

**Subject: PYTHON WITH DATA SCIENCE**

**Submitted by: PREETI SHARMA**

**Submitted to: SOUMATANU MAJUMDAR**

**Name: PREETI SHARMA**

**College: KURUKSHETRA UNIVERSITY(U.I.E.T.)**

**Course: BIOTECH ENGINEERING**

**Year: 3rd**

**Year of passing: 2023**

**E-mail ID:** [**preetishr27@gmail.com**](mailto:preetishr27@gmail.com)

**Linkedin Profile -** [linkedin.com/in/preeti-sharma-185758168](https://www.linkedin.com/in/preeti-sharma-185758168?lipi=urn%3Ali%3Apage%3Ad_flagship3_profile_view_base_contact_details%3B1ayRQYGxTWuA1HorD16U0Q%3D%3D)

**GitHub Profile -** [**https://github.com/Preeti123-cmd**](https://github.com/Preeti123-cmd)

**Project Link - https://github.com/NiiT-Kolkata/Sentiment-Analysis/blob/main/sentiment%20analysis.ipynb**

**Contents**

1.Acknowledgement

2.Introduction

3.Advantages

4.Future Scope

5.System Requirements

6.Objectives

7.Source Code

8.Conclusion

**Acknowledgment**

I am privileged and grateful to acknowledge my knowledge to all those who have guided me to put these ideas, well above the level of simplicity and into something concrete. I express my warm gratitude to National Institute for Industrial Training for their constant guidance and supervision as well as for providing necessary information regarding the project. I would like to express my sole thanks of gratitude to (Soumotanu Majumdar) for his support, co-operation and encouragement which helped me in the completion of this project.

My journey of completing the project wouldn’t would remain incomplete if I don’t extend my gratitude to my parents for their love , affection , support and constant guidance.

**Introduction**

“A combination of information technology , modelling , and business management”- Dr. Thomas Miller of Northwestern University.

In today’s world Data science and Machine Learning are one of the most prominent topics in the arena of technology. Data science and machine learning are two distinguishable topics although they look analogous to each other. Combining , computing , comparing and concluding the insights of datas are known as data science while on the flip side machine learning is the processes by which a result can be obtained from the given data.

**What is Data Science ?**

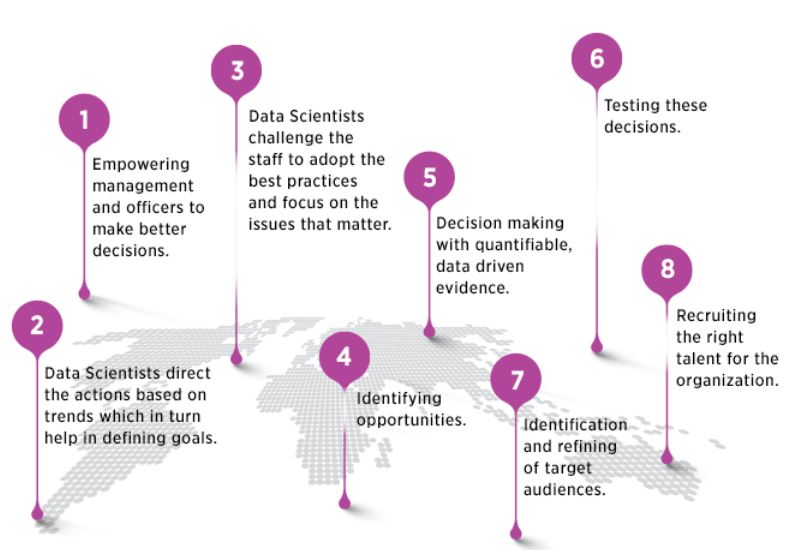
Have anyone wondered why our life in this era of iron age has become so comfortable and scientific? It’s only because of computers , smartphones , tablets , laptops and many more electronic devices which have completely digitalized our life and consequently resulted in huge amount data. This data needs to be processed , organized and coordinated. The phenomenon of mastering these process or the study of these data sets are basically categorized under data science.

Data Science is a blend of various tools, algorithms, and machine learning principles with the goal to discover hidden patterns from the raw data. But how is this different from what statisticians have been doing for years? The answer lies in the difference

between explaining and predicting. The principal purpose of Data Science is to find patterns within data. It uses various statistical techniques to analyze and draw insights from the data. From data extraction, wrangling and pre-processing, a Data Scientist must scrutinize the data thoroughly. Then, he has the responsibility of making predictions from the data. The goal of a Data Scientist is to derive conclusions from the data. Through these conclusions, he is able to assist companies in making smarter business decisions. We will divide this blog into various sections to understand the role of a Data Scientist in more detail. Industries need data to help them make careful decisions. Data Science churns raw data into meaningful insights. Therefore, industries need data science. A Data Scientist is a wizard who knows how to create magic using data. A skilled Data Scientist will know how to dig out meaningful information with whatever data he comes across. He helps the company in the right direction. The company requires strong data-driven decisions at which he’s an expert. The Data Scientist is an expert in various underlying fields of Statistics and Computer Science. He uses his analytical aptitude to solve business problems. Data Scientist is well versed with problem-solving and is assigned to find patterns in data. His goal is to recognize redundant samples and draw insights from it. Data science requires a variety of tools to extract information from the data. A Data Scientist is responsible for collecting, storing and maintaining the structured and unstructured form of data.

