

MACHINE LEARNING

Movie Review Classification

Summer Internship Report Submitted in partial fulfillment of the requirement for

undergraduate degree of

Bachelor of Technology

In

Computer Science and Engineering

By

Yashaswini.Y

3210316852

Under the Guidance of

Assistant Professor



Department Of Computer Science Engineering

GITAM School of Technology

GITAM (Deemed to be University) Hyderabad-502329

July 2020

DECLARATION

I submit this industrial training work entitled “**Movie Review Classification**” to GITAM (Deemed To Be University), Hyderabad in partial fulfillment of the requirements for the award of the degree of “**Bachelor of Technology**” in “**Computer Science and Engineering**”. I declare that it was carried out independently by me under the guidance of **Mr. , Asst. Professor**, GITAM (Deemed To Be University), Hyderabad, India.

The results embodied in this report have not been submitted to any other University or Institute for the award of any degree or diploma.

Place: HYDERABAD

Yashaswini.Y

Date: 12-7-2020

3210316852



GITAM (DEEMED TO BE UNIVERSITY)

Hyderabad-502329, India

Dated: 12-7-2020

CERTIFICATE

This is to certify that the Industrial Training Report entitled “**MOVIE REVIEW CALSSIFICATION**” is being submitted by Yashaswini.Y (3210316852) in partial fulfillment of the requirement for the award of **Bachelor of Technology in Computer Science and Engineering** at GITAM (Deemed to Be University), Hyderabad during the academic year 2020-2021

It is faithful record work carried out by her at the **Computer Science and Engineering Department**, GITAM University Hyderabad Campus under my guidance and supervision.

Assistant Professor

Department of CSE

Professor and HOD

S. Phani Kumar

Department of CSE

ACKNOWLEDGEMENT

My project would not have been successful without the help of several people, I would like to thank the personalities who were part of my project in numerous ways, those who gave me outstanding support from the birth of the project.

I would like to thank our honorable Pro-Vice-Chancellor, **Prof. N. Siva Prasad** for providing necessary infrastructure and resources for the accomplishment of my project.

I would like to thank respected **Prof. N. Seetharamaiah**, Principal, School of Technology, for his support during the tenure of the project.

I would like to thank respected **Prof. S. Phani Kumar**, Head of the Department of Computer Science & Engineering for providing the opportunity to undertake this project and encouragement in the completion of this project.

I would like to thank respected **Mr.** , Assistant Professor in Computer Science Department for the esteemed guidance, moral support and invaluable advice provided by him for the success of the project.

I would like to thank my parents and friends who extended their help, encouragement and moral support either directly or indirectly in my project work.

Sincerely,

Yashaswini.Y

3210316852

Abstract

As humans' opinions help enhance products efficiency, and since the success or the failure of a movie depends on its reviews, there is an increase in the demand and need to build a good sentiment analysis model that classifies movies reviews. In this research, tokenization is employed to transfer the input string into a word vector, stemming is utilized to extract the root of the words, feature selection is conducted to extract the essential words, and finally classification is performed to label reviews as being either positive or negative.

A model that makes use of all of the previously mentioned methods is presented. The model is evaluated and compared on eight different classifiers. The model is evaluated on a real world dataset. In order to compare the eight different classifiers, five different evaluation metrics are utilized. The results show that Random Forest outperforms the other classifiers. Furthermore, Ripper Rule Learning performed the worst on the dataset according to the results attained from the evaluation metrics.

Table of Contents

Chapter 1 Machine Learning	8-13
Chapter 2 Python	14-23
Chapter 3 Case Study	24
Chapter 4 Model Building	24-45
Chapter 5 Conclusion	45
Chapter 6 References	45

List of Figures

Figure 1.1 The process flow	10
Figure 1.2 Supervised Learning	11
Figure 1.3 Unsupervised Learning	13
Figure 1.4 Semi Supervised Learning	14
Figure 2.1 Python Download	17
Figure 2.2 Anaconda Download	18
Figure 2.3 Jupyter Notebook	19
Figure 2.4 Defining a Class	23
Figure 4.1 Importing Libraries	28
Figure 4.2 Reading Dataset	29
Figure 4.3 Describing Dataset	30
Figure 4.4 Sentiment Analysis	30
Figure 4.5 Splitting the Train Dataset	31
Figure 4.6 Tokenization of text	32
Figure 4.7 Removing html strips	32
Figure 4.8 Removing Special characters	33

Figure 4.9 Stemming Text	34
Figure 4.10 Displaying Stopwords	34
Figure 4.11 Removing Stopwords	35
Figure 4.12 Normalizing Test Data	35
Figure 4.13 BOW Model	36
Figure 4.14 TFIDF Model	37
Figure 4.15 Labeling Statement	37
Figure 4.16 Splitting Sentiment Data	38
Figure 4.17 Logistic Regression on BOW and TFIDF	38
Figure 4.18 LR Performance	39
Figure 4.19 Accuracy on Dataset	39
Figure 4.20 Classification Report	40
Figure 4.21 Confusion Matrix	40
Figure 4.22 MNB for BOW and TFIDF	41
Figure 4.23 Performance of NB on Dataset	42
Figure 4.24 Accuracy of Model	42
Figure 4.25 Classification Report	42
Figure 4.26 Confusion Matrix	43
Figure 4.27 Loading Data	43
Figure 4.28 KNN on BOW and TFIDF	44
Figure 4.29 Performance on Dataset	44
Figure 4.30 Accuracy Model	44
Figure 4.31 Classification Report	45
Figure 4.32 Confusion Matrix	45

CHAPTER 1 MACHINE LEARNING

INTRODUCTION:

Machine Learning (ML) is the scientific study of algorithms and statistical models that computer systems use in order to perform a specific task effectively without using explicit instructions, relying on patterns and inference instead. It is seen as a subset of Artificial Intelligence (AI).

IMPORTANCE OF MACHINE LEARNING:

Consider some of the instances where machine learning is applied: the self-driving Google car, cyber fraud detection, online recommendation engines—like friend suggestions on Facebook, Netflix showcasing the movies and shows you might like, and “more items to consider” and “get yourself a little something” on Amazon—are all examples of applied machine learning. All these examples echo the vital role machine learning has begun to take in today’s data-rich world.

Machines can aid in filtering useful pieces of information that help in major advancements, and we are already seeing how this technology is being implemented in a wide variety of industries.

With the constant evolution of the field, there has been a subsequent rise in the uses, demands, and importance of machine learning. Big data has become quite a buzzword in the last few years; that’s in part due to increased sophistication of machine learning, which helps analyze those big chunks of big data. Machine learning has also changed the way data extraction, and interpretation is done by involving automatic sets of generic methods that have replaced traditional statistical techniques.

