utf8")

**2. Svm model.ipynb file**

# -\*- coding: utf-8 -\*-

"""svm model for imdb reviews.ipynb

Automatically generated by Colaboratory.

Original file is located at

   https://colab.research.google.com/drive/0B3nlP16Z83t8TXoycTVScFNaV251RmpjOWtMYlJHTnJQX1NB

"""

!pip install -U -q PyDrive

from pydrive.auth import GoogleAuth

from pydrive.drive import GoogleDrive

from google.colab import auth

from oauth2client.client import GoogleCredentials

import io

import zipfile

# Authenticate and create the PyDrive client.

# This only needs to be done once per notebook.

auth.authenticate\_user()

gauth = GoogleAuth()

gauth.credentials = GoogleCredentials.get\_application\_default()

drive = GoogleDrive(gauth)

# Download a file based on its file ID.

#https://drive.google.com/open?id=1TgsnApIgUGzC1pd9RYkCFoSP0yFEuXVJ

file\_id = '1TgsnApIgUGzC1pd9RYkCFoSP0yFEuXVJ' #-- Updated File ID for my zip

downloaded = drive.CreateFile({'id': file\_id})

downloaded.GetContentFile('movie\_reviews.csv')

import pandas as pd

df=pd.read\_csv("movie\_reviews.csv",encoding="utf8")

df.head()

df = df[df.sentiment != 'unsup']

df['sentiment'] = df['sentiment'].map({'positive': 1, 'negative': 0})

df.head()

import re

import nltk

nltk.download('stopwords')

nltk.download('wordnet')

from nltk.stem import WordNetLemmatizer

from nltk.corpus import stopwords

stop\_words = set(stopwords.words("english"))

lemmatizer = WordNetLemmatizer()

def clean\_text(text):

   text = re.sub(r'[^\w\s]','',text, re.UNICODE)

   text = text.lower()

   text = [lemmatizer.lemmatize(token) for token in text.split(" ")]

   text = [lemmatizer.lemmatize(token, "v") for token in text]

   text = [word for word in text if not word in stop\_words]

   text = " ".join(text)

   return text

df['Processed\_Reviews'] = df.review.apply(lambda x: clean\_text(x))

df['Processed\_Reviews'][0]

df['review'][0]

df.head()

df = df.drop(['review'],axis=1)

df.head()

df['Processed\_Reviews'][9317]

trd = df[:35000]

ted = df[35000:]

trd

ted

import numpy as np

train\_reviews = np.array(trd['Processed\_Reviews'])

train\_sentiments = np.array(trd['sentiment'])

test\_reviews = np.array(ted['Processed\_Reviews'])

test\_sentiments = np.array(ted['sentiment'])

test\_reviews[5817]

sample\_docs = [100, 5817, 7626, 7356, 1008, 7155, 3533, 13010]

sample\_data = [(test\_reviews[index],

               test\_sentiments[index])

                 for index in sample\_docs]

sample\_data

from sklearn.feature\_extraction.text import CountVectorizer, TfidfVectorizer

def build\_feature\_matrix(documents, feature\_type='frequency',

                        ngram\_range=(1, 1), min\_df=0.0, max\_df=1.0):

   feature\_type = feature\_type.lower().strip()

   if feature\_type == 'binary':

       vectorizer = CountVectorizer(binary=True, min\_df=min\_df,

                                    max\_df=max\_df, ngram\_range=ngram\_range)

   elif feature\_type == 'frequency':

       vectorizer = CountVectorizer(binary=False, min\_df=min\_df,

                                    max\_df=max\_df, ngram\_range=ngram\_range)

   elif feature\_type == 'tfidf':

       vectorizer = TfidfVectorizer(min\_df=min\_df, max\_df=max\_df,

                                    ngram\_range=ngram\_range)

   else:

       raise Exception("Wrong feature type entered. Possible values: 'binary', 'frequency', 'tfidf'")

   feature\_matrix = vectorizer.fit\_transform(documents).astype(float)

   return vectorizer, feature\_matrix

vectorizer=TfidfVectorizer(ngram\_range=(1, 1), min\_df=0.0, max\_df=1.0)