While the role of Data Science focuses on the analysis and management of data, it is dependent on the area that the company is specialized in. This requires the Data Scientist to have domain knowledge of that particular industry.

**Advantages**



**A great library ecosystem:** A great choice of libraries is one of the main reasons Python is the most popular programming language used for AI. A library is a module or a group of modules published by different sources like Py Pi which include a pre-written piece of code that allows users to reach some functionality or perform different actions. Python libraries provide base level items so developers don’t have to code them from the very beginning every time.

**A low entry barrier:** Working in the ML and AI industry means dealing with a bunch of data that you need to process in the most convenient and effective way. The low entry barrier allows more data scientists to quickly pick up Python and start using it for AI development without wasting too much effort on learning the language.

**Flexibility:** Python for machine learning is a great choice, as this language is very flexible. It offers an option to choose either to use OOPs or scripting. There’s also no need to recompile the source code, developers can implement any changes and quickly see the results. Programmers can combine Python and other languages to reach their goals. The flexibility factor decreases the possibility of errors, as programmers have a chance to take the situation under control and work in a comfortable environment.

**Platform independence:** Python is not only comfortable to use and easy to learn but also very versatile. What we mean is that Python for machine learning development can run on any platform including Windows, MacOS, Linux, UNIX, and twenty-one others. To transfer the process from one platform to another, developers need to implement several small-scale changes and modify some lines of code to create an executable form of code for the chosen platform. Developers can use packages like Py Installer to prepare their code for running on different platforms. Again, this saves time and money for tests on various platforms and makes the overall process more simple and convenient.

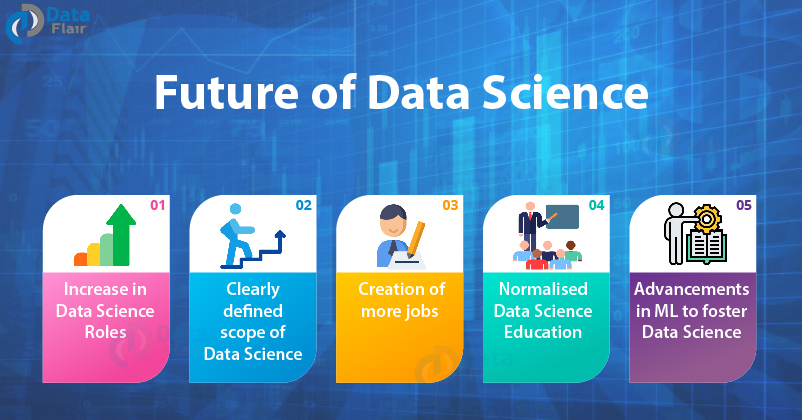
**Readability**: Python is very easy to read so every Python developer can understand the code of their peers and change, copy or share it. There’s no confusion, errors or conflicting paradigms, and this leads to a more efficient exchange of algorithms, ideas, and tools between AI and ML professionals.

**Good visualization options:** We’ve already mentioned that Python offers a variety of libraries, and some of them are great visualization tools. However, for AI developers, it’s important to highlight that inartificial intelligence, deep learning, and machine learning, it’s vital to be able to represent data in a human-readable format. Libraries like Matplotlib allow data scientists to build charts, histograms, and plots for better data comprehension, effective presentation, and visualization. Different application programming interfaces also simplify the visualization process and make it easier to create clear reports.

**Community support:** It’s always very helpful when there’s strong community support built around the programming language. Python is an open-source language which means that there’s a bunch of resources open for programmers starting from beginners and ending with pros. A lot of Python documentation is available online as well as in Python communities and forums, where programmers and machine learning developers discuss errors, solve problems, and help each other out. Python programming language is absolutely free as is the variety of useful libraries and tools.

**Growing Popularity:** As a result of the advantages discussed above, Python is becoming more and more popular among data scientists. This means it’s easier to search for developers and replace team players if required. Also, the cost of their work may be not as high as when using a less popular programming language.

**Future Scope**



Let’s dig deeper and see how Data Science is being used in various domains.

 How about if you could understand the precise requirements of your customers from the existing data like the customer’s past browsing history, purchase history, age and income. No doubt you had all this data earlier too, but now with the vast amount and variety of data, you can train models more effectively and recommend the product to your customers with more precision. Wouldn’t it be amazing as it will bring more business to your organization?

Let’s take a different scenario to understand the role of Data Science in decision making. How about if your car had the intelligence to drive you home? The self-driving cars collect live data from sensors, including radars, cameras, and lasers to create a map of its surroundings. Based

 on this data, it takes decisions like when to speed up, when to speed down, when to overtake, where to take a turn – making use of advanced machine learning algorithms.

 Let’s see how Data Science can be used in predictive analytics. Let’s take weather forecasting as an example. Data from ships, aircraft, radars, satellites can be collected and analyzed to build models. These models will not only forecast the weather but also help in predicting the occurrence of any natural calamities. It will help you to take appropriate measures beforehand and save many precious lives.