The process flow depicted here represents how machine learning works

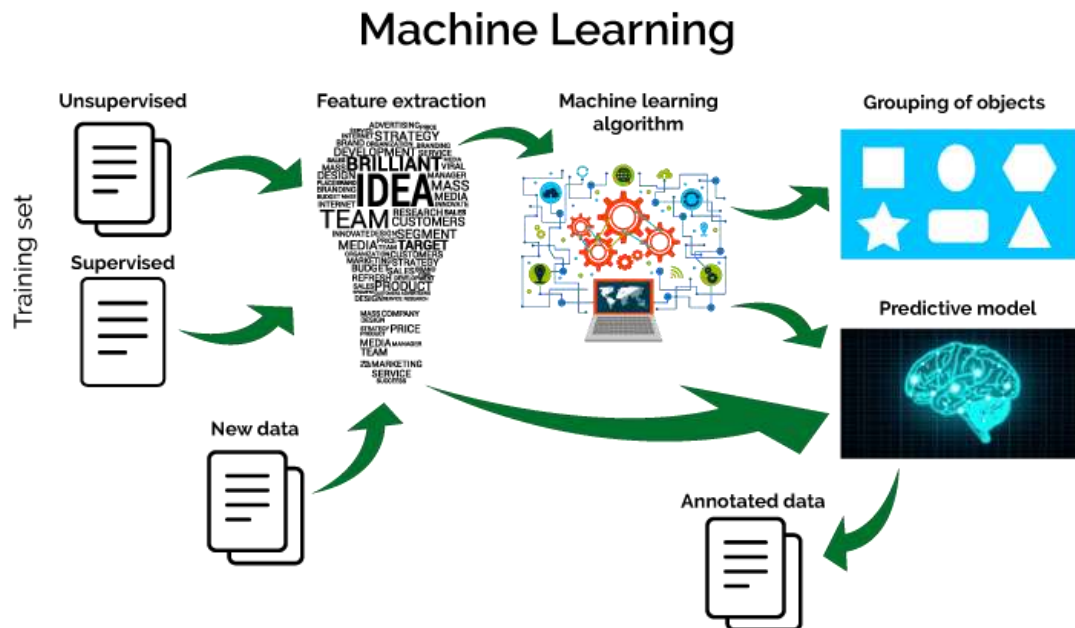


Figure 1.1 The Process Flow

USES OF MACHINE LEARNING:

Earlier in this article, we mentioned some applications of machine learning. To understand the concept of machine learning better, let's consider some more examples: web search results, real-time ads on web pages and mobile devices, email spam filtering, network intrusion detection, and pattern and image recognition. All these are by-products of applying machine learning to analyze huge volumes of data

Traditionally, data analysis was always being characterized by trial and error, an approach that becomes impossible when data sets are large and heterogeneous. Machine learning comes as the solution to all this chaos by proposing clever alternatives to analyzing huge volumes of data.

By developing fast and efficient algorithms and data-driven models for real-time processing of data, machine learning can produce accurate results and analysis.

TYPES OF LEARNING ALGORITHMS:

The types of machine learning algorithms differ in their approach, the type of data they input and output, and the type of task or problem that they are intended to solve.

Supervised Learning:

When an algorithm learns from example data and associated target responses that can consist of numeric values or string labels, such as classes or tags, in order to later predict the correct response when posed with new examples comes under the category of supervised learning.

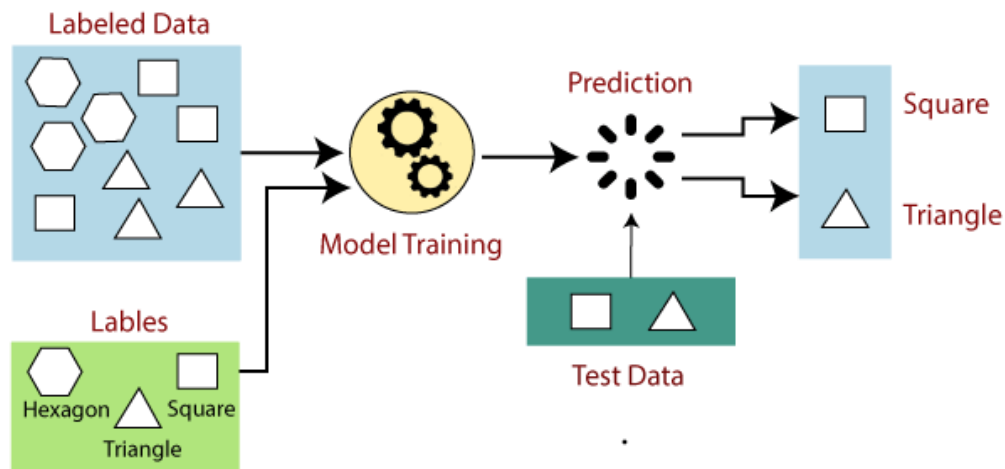


Figure 1.2 Supervised learning

Supervised machine learning algorithms uncover insights, patterns, and relationships from a labelled training dataset – that is, a dataset that already contains a known value for the target variable for each record. Because you provide the machine learning algorithm with the correct answers for a problem during training, it is able to “learn” how the rest of the features relate to the target, enabling you to uncover insights and make predictions about future outcomes based on historical data.

Examples of Supervised Machine Learning Techniques are Regression, in which the algorithm returns a numerical target for each example, such as how much revenue will be generated from a new marketing campaign.

Classification in which the algorithm attempts to label each example by choosing between two or more different classes. Choosing between two classes is called binary classification, such as determining whether or not someone will default on a loan. Choosing between more than two classes is referred to as multiclass classification.

Unsupervised Learning:

When an algorithm learns from plain examples without any associated response, leaving to the algorithm to determine the data patterns on its own. This type of algorithm tends to restructure the data into something else, such as new features that may represent a class or a new series of uncorrelated values. They are quite useful in providing humans with insights into the meaning of data and new useful inputs to supervised machine learning algorithms.

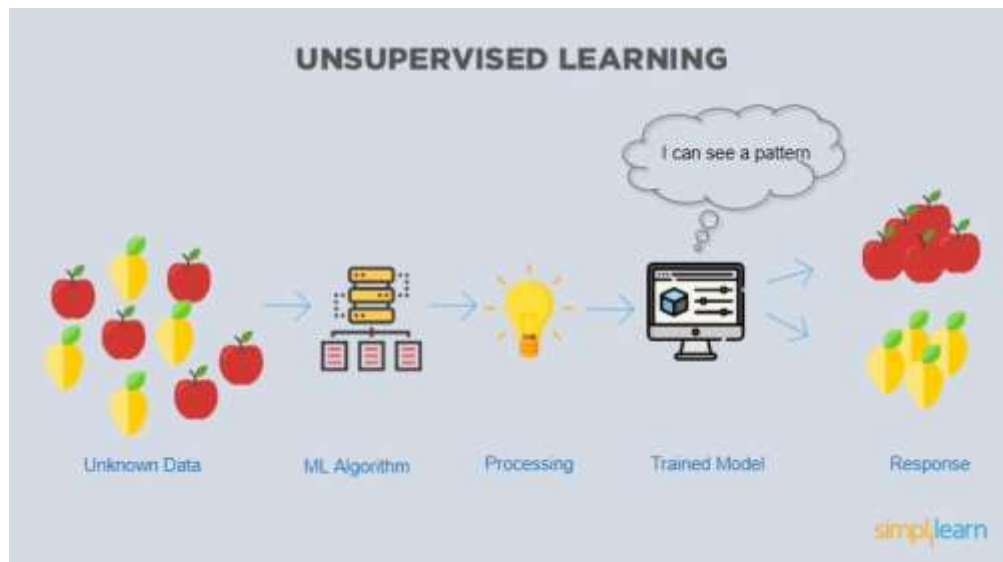


Figure 1.3 Unsupervised Learning

Popular techniques where unsupervised learning is used also include self-organizing maps, nearest neighbor mapping, singular value decomposition, and k-means clustering. Basically, online recommendations, identification of data outliers, and segment text topics are all examples of unsupervised learning.