vectorizer, train\_features = build\_feature\_matrix(documents=train\_reviews,

                                                 feature\_type='tfidf',

                                                 ngram\_range=(1, 1),

                                                 min\_df=0.0, max\_df=1.0)

from sklearn.linear\_model import SGDClassifier

# build the model

svm = SGDClassifier(loss='hinge', n\_iter=500)

svm.fit(train\_features, train\_sentiments)

test\_features = vectorizer.transform(test\_reviews)

for doc\_index in sample\_docs:

   print ('Review:-')

   print (test\_reviews[doc\_index])

   print ('Actual Labeled Sentiment:', test\_sentiments[doc\_index])

   doc\_features = test\_features[doc\_index]

   predicted\_sentiment = svm.predict(doc\_features)[0]

   print ('Predicted Sentiment:', predicted\_sentiment)

   print("")

svm.score(test\_features,test\_sentiments)

predicted\_sentiments = svm.predict(test\_features)

predicted\_sentiments

from sklearn import metrics

def display\_evaluation\_metrics(true\_labels, predicted\_labels, positive\_class=1):

   print ('Accuracy:', np.round(

                       metrics.accuracy\_score(true\_labels,

                                              predicted\_labels),

                       2))

   print ('Precision:', np.round(

                       metrics.precision\_score(true\_labels,

                                              predicted\_labels,

                                              pos\_label=positive\_class,

                                              average='binary'),

                       2))

   print ('Recall:', np.round(

                       metrics.recall\_score(true\_labels,

                                              predicted\_labels,

                                              pos\_label=positive\_class,

                                              average='binary'),

                       2))

   print ('F1 Score:', np.round(

                       metrics.f1\_score(true\_labels,

                                              predicted\_labels,

                                              pos\_label=positive\_class,

                                              average='binary'),

                       2))

def display\_confusion\_matrix(true\_labels, predicted\_labels, classes=[1,0]):

   cm = metrics.confusion\_matrix(y\_true=true\_labels,

                                 y\_pred=predicted\_labels,

                                 labels=classes)

   cm\_frame = pd.DataFrame(data=cm,

                           columns=pd.MultiIndex(levels=[['Predicted:'], classes],

                                                 labels=[[0,0],[0,1]]),

                           index=pd.MultiIndex(levels=[['Actual:'], classes],

                                               labels=[[0,0],[0,1]]))

   print( cm\_frame)

def display\_classification\_report(true\_labels, predicted\_labels, classes=[1,0]):

   report = metrics.classification\_report(y\_true=true\_labels,

                                          y\_pred=predicted\_labels,

                                          labels=classes)

   print (report)

display\_evaluation\_metrics(true\_labels=test\_sentiments,

                          predicted\_labels=predicted\_sentiments,

                          positive\_class=1)

display\_confusion\_matrix(true\_labels=test\_sentiments,

                        predicted\_labels=predicted\_sentiments,

                        classes=[1, 0])

display\_classification\_report(true\_labels=test\_sentiments,

                             predicted\_labels=predicted\_sentiments,

                             classes=[1, 0])

import pickle

svm\_pkl\_filename='svm\_classifier.pkl'

svm\_pkl=open(svm\_pkl\_filename,'wb')

pickle.dump(svm,svm\_pkl)

svm\_pkl.close()

import pickle as pb

with open('vectorizer.pkl','wb') as f:

   pb.dump(vectorizer,f)

#load the saved model

svm\_pkl=open('svm\_classifier.pkl','rb')

svm=pickle.load(svm\_pkl)

print("Loaded the model ", svm)

vectorizer\_pkl=open('vectorizer.pkl','rb')

vectorizer=pickle.load(vectorizer\_pkl)

print("Loaded the model ", vectorizer)

#test the model with users data

t=["i am bad"]

a=svm.predict(vectorizer.transform(t))

print(a)