**SOURCE CODE**

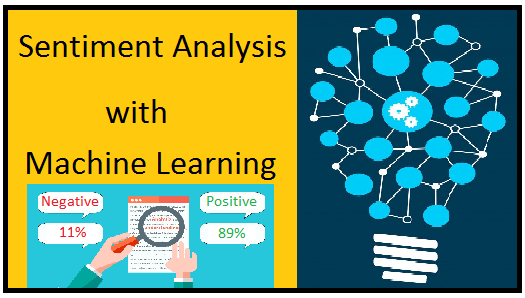
In [1]:

**from** **IPython.display** **import** Image

**from** **IPython.core.display** **import** HTML

Image(url= "https://miro.medium.com/max/525/1\*96kuTKGS6y9\_gf7MRS2gHw.png", width=900, height=600)

Out[1]:



or

from IPython.display import Image Image(url= "[https://miro.medium.com/max/525/1\*96kuTKGS6y9\_gf7MRS2gHw.png](https://miro.medium.com/max/525/1*96kuTKGS6y9_gf7MRS2gHw.png)")

USEFUL LINKS/CITATION -

<https://nycdatascience.com/blog/student-works/amazon-fine-foods-visualization/> <http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html>

# OBJECTIVE:

To do Sentiment Analysis and determine whether the Reviews from Amazon are Negative (rating of 1 or 2), Positive (rating of 4 or 5) or Neutral.

# OVERVIEW:

Sentiment analysis is the use of natural language processing, text analysis, computational linguistics, and biometrics to systematically identify, extract, quantify, and study affective states and subjective information.

Since customers express their thoughts and feelings more openly than ever before, sentiment analysis is becoming an essential tool to monitor and understand that sentiment. Automatically analyzing customer feedback, such as opinions in survey responses and social media conversations, allows brands to learn what makes customers happy or frustrated, so that they can tailor products and services to meet their customers’ needs. For example, using sentiment analysis to automatically analyze 4,000+ reviews about your product could help you discover if customers are happy about your pricing plans and customer service.

Ques- How to determine if a review is positive or negative?

[Ans] We could use the Score/Rating. A rating of 4 or 5 could be cosnidered a positive review. A review of 1 or 2 could be considered negative. A review of 3 is nuetral and ignored. This is an approximate and proxy way of determining the polarity (positivity/negativity) of a review.

# What is Natural Language Processing?

# Natural language processing (NLP) is an area of computer science and artificial intelligence concerned with the interactions between computers and human (natural) languages, in particular how to program computers to process and analyze large amounts of natural language data. Challenges in natural language processing frequently involve speech recognition, natural language understanding, and natural language generation.

Natural language processing includes many different techniques for interpreting human language, ranging from statistical and machine learning methods to rules-based and algorithmic approaches. We need a broad array of approaches because the text- and voice-based data varies widely, as do the practical applications.

# ARTIFICIAL NEURAL NETWORK(ANN):

An artificial neural network (ANN) is the component of artificial intelligence that is meant to simulate the functioning of a human brain. Processing units make up ANNs, which in turn consist of inputs and outputs. The inputs are what the ANN learns from to produce the desired output.

EXAMPLES - Commercial applications of these technologies generally focus on solving complex signal processing or pattern recognition problems. Examples of significant commercial applications since 2000 include handwriting recognition for check processing, speech-to-text transcription, oil-exploration data analysis, weather prediction and facial recognition.

# TYPES:

Fine-grained Sentiment Analysis If polarity precision is important to your business, you might consider expanding your polarity categories to include:

1.Very positive

2.Positive

3.Neutral

4.Negative

5.Very negative

# Is spaCy or NLTK better?

While NLTK provides access to many algorithms to get something done, spaCy provides the best way to do it. It provides the fastest and most accurate syntactic analysis of any NLP library released to date. It also offers access to larger word vectors that are easier to customize

# WHY IS IT IMPORTANT?

Sentiment analysis is extremely useful in social media monitoring as it allows us to gain an overview of the wider public opinion behind certain topics. Being able to quickly see the sentiment behind everything from forum posts to news articles means being better able to strategise and plan for the future. companies want their brand being perceived positively, or at least more positively than the brands of competitors.

# BASIC STEPS

The common and most basic steps are:

1.Remove URLs and email addresses from every single sample — because they won’t add meaningful value.

2.Remove punctuation signs — otherwise your model won’t understand that “good!” and “good” are actually meaning the same thing.

3.Lowercase all text — because you want to make the input text as generic as possible and avoid that, for example, a “Good” which is at the beginning of a phrase to be understood differently than the “good” in another sample.

4.Remove stop-words — stop-words refer to the most common words in a language, such as “I”, “have”, “are” and so on. I hope you get the point because there’s not an official stop-words list out there.

5.Stemming/Lemmatizing: Lemmatizing generally returns valid words (that exist) while stemming techniques return (most of the times) shorten words, that’s why lemmatizing is used more in real world implementations. This is how lemmatizers vs. stemmers work: suppose you want to find the root word of ‘caring’: ‘Caring’ -> Lemmatization -> ‘Care’. In the other hand: ‘Caring’ -> Stemming -> ‘Car’; did you get the point? You can research about both and obviously implement any if the business requires it.