Semi Supervised Learning:

As the name suggests, semi-supervised learning is a bit of both supervised and unsupervised learning and uses both labeled and unlabeled data for training. In a typical scenario, the algorithm would use a small amount of labeled data with a large amount of unlabeled data.

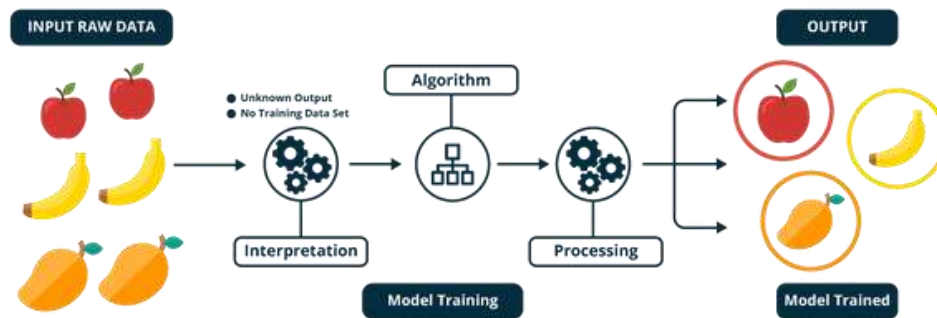


Figure 1.4 Semi Supervised Learning

RELATION BETWEEN DATA MINING, MACHINE LEARNING AND DEEP LEARNING:

Machine learning and data mining use the same algorithms and techniques as data mining, except the kinds of predictions vary. While data mining discovers previously unknown patterns and knowledge, machine learning reproduces known patterns and knowledge—and further automatically applies that information to data, decision-making, and actions.

Deep learning, on the other hand, uses advanced computing power and special types of neural networks and applies them to large amounts of data to learn, understand, and identify complicated patterns. Automatic language translation and medical diagnoses are examples of deep learning.

CHAPTER 2 PYTHON

Basic programming language used for machine learning is: PYTHON

INTRODUCTION TO PYHTON:

- Python is a high-level, interpreted, interactive and object-oriented scripting language.
- Python is a general purpose programming language that is often applied in scripting roles
- Python is interpreted: Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is like PERL and PHP.
- Python is Interactive: You can sit at a Python prompt and interact with the interpreter directly to write your programs.
- Python is Object-Oriented: Python supports the Object-Oriented style or technique of programming that encapsulates code within objects.

HISTORY OF PYTHON:

- Python was developed by GUIDO VAN ROSSUM in early 1990's
- Its latest version is 3.7 , it is generally called as python3

FEATURES OF PYTHON:

- Easy-to-learn: Python has few keywords, simple structure, and a clearly defined syntax, this allows the student to pick up the language quickly.
- Easy-to-read: Python code is more clearly defined and visible to the eyes.
- Easy-to-maintain: Python's source code is fairly easy-to-maintaining.
- A broad standard library: Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- Portable: Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- Extendable: You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- Databases: Python provides interfaces to all major commercial databases.
- GUI Programming: Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of UNIX.

HOW TO SETUP PYTHON:

- Python is available on a wide variety of platforms including Linux and Mac OS X. Let's understand how to set up our Python environment.
- The most up-to-date and current source code, binaries, documentation, news, etc., is available on the official website of Python.

Installation (using python IDLE):

- Installing python is generally easy, and nowadays many Linux and Mac OS distributions include a recent python.
- Download python from www.python.org
- When the download is completed, double click the file and follow the instructions to install it.
- When python is installed, a program called IDLE is also installed along with it. It provides a graphical user interface to work with python.



Figure 2.1 Python download

Installation (using Anaconda):

- Python programs are also executed using Anaconda.
- Anaconda is a free open source distribution of python for large scale dataprocessing, predictive analytics and scientific computing.
- In windows:
Step 1: Open Anaconda.com/downloads in web browser.
Step 2: Download python 3.4 version for (32-bit graphic installer/64-bit graphic installer)
Step 3: Select installation type (all users)
Step 4: Select path (i.e. add anaconda to path & register anaconda as default python 3.4) next click install and next click finish.
Step 5: Open jupyter notebook (it opens in default browser)

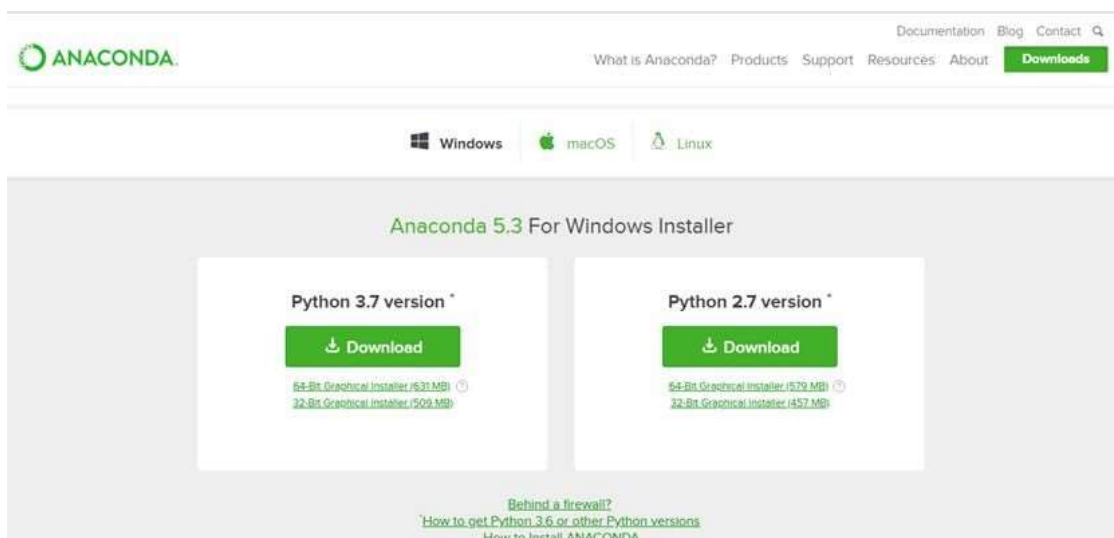


Figure 2.2 Anaconda download



Figure 2.3 Jupyter notebook

PYTHON VARIABLE TYPES:

- Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.
- Variables are nothing but reserved memory locations to store values.
- Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory.
- Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable.
- Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.

- Python has five standard data types
 - Numbers
 - Strings
 - Lists
 - Tuples
 - Dictionary

PythonNumbers:

- Number data types store numeric values. Number objects are created when you assign a value to them.
- Python supports four different numerical types – int (signed integers) long (longintegers, they can also be represented in octal and hexadecimal) float (floating point real values) complex (complex numbers).

PythonStrings:

- Strings in Python are identified as a contiguous set of characters represented in the quotation marks.
- Python allows for either pairs of single or double quotes.
- Subsets of strings can be taken using the slice operator ([] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.
- The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator.

PythonLists:

- Lists are the most versatile of Python's compound data types.
- A list contains items separated by commas and enclosed within square brackets ([]).
- To some extent, lists are similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data type.
- The values stored in a list can be accessed using the slice operator ([] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1.
- The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator.