**3. App.py file**

from flask import Flask, render\_template, jsonify, request

from sklearn.externals import joblib

import traceback

import pandas as pd

import numpy as np

from sklearn.linear\_model import SGDClassifier

from sklearn.feature\_extraction.text import CountVectorizer, TfidfVectorizer

global label\_dictionary,svm,vectorizer

label\_dictionary = {0: 'negative review', 1: 'positive review'}

app = Flask(\_\_name\_\_,)

@app.route("/")

def index():

   return render\_template('form.html')

@app.route("/predict", methods=["POST"])

def predict():

   try:

           # json\_ = request.json

       query=[request.form['Reviews']]

       print(query)

       prediction = (svm.predict(vectorizer.transform(query)))

       print("#################################")

       print(prediction)

       return jsonify({'prediction': str(prediction),'label':label\_dictionary[int(prediction)]})

   except:

       return jsonify({'trace': traceback.format\_exc()})

if \_\_name\_\_ == '\_\_main\_\_':

   try:

       port = int(sys.argv[1])

   except:

       port = 12345

   svm = joblib.load("svm\_classifier.pkl") # Load "model.pkl"

   print ('Model loaded')

   vectorizer=joblib.load("vectorizer.pkl") # Load "vectorizer.pkl"

   app.run(port=port, debug=False)

**4. Form.html file**

<!DOCTYPE html>

<html>

<head>

  <title>Info</title>

 <!-- <link rel="stylesheet" type="text/css" href="style1.css">-->

  <style>

.button {

 padding: 15px 25px;

 font-size: 24px;

 text-align: center;

 cursor: pointer;

 outline: none;

 color: #fff;

 background-color: #4CAF50;

 border: none;

 border-radius: 15px;

 box-shadow: 0 9px #999;

}

.button:hover {background-color: #3e8e41}

.button:active {

 background-color: #3e8e41;

 box-shadow: 0 5px #666;

 transform: translateY(4px);

}

form{

 display: block;

 margin: 7% auto 5%;

 border: 3px solid black;

 background-position: center;

 background: lightgrey;

 width: 500px;

 font-family: calibri;

 padding: 25px;

}

body{

 font-family: calibri;

 background-image: url({{ url\_for('static', filename="Movie.jpg") }});

 /\*background-color: #cccccc;\*/

 /\*background:transparent;\*/

 height:100vh;

 background-size: cover;

 background-position: center;

}

h1{

 text-align: center;

 font-size: 50px;

 font-style: oblique;

}

label{

 font-size: 20px;

 font-family:Georgia;

}

input[type=text]{

 background-color: white;

 width: 400px;

 border-radius: 10px;

 max-height: 30px;

 padding: 8px;

}

</style>

</head>

<body>

<!-- <img src="../static/Movie.jpg"> -->

<form action="/predict" method="POST">

<div id="form">

<h1><center>Movie Reviews</center></h1>

  <label class="label" for="Reviews">Write movie reviews</label><br><br>

  <div id="text">

<input type="text" name="Reviews" id="Reviews"/><br><br>

 </div>

 <br><br>

 <center><div id="button">

  <button class="button">Submit</button>

 </div></center>

</div>

</form>

</body>

</html>

1. **References**

[1] Turney and Pantel, 2010; Collobert and Weston, 2008; Turian et al., 2010)

[2] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts Stanford University Stanford, CA 94305 [amaas, rdaly, ptpham, yuze, ang, cgpotts]@stanford.edu

[3] Dual Sentiment Analysis: Considering Two Sides of One Review: Rui Xia, Feng Xu, Chengqing Zong, Qianmu Li, Yong Qi, and Tao Li

[4] SENTIMENT CLASSIFICATION OF MOVIE REVIEWS USING CONTEXTUAL VALENCE SHIFTERS

ALISTAIR KENNEDY AND DIANA INKPEN

School of Information Technology and Engineering, University of Ottawa, Ottawa, ON, K1N 6N5, Canada

**Links**

* Data set: <https://ai.stanford.edu/~amaas/data/sentiment/>
* [https://monkeylearn.com/sentiment-analysis/](https://www.linkedin.com/pulse/future-sentiment-analysis-shahbaz-anwar)
* <https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html>
* <https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47>
* <https://scikit-learn.org/stable/modules/feature_extraction.html>

## 

## 