6.Transform dataset (text) into numeric tensors — Usually referred as vectorization.like all other neural networks, deep-learning models don’t take as input raw text: they only work with numeric tensors, that’s why this step is not negotiable. There are multiple ways to do so; for example, if you’re going to use a classic ML model (not DL) then you definitely should go with CountVectorizer, TFIDF Vectorizer or just the basic but not so good approach: Bag-Of-Words. It’s up to you. However, if you’re going to implement Deep Learning you might know that the best way is to turn your text data (that can be understood as sequences of word or sequences of characters) into low-dimensional floating-point vectors

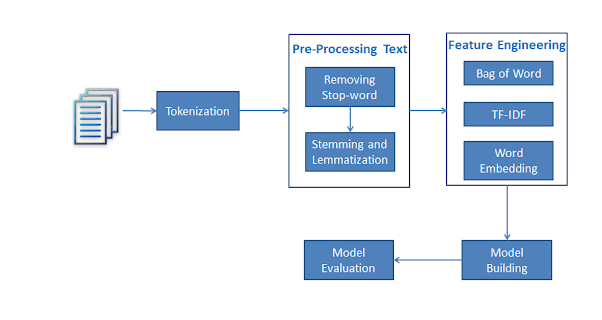
In [2]:

**from** **IPython.display** **import** Image

**from** **IPython.core.display** **import** HTML

Image(url= "https://miro.medium.com/max/650/1\*P9agr-brgGGbTCjA7PEBvQ.png", width=600, height=600)

Out[2]:



# ABOUT DATASET

Context This dataset consists of reviews of fine foods from amazon. The data span a period of more than 10 years, including all ~500,000 reviews up to October 2012. Reviews include product and user information, ratings, and a plain text review. It also includes reviews from all other Amazon categories.

Contents Reviews.csv: Pulled from the corresponding SQLite table named Reviews in database.sqlite

database.sqlite: Contains the table 'Reviews'

Data includes:

Reviews from Oct 1999 - Oct 2012

568,454 reviews

256,059 users

74,258 products

260 users with > 50 reviews

Attribute Information:

Id

ProductId - unique identifier for the product

UserId - unqiue identifier for the user

ProfileName HelpfulnessNumerator - number of users who found the review helpful

HelpfulnessDenominator - number of users who indicated whether they found the review helpful or not

Score - rating between 1 and 5

Time - timestamp for the review

Summary - brief summary of the review

Text - text of the review

Dataset source : <https://www.kaggle.com/snap/amazon-fine-food-reviews?select=Reviews.csv>

# SPACY

Spacy is written in cython language, (C extension of Python designed to give C like performance to the python program). Hence is a quite fast library. spaCy provides a concise API to access its methods and properties governed by trained machine (and deep) learning models.

Implementation of spacy and access to different properties is initiated by creating pipelines. A pipeline is created by loading the models. There are different type of models provided in the package which contains the information about language – vocabularies, trained vectors, syntaxes and entities.

These pipelines outputs a wide range of document properties such as – tokens, token’s reference index, part of speech tags, entities, vectors, sentiment, vocabulary etc.

a)Tokenization: Every spaCy document is tokenized into sentences and further into tokens which can be accessed by iterating the document.

b)Part of Speech Tagging: Part-of-speech tags are the properties of the word that are defined by the usage of the word in the grammatically correct sentence. These tags can be used as the text features in information filtering, statistical models, and rule based parsing.

c)Entity Detection Spacy consists of a fast entity recognition model which is capable of identifying entitiy phrases from the document. Entities can be of different types, such as – person, location, organization, dates, numerals, etc. These entities can be accessed through “.ents” property.

d)Dependency Parsing One of the most powerful feature of spacy is the extremely fast and accurate syntactic dependency parser which can be accessed via lightweight API. The parser can also be used for sentence boundary detection and phrase chunking. The relations can be accessed by the properties “.children” , “.root”, “.ancestor” etc.

e)Noun Phrases Dependency trees can also be used to generate noun phrases

f)Word to Vectors Integration Spacy also provides inbuilt integration of dense, real valued vectors representing distributional similarity information. It uses GloVe vectors to generate vectors. GloVe is an unsupervised learning algorithm for obtaining vector representations for words.

In [2]:

!pip install spacy

Requirement already satisfied: spacy in c:\users\work\anaconda3\lib\site-packages (3.0.6)

Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in c:\users\work\anaconda3\lib\site-packages (from spacy) (4.62.0)

Requirement already satisfied: thinc<8.1.0,>=8.0.3 in c:\users\work\anaconda3\lib\site-packages (from spacy) (8.0.7)

Requirement already satisfied: wasabi<1.1.0,>=0.8.1 in c:\users\work\anaconda3\lib\site-packages (from spacy) (0.8.2)

Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in c:\users\work\anaconda3\lib\site-packages (from spacy) (1.0.5)

Requirement already satisfied: catalogue<2.1.0,>=2.0.3 in c:\users\work\anaconda3\lib\site-packages (from spacy) (2.0.4)

Requirement already satisfied: jinja2 in c:\users\work\anaconda3\lib\site-packages (from spacy) (2.11.3)

Requirement already satisfied: srsly<3.0.0,>=2.4.1 in c:\users\work\anaconda3\lib\site-packages (from spacy) (2.4.1)