PythonTuples:

- A tuple is another sequence data type that is similar to the list.
- A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.
- The main differences between lists and tuples are: Lists are enclosed in brackets ([]) and their elements and size can be changed, while tuples are enclosed in parentheses (()) and cannot be updated.
- Tuples can be thought of as read-only lists.
- For example – Tuples are fixed size in nature whereas lists are dynamic. In other words, a tuple is immutable whereas a list is mutable. You can't add elements to a tuple. Tuples have no append or extend method. You can't remove elements from a tuple. Tuples have no remove or pop method.

Python Dictionary:

- Python's dictionaries are kind of hash table type. They work like associative arrays or hashes found in Perl and consist of key-value pairs. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.
- Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]).
- You can use numbers to "index" into a list, meaning you can use numbers to find out what's in lists. You should know this about lists by now, but make sure you understand that you can only use numbers to get items out of a list.
- What a dict does is let you use anything, not just numbers. Yes, a dict associates one thing to another, no matter what it is.

PYTHONFUNCTION:

Defining aFunction:

You can define functions to provide the required functionality. Here are simple rules to define a function in Python. Function blocks begin with the keyword `def` followed by the function name and parentheses (i.e.()).

Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses

The code block within every function starts with a colon (:) and is indented. The statement `return [expression]` exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as `return None`.

Calling a Function:

Defining a function only gives it a name, specifies the parameters that are to be included in the function and structures the blocks of code. Once the basic structure of a function is finalized, you can execute it by calling it from another function or directly from the Python prompt.

PYTHON USING OOP's CONCEPTS:

Class:

- **Class:** A user-defined prototype for an object that defines a set of attributes that characterize any object of the class. The attributes are data members (class variables and instance variables) and methods, accessed via dot notation.
- **Class variable:** A variable that is shared by all instances of a class. Class variables are defined within a class but outside any of the class's methods. Class variables are not used as frequently as instance variables are.
- **Data member:** A class variable or instance variable that holds data associated with a class and its objects.
- **Instance variable:** A variable that is defined inside a method and belongs only to the current instance of a class.

Defining a Class:

- We define a class in a very similar way how we define a function.
- Just like a function, we use parentheses and a colon after the class name (i.e. ()) when we define a class. Similarly, the body of our class is indented like a function's body is.

```
def my_function():  
    # the details of the  
    # function go here
```

```
class MyClass():  
    # the details of the  
    # class go here
```

Figure 2.4 Defining a Class

init method in Class:

- The init method — also called a constructor — is a special method that runs when an instance is created so we can perform any tasks to set up the instance.
- The init method has a special name that starts and ends with two underscores: `init()`.

CHAPTER 3 CASE STUDY

PROBLEM STATEMENT:

Movie Review Classification:

This database was created to predict the number of positive and negative reviews based on sentiments by using different classification models.

DATASET:

The following are the parameters present in the given dataset.

- **path:** where to cache the data (relative to `..\IMDB Dataset.csv`)
- **num_words:** integer or None. Words are ranked by how often they occur (in the training set) and only the num_words most frequent words are kept. Any less frequent word will appear as oov_char value in the sequence data. If None, all words are kept. Defaults to None, so all words are kept.
- **skip_top:** skip the top N most frequently occurring words (which may not be informative). These words will appear as oov_char value in the dataset. Defaults to 0, so no words are skipped.
- **maxlen:** int or None. Maximum sequence length. Any longer sequence will be truncated. Defaults to None, which means no truncation.
- **start_char:** int. The start of a sequence will be marked with this character. Defaults to 1 because 0 is usually the padding character.
- **index_from:** int. Index actual words with this index and higher.

3.3 OBJECTIVE OF THE CASE STUDY:

This work aims to build a classifier for recognizing the objective of the case study will be to classify the movie reviews on positive and negative comments. Positive comments gives label as good review and negative comments gives label as bad or average.

CHAPTER 4 MODEL BUILDING

PREPROCESSING OF THE DATA:

Preprocessing of the data actually involves the following steps:

GETTING THE DATASET:

We can get the data set from the database.

Dataset:

The Train and Test datasets which are considered in this notebook file are taken from drive.

The obtained IMDB dataset split into 25,000 reviews for training and 25,000 reviews for testing, each set consisting in 50% negative and 50% positive reviews.

<https://drive.google.com/drive/folders/1vZgoh1o5xT6vehsRdfAQEX4nA79yoSVe>

IMPORTING THE LIBRARIES:

- numpy package can be used to perform mathematical operations like 'mean'.
- pandas package can be used to process dataframes.
- seaborn package can be used to visualise data in the form of various effective
- graphs and plots.
- sklearn is the main package which is used for machine learning.
- LabelEncoder is used to encode the non-numeric data into numericals so that
- machine learning model can be built.
- train_test_split module is used to split the data into training and testing sets.
- LinearRegression module is used to fit a LinearRegression model.
- sklearn.metrics can be used to calculate statistical results like mean squared
- error, root mean squared error, etc.
- **TextBlob**: It provides a simple API for diving into common natural language processing (NLP) tasks such as part-of-speech tagging, noun phrase extraction, sentiment analysis, classification, translation, and more.
- RE specifies a set of strings that matches it; the functions in this module let you check if a particular string matches a given regular expression (or if a given regular expression matches a particular string, which comes down to the same thing).
- **NLTK** toolkit is one of the most powerful NLP libraries which contains **packages** to make machines understand human language and reply to it with an appropriate response.
- **Beautiful Soup** is a Python library for pulling data out of HTML and XML files.

Import necessary libraries:

Import necessary libraries

```
In [ ]: 1 #Load the libraries
        2 import numpy as np
        3 import pandas as pd
        4 import seaborn as sns
        5 import matplotlib.pyplot as plt
        6 import nltk
        7 from sklearn.feature_extraction.text import CountVectorizer
        8 from sklearn.feature_extraction.text import TfidfVectorizer
        9 from sklearn.preprocessing import LabelBinarizer
        10 from nltk.corpus import stopwords
        11 from nltk.stem.porter import PorterStemmer
        12 from wordcloud import WordCloud,STOPWORDS
        13 from nltk.stem import WordNetLemmatizer
        14 from nltk.tokenize import word_tokenize,sent_tokenize
        15 from bs4 import BeautifulSoup
        16 import spacy
        17 import re,string,unicodedata
        18 from nltk.tokenize.toktok import ToktokTokenizer
        19 from nltk.stem import LancasterStemmer,WordNetLemmatizer
        20 from sklearn.linear_model import LogisticRegression,SGDClassifier
        21 from sklearn.naive_bayes import MultinomialNB
        22 from sklearn.svm import SVC
        23 from textblob import TextBlob
        24 from textblob import Word
        25 from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
```

Fig 4.1 Importing libraries

IMPORTING THE DATA-SET:

Pandas in python provide an interesting method `read_csv()`. The `read_csv` function reads the entire dataset from a comma separated values file and we can assign it to a DataFrame to which all the operations can be performed. It helps us to access each and every row as well as columns and each and every value can be access using the dataframe. Any missing value or NaN value have to be cleaned.