Requirement already satisfied: preshed<3.1.0,>=3.0.2 in c:\users\work\anaconda3\lib\site-packages (from spacy) (3.0.5)

Requirement already satisfied: pathy>=0.3.5 in c:\users\work\anaconda3\lib\site-packages (from spacy) (0.6.0)

Requirement already satisfied: blis<0.8.0,>=0.4.0 in c:\users\work\anaconda3\lib\site-packages (from spacy) (0.7.4)

Requirement already satisfied: typing-extensions<4.0.0.0,>=3.7.4 in c:\users\work\anaconda3\lib\site-packages (from spacy) (3.10.0.0)

Requirement already satisfied: cymem<2.1.0,>=2.0.2 in c:\users\work\anaconda3\lib\site-packages (from spacy) (2.0.5)

Requirement already satisfied: requests<3.0.0,>=2.13.0 in c:\users\work\anaconda3\lib\site-packages (from spacy) (2.25.1)

Requirement already satisfied: pydantic<1.8.0,>=1.7.1 in c:\users\work\anaconda3\lib\site-packages (from spacy) (1.7.4)

Requirement already satisfied: setuptools in c:\users\work\anaconda3\lib\site-packages (from spacy) (52.0.0.post20210125)

Requirement already satisfied: numpy>=1.15.0 in c:\users\work\anaconda3\lib\site-packages (from spacy) (1.19.5)

Requirement already satisfied: packaging>=20.0 in c:\users\work\anaconda3\lib\site-packages (from spacy) (21.0)

Requirement already satisfied: spacy-legacy<3.1.0,>=3.0.4 in c:\users\work\anaconda3\lib\site-packages (from spacy) (3.0.6)

Requirement already satisfied: typer<0.4.0,>=0.3.0 in c:\users\work\anaconda3\lib\site-packages (from spacy) (0.3.2)

Requirement already satisfied: zipp>=0.5 in c:\users\work\anaconda3\lib\site-packages (from catalogue<2.1.0,>=2.0.3->spacy) (3.5.0)

Requirement already satisfied: pyparsing>=2.0.2 in c:\users\work\anaconda3\lib\site-packages (from packaging>=20.0->spacy) (2.4.7)

Requirement already satisfied: smart-open<6.0.0,>=5.0.0 in c:\users\work\anaconda3\lib\site-packages (from pathy>=0.3.5->spacy) (5.1.0)

Requirement already satisfied: idna<3,>=2.5 in c:\users\work\anaconda3\lib\site-packages (from requests<3.0.0,>=2.13.0->spacy) (2.10)

Requirement already satisfied: chardet<5,>=3.0.2 in c:\users\work\anaconda3\lib\site-packages (from requests<3.0.0,>=2.13.0->spacy) (4.0.0)

Requirement already satisfied: urllib3<1.27,>=1.21.1 in c:\users\work\anaconda3\lib\site-packages (from requests<3.0.0,>=2.13.0->spacy) (1.26.6)

Requirement already satisfied: certifi>=2017.4.17 in c:\users\work\anaconda3\lib\site-packages (from requests<3.0.0,>=2.13.0->spacy) (2021.5.30)

Requirement already satisfied: colorama in c:\users\work\anaconda3\lib\site-packages (from tqdm<5.0.0,>=4.38.0->spacy) (0.4.4)

Collecting click<7.2.0,>=7.1.1

Using cached click-7.1.2-py2.py3-none-any.whl (82 kB)

Requirement already satisfied: MarkupSafe>=0.23 in c:\users\work\anaconda3\lib\site-packages (from jinja2->spacy) (2.0.1)

Installing collected packages: click

Attempting uninstall: click

Found existing installation: click 8.0.1

Uninstalling click-8.0.1:

Successfully uninstalled click-8.0.1

Successfully installed click-7.1.2

Error processing line 1 of C:\Users\work\anaconda3\lib\site-packages\matplotlib-3.4.2-py3.7-nspkg.pth:

Traceback (most recent call last):

File "C:\Users\work\anaconda3\lib\site.py", line 168, in addpackage

exec(line)

File "<string>", line 1, in <module>

File "<frozen importlib.\_bootstrap>", line 580, in module\_from\_spec

AttributeError: 'NoneType' object has no attribute 'loader'

Remainder of file ignored

WARNING: Ignoring invalid distribution -umpy (c:\users\work\anaconda3\lib\site-packages)

WARNING: Ignoring invalid distribution -umpy (c:\users\work\anaconda3\lib\site-packages)

WARNING: Ignoring invalid distribution -umpy (c:\users\work\anaconda3\lib\site-packages)

WARNING: Ignoring invalid distribution -umpy (c:\users\work\anaconda3\lib\site-packages)

WARNING: Ignoring invalid distribution -umpy (c:\users\work\anaconda3\lib\site-packages)

WARNING: Ignoring invalid distribution -umpy (c:\users\work\anaconda3\lib\site-packages)

WARNING: Ignoring invalid distribution -umpy (c:\users\work\anaconda3\lib\site-packages)

WARNING: Ignoring invalid distribution -umpy (c:\users\work\anaconda3\lib\site-packages)

In [4]:

!pip install pandas --upgrade

Requirement already satisfied: pandas in c:\users\work\anaconda3\lib\site-packages (1.3.2)