READING THE DATA:

```
In [3]: 1 #importing the training data
        2 indb_data=pd.read_csv('C:\\Users\\ACER\\Desktop\\aim1\\IMDB Dataset.csv')
        3 print(indb_data.shape)
        4 indb_data.head(10)
```

(50000, 2)

```
Out[3]:
```

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The ...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Matte's 'Love in the Time of Money' is...	positive
5	Probably my all-time favorite movie, a story o...	positive
6	I sure would like to see a resurrection of a u...	positive
7	This show was an amazing, fresh & innovative l...	negative
8	Encouraged by the positive comments about this...	negative
9	If you like original gut wrenching laughter yo...	positive

Fig 4.2 Reading dataset

Visualizing summary of dataset:

```
In [4]: 1 #Summary of the dataset
        2 imdb_data.describe()

Out[4]:
```

	review	sentiment
count	50000	50000
unique	49582	2
top	Loved today's show!!! It was a variety and not...	positive
freq	5	25000

Fig 4.3 Describing Dataset

Sentiment Analysis:

In simple terms, **sentiment analysis**, also called opinion mining or emotion AI, **works** by using natural language processing (NLP) and machine training to pair AI training data with predefined labels such as positive, negative, or neutral.

```
Sentiment count

In [5]: 1 #sentiment count
        2 imdb_data['sentiment'].value_counts()

Out[5]: positive    25000
        negative    25000
        Name: sentiment, dtype: int64

We can see that dataset is balanced
```

Fig 4.4 Sentiment analysis

Splitting the training Dataset:

Training data:

Training data is used to **train** an algorithm, typically making up a certain percentage of an overall dataset along with a testing **set**.

Why do we split the training dataset?

The data we use is usually **split into training** data and test data. The **training** set contains a known output and the model learns on this data **in order to** be generalized **to** other data later on.



```
Splitting the training dataset

In [6]: 1 #split the dataset
        2 #train dataset
        3 train_reviews=imdb_data.review[:40000]
        4 train_sentiments=imdb_data.sentiment[:40000]
        5 #test dataset
        6 test_reviews=imdb_data.review[40000:]
        7 test_sentiments=imdb_data.sentiment[40000:]
        8 print(train_reviews.shape,train_sentiments.shape)
        9 print(test_reviews.shape,test_sentiments.shape)

(40000,) (40000,)
(10000,) (10000,)
```

Fig 4.5 Splitting the train dataset

Text Normalization:

Why data normalization is important?

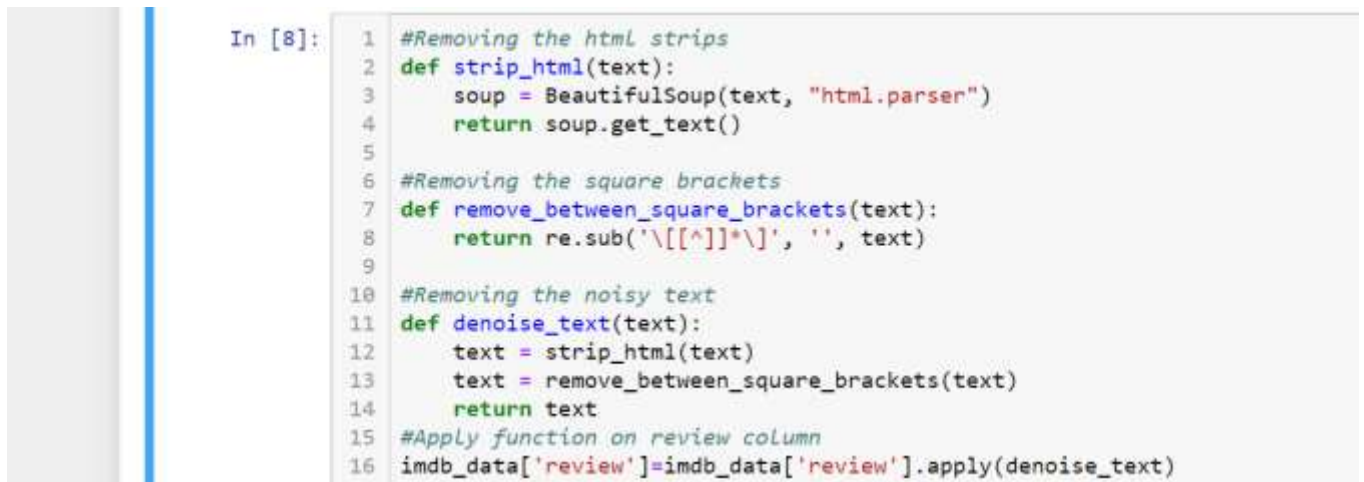
The goal of **normalization** is to change the values of numeric columns in the dataset to use a common scale, without distorting differences in the ranges of values or losing information. **Normalization** is also required for some algorithms to model the **data** correctly.



```
In [7]: 1 #Tokenization of text
2 tokenizer=ToktokTokenizer()
3 #Setting English stopwords
4 stopword_list=nlk.corpus.stopwords.words('english')
```

Fig 4.6 Tokenization of text

Removing html strips and noise text



```
In [8]: 1 #Removing the html strips
2 def strip_html(text):
3     soup = BeautifulSoup(text, "html.parser")
4     return soup.get_text()
5
6 #Removing the square brackets
7 def remove_between_square_brackets(text):
8     return re.sub('\[[^\]]*\]', '', text)
9
10 #Removing the noisy text
11 def denoise_text(text):
12     text = strip_html(text)
13     text = remove_between_square_brackets(text)
14     return text
15
16 #Apply function on review column
17 imdb_data['review']=imdb_data['review'].apply(denoise_text)
```

Fig 4.7 Removing html strips and noise data

- **Beautiful Soup** is a Python library for pulling data out of HTML and XML files.
- The **Python** module **re** provides full support for regular expressions in **Python**.
- **Denoising** is done to remove unwanted noise from image to analyze it in better form.

Removing special characters:

```
In [9]: 1 #Define function for removing special characters
2 def remove_special_characters(text, remove_digits=True):
3     pattern=r'^a-zA-Z0-9\s'
4     text=re.sub(pattern,'',text)
5     return text
6 #Apply function on review column
7 imdb_data['review']=imdb_data['review'].apply(remove_special_characters)
```

Fig 4.8 Removing special characters.

Text Stemming:

Stemming:

Stemming is the process of reducing a word to its word stem that affixes to suffixes and prefixes or to the roots of words known as a lemma. Stemming is important in natural language understanding (NLU) and natural language processing (NLP).

```
In [10]: 1 #Stemming the text
2 def simple_stemmer(text):
3     ps=nltk.porter.PorterStemmer()
4     text= ' '.join([ps.stem(word) for word in text.split()])
5     return text
6 #Apply function on review column
7 imdb_data['review']=imdb_data['review'].apply(simple_stemmer)
```

Fig 4.9 Stemming text.

Removing Stopwords:

```
In [11]: 1 #set stopwords to english
2 stop-set(stopwords.words('english'))
3 print(stop)

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

Fig 4.10 Displaying stopwords

Stopwords:

Stopwords are the English **words** which does not add much meaning to a sentence. They can safely be ignored without sacrificing the meaning of the sentence.