Error processing line 1 of C:\Users\work\anaconda3\lib\site-packages\matplotlib-3.4.2-py3.7-nspkg.pth:

Traceback (most recent call last):

File "C:\Users\work\anaconda3\lib\site.py", line 168, in addpackage

exec(line)

File "<string>", line 1, in <module>

File "<frozen importlib.\_bootstrap>", line 580, in module\_from\_spec

AttributeError: 'NoneType' object has no attribute 'loader'

Remainder of file ignored

WARNING: Ignoring invalid distribution -umpy (c:\users\work\anaconda3\lib\site-packages)

Requirement already satisfied: pytz>=2017.3 in c:\users\work\anaconda3\lib\site-packages (from pandas) (2021.1)

Requirement already satisfied: numpy>=1.17.3 in c:\users\work\anaconda3\lib\site-packages (from pandas) (1.19.5)

Requirement already satisfied: python-dateutil>=2.7.3 in c:\users\work\anaconda3\lib\site-packages (from pandas) (2.8.2)

Requirement already satisfied: six>=1.5 in c:\users\work\anaconda3\lib\site-packages (from python-dateutil>=2.7.3->pandas) (1.16.0)

WARNING: Ignoring invalid distribution -umpy (c:\users\work\anaconda3\lib\site-packages)

WARNING: Ignoring invalid distribution -umpy (c:\users\work\anaconda3\lib\site-packages)

WARNING: Ignoring invalid distribution -umpy (c:\users\work\anaconda3\lib\site-packages)

WARNING: Ignoring invalid distribution -umpy (c:\users\work\anaconda3\lib\site-packages)

WARNING: Ignoring invalid distribution -umpy (c:\users\work\anaconda3\lib\site-packages)

# The plotly Python library is an interactive, open-source plotting library that supports over 40 unique chart types covering a wide range of statistical, financial, geographic, scientific, and 3-dimensional use-cases.

In [7]:

!pip install plotly

Requirement already satisfied: plotly in c:\users\work\anaconda3\lib\site-packages (5.1.0)

Requirement already satisfied: tenacity>=6.2.0 in c:\users\work\anaconda3\lib\site-packages (from plotly) (7.0.0)

Requirement already satisfied: six in c:\users\work\anaconda3\lib\site-packages (from plotly) (1.16.0)

Error processing line 1 of C:\Users\work\anaconda3\lib\site-packages\matplotlib-3.4.2-py3.7-nspkg.pth:

Traceback (most recent call last):

File "C:\Users\work\anaconda3\lib\site.py", line 168, in addpackage

exec(line)

File "<string>", line 1, in <module>

File "<frozen importlib.\_bootstrap>", line 580, in module\_from\_spec

AttributeError: 'NoneType' object has no attribute 'loader'

Remainder of file ignored

WARNING: Ignoring invalid distribution -umpy (c:\users\work\anaconda3\lib\site-packages)

WARNING: Ignoring invalid distribution -umpy (c:\users\work\anaconda3\lib\site-packages)

WARNING: Ignoring invalid distribution -umpy (c:\users\work\anaconda3\lib\site-packages)

WARNING: Ignoring invalid distribution -umpy (c:\users\work\anaconda3\lib\site-packages)

WARNING: Ignoring invalid distribution -umpy (c:\users\work\anaconda3\lib\site-packages)

WARNING: Ignoring invalid distribution -umpy (c:\users\work\anaconda3\lib\site-packages)

In [8]:

!pip install spacy

Requirement already satisfied: spacy in c:\users\work\anaconda3\lib\site-packages (3.0.6)

Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in c:\users\work\anaconda3\lib\site-packages (from spacy) (1.0.5)

Requirement already satisfied: spacy-legacy<3.1.0,>=3.0.4 in c:\users\work\anaconda3\lib\site-packages (from spacy) (3.0.6)

Requirement already satisfied: wasabi<1.1.0,>=0.8.1 in c:\users\work\anaconda3\lib\site-packages (from spacy) (0.8.2)

Requirement already satisfied: pydantic<1.8.0,>=1.7.1 in c:\users\work\anaconda3\lib\site-packages (from spacy) (1.7.4)

Requirement already satisfied: blis<0.8.0,>=0.4.0 in c:\users\work\anaconda3\lib\site-packages (from spacy) (0.7.4)

Requirement already satisfied: jinja2 in c:\users\work\anaconda3\lib\site-packages (from spacy) (2.11.3)

Requirement already satisfied: thinc<8.1.0,>=8.0.3 in c:\users\work\anaconda3\lib\site-packages (from spacy) (8.0.7)

Requirement already satisfied: typer<0.4.0,>=0.3.0 in c:\users\work\anaconda3\lib\site-packages (from spacy) (0.3.2)

Requirement already satisfied: catalogue<2.1.0,>=2.0.3 in c:\users\work\anaconda3\lib\site-packages (from spacy) (2.0.4)

Requirement already satisfied: cymem<2.1.0,>=2.0.2 in c:\users\work\anaconda3\lib\site-packages (from spacy) (2.0.5)

Requirement already satisfied: typing-extensions<4.0.0.0,>=3.7.4 in c:\users\work\anaconda3\lib\site-packages (from spacy) (3.10.0.0)