```
In [12]: 1 #removing the stopwords
          2 def remove_stopwords(text, is_lower_case=False):
          3     tokens = tokenizer.tokenize(text)
          4     tokens = [token.strip() for token in tokens]
          5     if is_lower_case:
          6         filtered_tokens = [token for token in tokens if token not in stopwords_list]
          7     else:
          8         filtered_tokens = [token for token in tokens if token.lower() not in stopwords_list]
          9     filtered_text = ' '.join(filtered_tokens)
         10     return filtered_text

In [13]: 1 #Apply function on review column
          2 imdb_data['review'] = imdb_data['review'].apply(remove_stopwords)
```

Fig 4.11 Removing stopwords

Normalized test reviews:

Normalization avoids these problems by creating new values that maintain the general distribution and ratios in the source **data**, while keeping values within a scale applied across all numeric columns used in the model.

Normalized test reviews

```
In [15]: 1 #Normalized test reviews
          2 norm_test_reviews = imdb_data.review[40000:]
          3 norm_test_reviews = [45005]
          4 #convert dataframe to string
          5 norm_test_string = norm_test_reviews.to_string()
          6 #spelling correction using TextBlob
          7 norm_test_spelling = TextBlob(norm_test_string)
          8 #print(norm_test_spelling.correct())
          9 #Tokenization using TextBlob
         10 norm_test_words = norm_test_spelling.words
         11 norm_test_words

Out[15]: 'read review watch thi piec cinemat garbag took least 2 page find somebodi els didnt think thi appallingli unfunni montag wasnt
acm humour 70 inde ani era thi isnt least funni set sketch comedi ive ever seen itll till come along half skit already done inf
init better act monty python woodi allen wa say nice piec anim last 90 second highlight thi film would still get close sum mind
less drive ridden thi wast 75 minut samin comedi onli world semin realli doe mean semen scatolog humour onli world scat actual
face precursor joke onli mean thi handbook comedi tit bun odd beaver niceif pubesc boy least one hand free havent found playboy
exist give break becaus wa earli 70 way sketch comedi go back least ten year prior onli way could even forgiv thi film even sad
e wa gunpoint retro hardli sketch clown subtil pervert children may cut edg clrci could actual funni come realli quit sad kept
go throughout entir 75 minut sheer belief may save genuin funni skit end gave film i becaus wa lower scoreand onli recommend in
sonniac coma patientsor perhap peopl suffer lockjawtheir jaw would final drop open disbelief'
```

Fig 4.12 Normalizing test data

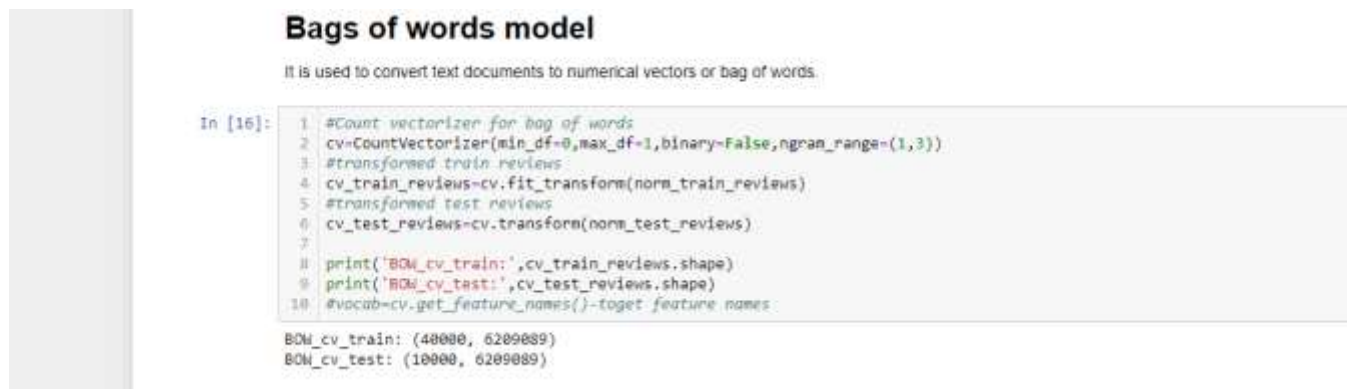
Bag of words model:

- **Count vectorizer:**

The **CountVectorizer** provides a simple way to both tokenize a collection of text documents and build a vocabulary of known words, but also to encode new documents **using** that vocabulary.

- **BOW (Bag of Words):**

A **bag-of-words** model, or BOW for short, is a way of extracting features from text for use in modeling, such as with **machine learning** algorithms.



```
Bags of words model
It is used to convert text documents to numerical vectors or bag of words.

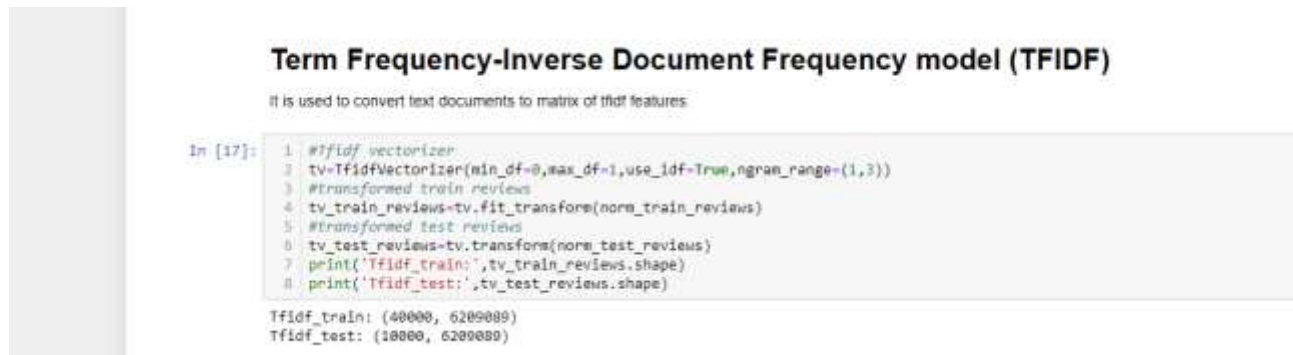
In [16]: 1 #Count vectorizer for bag of words
          2 cv=CountVectorizer(min_df=0,max_df=1,binary=False,ngram_range=(1,3))
          3 #transformed train reviews
          4 cv_train_reviews=cv.fit_transform(norm_train_reviews)
          5 #transformed test reviews
          6 cv_test_reviews=cv.transform(norm_test_reviews)
          7
          8 print('BOW_cv_train:',cv_train_reviews.shape)
          9 print('BOW_cv_test:',cv_test_reviews.shape)
         10 #vocab=cv.get_feature_names()-to get feature names

BOW_cv_train: (40000, 6209089)
BOW_cv_test: (10000, 6209089)
```

Fig 4.13 BOW model

Term Frequency-Inverse Document Frequency model (TFIDF):

In information retrieval, tf-idf or TFIDF, short for term frequency-inverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus.



The screenshot shows a Jupyter Notebook cell with the title "Term Frequency-Inverse Document Frequency model (TFIDF)". Below the title, a text box states "It is used to convert text documents to matrix of tfidf features". The code cell contains the following Python code:

```
In [17]: 1 #tfidf vectorizer
2 tv=TfidfVectorizer(min_df=0,max_df=1,use_idf=True,ngram_range=(1,3))
3 #transformed train reviews
4 tv_train_reviews=tv.fit_transform(norm_train_reviews)
5 #transformed test reviews
6 tv_test_reviews=tv.transform(norm_test_reviews)
7 print('Tfidf_train:',tv_train_reviews.shape)
8 print('Tfidf_test:',tv_test_reviews.shape)
```

The output of the code is displayed below the cell:

```
Tfidf_train: (40000, 6200000)
Tfidf_test: (10000, 6200000)
```

Fig 4.14 TFIDF model

Labeling the sentiment text:



The screenshot shows a Jupyter Notebook cell with the title "Labeling the sentiment text". The code cell contains the following Python code:

```
In [18]: 1 #labeling the sentiment data
2 lb=LabelBinarizer()
3 #transformed sentiment data
4 sentiment_data=lb.fit_transform(loaded_data['sentiment'])
5 print(sentiment_data.shape)
```

The output of the code is displayed below the cell:

```
(50000, 1)
```

Fig 4.15 Labeling sentiment

Split the sentiment data:

```
Split the sentiment data

In [19]: 1 #Splitting the sentiment data
          2 train_sentiments=sentiment_data[:40000]
          3 test_sentiments=sentiment_data[40000:]
          4 print(train_sentiments)
          5 print(test_sentiments)

[[1]
 [1]
 [1]
 ...
 [1]
 [0]
 [0]]
[[0]
 [0]
 [0]
 ...
 [0]
 [0]
 [0]]
```

Fig 4.16 splitting sentiment data

Modelling the dataset:

Let us see the performance of dataset using

1. Logistic Regression
2. Naive bayes
3. Knn(K-Nearest Neighbor)

Let us build logistic regression model for both bag of words and tfidf features

```
In [20]: 1 #training the model
          2 lr=LogisticRegression(penalty='l2',max_iter=500,C=1,random_state=42)
          3 #fitting the model for Bag of words
          4 lr_bow=lr.fit(cv_train_reviews,train_sentiments)
          5 print(lr_bow)
          6 #fitting the model for tfidf features
          7 lr_tfidf=lr.fit(tv_train_reviews,train_sentiments)
          8 print(lr_tfidf)

LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,
 intercept_scaling=1, max_iter=500, multi_class='warn',
 n_jobs=None, penalty='l2', random_state=42, solver='warn',
 tol=0.0001, verbose=0, warm_start=False)
LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,
 intercept_scaling=1, max_iter=500, multi_class='warn',
 n_jobs=None, penalty='l2', random_state=42, solver='warn',
 tol=0.0001, verbose=0, warm_start=False)
```

Fig 4.17 Logistic regression on BOW and TFIDF

Logistic regression model performance on test dataset:

```
Logistic regression model performane on test dataset

In [21]: 1 #Predicting the model for bag of words
          2 lr_bow_predict=lr.predict(cv_test_reviews)
          3 print(lr_bow_predict)
          4 ##Predicting the model for tfidf features
          5 lr_tfidf_predict=lr.predict(tv_test_reviews)
          6 print(lr_tfidf_predict)

          [0 0 0 ... 0 1 1]
          [0 0 0 ... 0 1 1]
```

Fig 4.18 LR performance

Accuracy of the model:

```
Accuracy of the model

In [22]: 1 #Accuracy score for bag of words
          2 lr_bow_score=accuracy_score(test_sentiments,lr_bow_predict)
          3 print("lr_bow_score :",lr_bow_score)
          4 #Accuracy score for tfidf features
          5 lr_tfidf_score=accuracy_score(test_sentiments,lr_tfidf_predict)
          6 print("lr_tfidf_score :",lr_tfidf_score)

lr_bow_score : 0.7512
lr_tfidf_score : 0.75
```

Fig 4.19 Accuracy on dataset

Classification report:

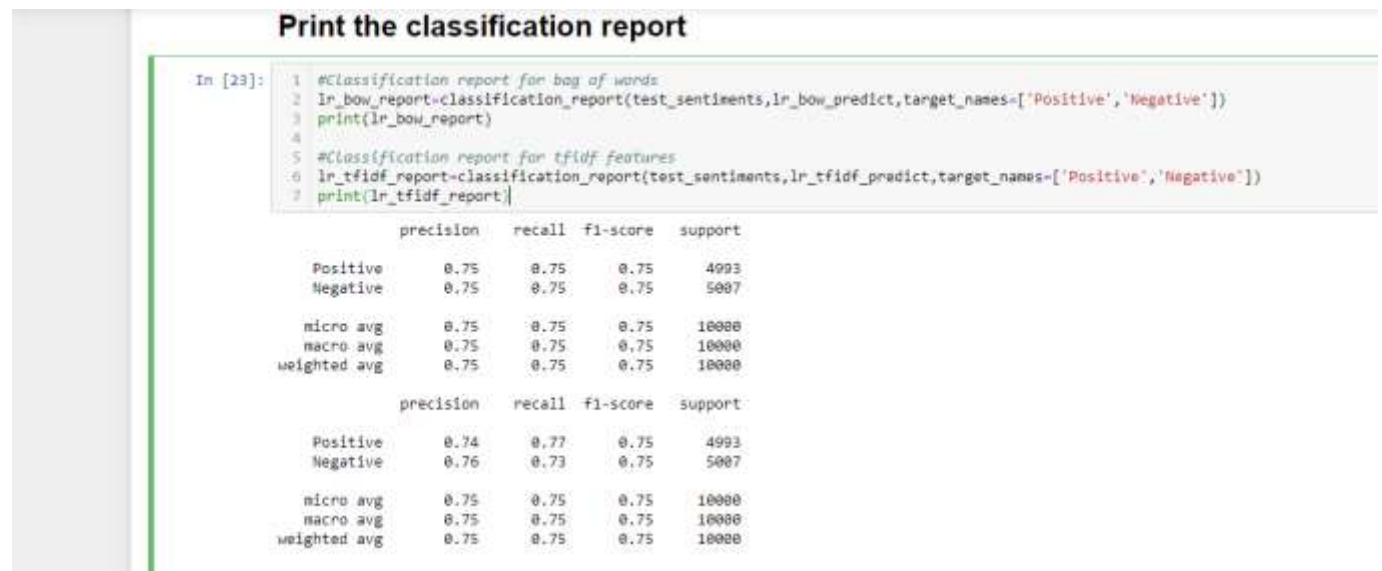


Fig 4.20 Classification report

Confusion matrix:

A confusion matrix is a technique for summarizing the performance of a classification algorithm.



Fig 4.21 Confusion matrix

Naive Bayes:

The Naive Bayes classifier is a simple classifier that classifies based on probabilities of events. It is applied commonly to text classification. Though it is a simple algorithm, it performs well in many text classification problems.

There are a lot of extensions we could add to this algorithm to make it perform better. We could remove punctuation and other non-characters. We could remove stopwords, or perform stemming or lemmatization.

We don't want to have to code the entire algorithm out every time, though. An easier way to use Naive Bayes is to use the implementation in scikit-learn. Scikit-learn is a Python machine learning library that contains implementations of all the common machine learning algorithms

Multinomial Naive Bayes for bag of words and tfidf features:

It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.