Requirement already satisfied: numpy>=1.15.0 in c:\users\work\anaconda3\lib\site-packages (from spacy) (1.19.5)

Requirement already satisfied: setuptools in c:\users\work\anaconda3\lib\site-packages (from spacy) (52.0.0.post20210125)

Requirement already satisfied: packaging>=20.0 in c:\users\work\anaconda3\lib\site-packages (from spacy) (21.0)

Requirement already satisfied: srsly<3.0.0,>=2.4.1 in c:\users\work\anaconda3\lib\site-packages (from spacy) (2.4.1)

Requirement already satisfied: preshed<3.1.0,>=3.0.2 in c:\users\work\anaconda3\lib\site-packages (from spacy) (3.0.5)

Requirement already satisfied: requests<3.0.0,>=2.13.0 in c:\users\work\anaconda3\lib\site-packages (from spacy) (2.25.1)

Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in c:\users\work\anaconda3\lib\site-packages (from spacy) (4.62.0)

Requirement already satisfied: pathy>=0.3.5 in c:\users\work\anaconda3\lib\site-packages (from spacy) (0.6.0)

Requirement already satisfied: zipp>=0.5 in c:\users\work\anaconda3\lib\site-packages (from catalogue<2.1.0,>=2.0.3->spacy) (3.5.0)

Requirement already satisfied: pyparsing>=2.0.2 in c:\users\work\anaconda3\lib\site-packages (from packaging>=20.0->spacy) (2.4.7)

Requirement already satisfied: smart-open<6.0.0,>=5.0.0 in c:\users\work\anaconda3\lib\site-packages (from pathy>=0.3.5->spacy) (5.1.0)

Requirement already satisfied: chardet<5,>=3.0.2 in c:\users\work\anaconda3\lib\site-packages (from requests<3.0.0,>=2.13.0->spacy) (4.0.0)

Requirement already satisfied: certifi>=2017.4.17 in c:\users\work\anaconda3\lib\site-packages (from requests<3.0.0,>=2.13.0->spacy) (2021.5.30)

Requirement already satisfied: urllib3<1.27,>=1.21.1 in c:\users\work\anaconda3\lib\site-packages (from requests<3.0.0,>=2.13.0->spacy) (1.26.6)

Requirement already satisfied: idna<3,>=2.5 in c:\users\work\anaconda3\lib\site-packages (from requests<3.0.0,>=2.13.0->spacy) (2.10)

Requirement already satisfied: colorama in c:\users\work\anaconda3\lib\site-packages (from tqdm<5.0.0,>=4.38.0->spacy) (0.4.4)

Requirement already satisfied: click<7.2.0,>=7.1.1 in c:\users\work\anaconda3\lib\site-packages (from typer<0.4.0,>=0.3.0->spacy) (7.1.2)

Requirement already satisfied: MarkupSafe>=0.23 in c:\users\work\anaconda3\lib\site-packages (from jinja2->spacy) (2.0.1)

Error processing line 1 of C:\Users\work\anaconda3\lib\site-packages\matplotlib-3.4.2-py3.7-nspkg.pth:

Traceback (most recent call last):

File "C:\Users\work\anaconda3\lib\site.py", line 168, in addpackage

exec(line)

File "<string>", line 1, in <module>

File "<frozen importlib.\_bootstrap>", line 580, in module\_from\_spec

AttributeError: 'NoneType' object has no attribute 'loader'

Remainder of file ignored

WARNING: Ignoring invalid distribution -umpy (c:\users\work\anaconda3\lib\site-packages)

WARNING: Ignoring invalid distribution -umpy (c:\users\work\anaconda3\lib\site-packages)

WARNING: Ignoring invalid distribution -umpy (c:\users\work\anaconda3\lib\site-packages)

WARNING: Ignoring invalid distribution -umpy (c:\users\work\anaconda3\lib\site-packages)

WARNING: Ignoring invalid distribution -umpy (c:\users\work\anaconda3\lib\site-packages)

WARNING: Ignoring invalid distribution -umpy (c:\users\work\anaconda3\lib\site-packages)

In [9]:

!pip install numpy

Requirement already satisfied: numpy in c:\users\work\anaconda3\lib\site-packages (1.19.5)

Error processing line 1 of C:\Users\work\anaconda3\lib\site-packages\matplotlib-3.4.2-py3.7-nspkg.pth:

Traceback (most recent call last):

File "C:\Users\work\anaconda3\lib\site.py", line 168, in addpackage

exec(line)

File "<string>", line 1, in <module>

File "<frozen importlib.\_bootstrap>", line 580, in module\_from\_spec

AttributeError: 'NoneType' object has no attribute 'loader'

Remainder of file ignored

WARNING: Ignoring invalid distribution -umpy (c:\users\work\anaconda3\lib\site-packages)

WARNING: Ignoring invalid distribution -umpy (c:\users\work\anaconda3\lib\site-packages)

WARNING: Ignoring invalid distribution -umpy (c:\users\work\anaconda3\lib\site-packages)

WARNING: Ignoring invalid distribution -umpy (c:\users\work\anaconda3\lib\site-packages)

WARNING: Ignoring invalid distribution -umpy (c:\users\work\anaconda3\lib\site-packages)