```
In [30]: 1 #training the model
          2 mnb=MultinomialNB()
          3 #fitting the svm for bag of words
          4 mnb_bow=mnb.fit(cv_train_reviews,train_sentiments)
          5 print(mnb_bow)
          6 #fitting the svm for tfidf features
          7 mnb_tfidf=mnb.fit(tv_train_reviews,train_sentiments)
          8 print(mnb_tfidf)

MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
```

Fig 4.22 MNB for BOW and TFIDF

Performance on test data:

```
Model performance on test data

In [31]: 1 #Predicting the model for bag of words
          2 mnb_bow_predict=mnb.predict(cv_test_reviews)
          3 print(mnb_bow_predict)
          4 #Predicting the model for tfidf features
          5 mnb_tfidf_predict=mnb.predict(tv_test_reviews)
          6 print(mnb_tfidf_predict)

[0 0 0 ... 0 1 1]
[0 0 0 ... 0 1 1]
```

Fig 4.23 Performance of NB on dataset

Accuracy of the model:

```
In [32]: 1 #Accuracy score for bag of words
          2 mnb_bow_score=accuracy_score(test_sentiments,mnb_bow_predict)
          3 print("mnb_bow_score :",mnb_bow_score)
          4 #Accuracy score for tfidf features
          5 mnb_tfidf_score=accuracy_score(test_sentiments,mnb_tfidf_predict)
          6 print("mnb_tfidf_score :",mnb_tfidf_score)

mnb_bow_score : 0.751
mnb_tfidf_score : 0.7589
```

Fig 4.24 Accuracy of model

Printing the classification report:

```
In [33]: 1 #Classification report for bag of words
          2 mnb_bow_report=classification_report(test_sentiments,mnb_bow_predict,target_names=['Positive','Negative'])
          3 print(mnb_bow_report)
          4 #Classification report for tfidf features
          5 mnb_tfidf_report=classification_report(test_sentiments,mnb_tfidf_predict,target_names=['Positive','Negative'])
          6 print(mnb_tfidf_report)

              precision    recall  f1-score   support

 Positive      0.75      0.76      0.75      4993
 Negative      0.75      0.75      0.75      5007

 micro avg      0.75      0.75      0.75     10000
 macro avg      0.75      0.75      0.75     10000
 weighted avg    0.75      0.75      0.75     10000

              precision    recall  f1-score   support

 Positive      0.75      0.76      0.75      4993
 Negative      0.75      0.74      0.75      5007

 micro avg      0.75      0.75      0.75     10000
 macro avg      0.75      0.75      0.75     10000
 weighted avg    0.75      0.75      0.75     10000
```

Fig 4.25 Classification report

Plot the confusion matrix:

```
In [34]: 1 #confusion matrix for bag of words
2 cm_bow=confusion_matrix(test_sentiments.mnb_bow_predict,labels=[1,0])
3 print(cm_bow)
4 #confusion matrix for tfidf features
5 cm_tfidf=confusion_matrix(test_sentiments.mnb_tfidf_predict,labels=[1,0])
6 print(cm_tfidf)

[[3736 1271]
 [1219 3774]]
[[3729 1278]
 [1213 3780]]
```

Fig 4.26 Confusion matrix

Let us see the positive and negative reviews using KNN:

K-Nearest Neighbors:

The k-nearest neighbors algorithm (k-NN) is a non-parametric used for classification and regression.

In both cases, the input consists of the k closest training examples in the feature space. ...
In k-NN classification.

Importing required data:

```
In [39]: 1 import numpy as np
2 import pandas as pd
3 from matplotlib import pyplot as plt
4 from sklearn.datasets import load_breast_cancer
5 from sklearn.metrics import confusion_matrix
6 from sklearn.neighbors import KNeighborsClassifier
7 from sklearn.model_selection import train_test_split
8 import seaborn as sns
9 sns.set()
```

Fig 4.27 Loading data

knn for bag of words and tfidf features:

```
In [44]: 1 #training the model
2 knn = KNeighborsClassifier()
3 #fitting the svm for bag of words
4 knn_bow=knn.fit(cv_train_reviews,train_sentiments)
5 print(knn_bow)
6 #fitting the svm for tfidf features
7 knn_tfidf=knn.fit(tv_train_reviews,train_sentiments)
8 print(knn_tfidf)

KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
metric_params=None, n_jobs=None, n_neighbors=5, p=2,
weights='uniform')
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
metric_params=None, n_jobs=None, n_neighbors=5, p=2,
weights='uniform')
```

Fig 4.28 KNN on BOW and TFIDF

Model performance on data:

```
In [45]: 1 #Predicting the model for bag of words
2 knn_bow_predict=knn.predict(cv_test_reviews)
3 print(knn_bow_predict)
4 #Predicting the model for tfidf features
5 knn_tfidf_predict=knn.predict(tv_test_reviews)
6 print(knn_tfidf_predict)

[0 1 0 ... 0 1 0]
[0 0 0 ... 0 1 0]
```

Fig 4.29 Performance on dataset

Accuracy model:

```
In [46]: 1 #Accuracy score for bag of words
2 knn_bow_score=accuracy_score(test_sentiments,knn_bow_predict)
3 print("knn_bow_score :",knn_bow_score)
4 #Accuracy score for tfidf features
5 knn_tfidf_score=accuracy_score(test_sentiments,knn_tfidf_predict)
6 print("knn_tfidf_score :",knn_tfidf_score)

knn_bow_score : 0.5053
knn_tfidf_score : 0.507
```

Fig 4.30 Accuracy model

Printing the classification report:

```
In [47]: 1 #Classification report for bag of words
2 knn_bow_report=classification_report(test_sentiments,knn_bow_predict,target_names=['Positive','Negative'])
3 print(knn_bow_report)
4 #Classification report for tfidf features:
5 knn_tfidf_report=classification_report(test_sentiments,knn_tfidf_predict,target_names=['Positive','Negative'])
6 print(knn_tfidf_report)
```

	precision	recall	f1-score	support
Positive	0.50	0.85	0.63	4993
Negative	0.52	0.16	0.25	5807
micro avg	0.51	0.51	0.51	10800
macro avg	0.51	0.51	0.44	10800
weighted avg	0.51	0.51	0.44	10800

	precision	recall	f1-score	support
Positive	0.50	0.85	0.63	4993
Negative	0.53	0.16	0.25	5807
micro avg	0.51	0.51	0.51	10800
macro avg	0.51	0.51	0.44	10800
weighted avg	0.51	0.51	0.44	10800

Fig 4.31 Classification report

Plot the confusion matrix:

```
In [48]: 1 #confusion matrix for bag of words
2 cm_bow=confusion_matrix(test_sentiments,knn_bow_predict,labels=[1,0])
3 print(cm_bow)
4 #confusion matrix for tfidf features
5 cm_tfidf=confusion_matrix(test_sentiments,knn_tfidf_predict,labels=[1,0])
6 print(cm_tfidf)
```

```
[[ 809 4198]
 [ 749 4244]]
[[ 803 4204]
 [ 726 4267]]
```

Fig 4.32 Confusion matrix

Chapter 5 Conclusion:

From the results above, we can infer that for our problem statement, Logistic Regression Model with feature set using mixture of Unigrams and Bigrams is best. Apart from this, one can also use a Naïve Bayes' Classifier as they also provide good accuracy percentage. This might be because of over-fitting of decision trees to the training data. Also, low accuracy of kNN Classifiers shows us that people have varied writing styles and kNN Models are not suited to data with high variance.

One of the major improvements that can be incorporated as we move ahead in this project is to merge words with similar meanings before training the classifiers. Another point of improvement can be to model this problem as a multi-class classification problem where we classify the sentiments of reviewer in more than binary fashion like "Happy", "Bored", "Afraid", etc. This problem can be further remodeled as a regression problem where we can predict the degree of affinity for the movie instead of complete like/dislike.

Chapter 6 References:

- [1] <https://docs.python.org/3/tutorial/>
- [2] <https://towardsdatascience.com/text-classification-in-python-dd95d264c802>
- [3] <https://scikit-learn.org/stable/modules/classes.html>
- [4] <https://matplotlib.org/tutorials/introductory/pyplot.html>

GitHub Link - <https://github.com/Yashaswiniy-852/AIMLProject>