WARNING: Ignoring invalid distribution -umpy (c:\users\work\anaconda3\lib\site-packages)

In [10]:

**from** **IPython.core.display** **import** display, HTML

display(HTML("<style>.container { width:100% !important; }</style>"))

**import** **pandas** **as** **pd** *# data processing, CSV file I/O (e.g. pd.read\_csv)*

**import** **numpy** **as** **np** *# linear algebra*

**import** **seaborn** **as** **sns**

color = sns.color\_palette()

**import** **spacy**

**from** **spacy** **import** displacy *#spaCy also comes with a built-in dependency visualizer that lets you check your model's predictions in your browse*

**from** **spacy.util** **import** minibatch, compounding

**import** **warnings**

warnings.filterwarnings(action="ignore")

**import** **matplotlib.pyplot** **as** **plt**

%matplotlib inline

**import** **plotly.offline** **as** **py**

py.init\_notebook\_mode(connected=**True**)

**import** **plotly.graph\_objs** **as** **go** *#Plotly's Python graphing library makes interactive, publication-quality graphs*

**import** **plotly.tools** **as** **tls**

**import** **plotly.express** **as** **px**

**from** **nltk.stem** **import** PorterStemmer *#The Porter stemming algorithm (or 'Porter stemmer') is a process for removing the commoner morphological and inflexional endings from words in English*

!pip install textblob *#TextBlob is a simple library which supports complex analysis and operations on textual data.*

**from** **textblob** **import** TextBlob

**import** **sqlite3**

**import** **string**

*# This is used for fast string concatination*

**from** **io** **import** StringIO

---------------------------------------------------------------------------

AttributeError Traceback (most recent call last)

<ipython-input-10-1d9781c2ce58> in <module>

1 from IPython.core.display import display, HTML

2 display(HTML("<style>.container { width:100% !important; }</style>"))

----> 3 import pandas as pd # data processing, CSV file I/O (e.g. pd.read\_csv)

4 import numpy as np # linear algebra

5 import seaborn as sns

~\anaconda3\lib\site-packages\pandas\\_\_init\_\_.py in <module>

20

21 # numpy compat

---> 22 from pandas.compat import (

23 np\_version\_under1p18 as \_np\_version\_under1p18,

24 is\_numpy\_dev as \_is\_numpy\_dev,

~\anaconda3\lib\site-packages\pandas\compat\\_\_init\_\_.py in <module>

12 import warnings

13

---> 14 from pandas.\_typing import F

15 from pandas.compat.numpy import (

16 is\_numpy\_dev,

~\anaconda3\lib\site-packages\pandas\\_typing.py in <module>

82 # array-like

83

---> 84 ArrayLike = Union["ExtensionArray", np.ndarray]

85 AnyArrayLike = Union[ArrayLike, "Index", "Series"]

86

AttributeError: module 'numpy' has no attribute 'ndarray'

Gensim is a free open-source Python library for representing documents as semantic vectors, as efficiently (computer-wise) and painlessly (human-wise) as possible. Gensim is designed to process raw, unstructured digital texts (“plain text”) using unsupervised machine learning algorithms.

In [ ]:

pip install gensim

In [ ]:

*#The sklearn. metrics module implements several loss, score, and utility functions to measure classification performance. Some metrics might require probability estimates*

*#of the positive class, confidence values, or binary decisions values.*

**from** **sklearn** **import** metrics

**from** **sklearn.metrics** **import** roc\_curve, auc *#The Area Under the Curve (AUC) is the measure of the ability of a classifier to distinguish between classes and is used as a summary of the ROC curve. The higher the AUC, the better the performance*

*#of the model at distinguishing between the positive and negative classes.*

**from** **nltk.stem.porter** **import** PorterStemmer

**import** **re**

*# Tutorial about Python regular expressions: https://pymotw.com/2/re/*

**import** **string**

**from** **nltk.corpus** **import** stopwords

**from** **nltk.stem.wordnet** **import** WordNetLemmatizer

**from** **gensim.models** **import** Word2Vec

**from** **gensim.models** **import** KeyedVectors

**import** **pickle**

**from** **tqdm** **import** tqdm

**import** **os**

In [ ]:

**from** **sklearn.feature\_extraction.text** **import** TfidfTransformer *#TF-IDF is a statistical measure that evaluates how relevant a word is to a document in a collection of documents.*

**from** **sklearn.feature\_extraction.text** **import** TfidfVectorizer *#Convert a collection of raw documents to a matrix of TF-IDF features.*

**from** **sklearn.feature\_extraction.text** **import** CountVectorizer *#Scikit-learn's CountVectorizer is used to convert a collection of text documents to a vector of term/token counts.*

In [ ]:

df = pd.read\_csv('Reviews.csv')

df.head()

In [ ]:

*# check data types and counts for each column*

df.info()

In [ ]:

*# check size of dataframe*

df.shape

In [ ]:

*# obtain list of unique words in Summary*

total\_options = df["Summary"]

total\_options.unique()

In [ ]:

*# obtain list of unique words in Text*

total\_options = df["Text"]

total\_options.unique()

In [ ]:

plt.figure(figsize=(10,5))

sns.countplot(dataframe['Score'], palette='gist\_rainbow')

plt.title("Distribution of Ratings across the whole dataset")

plt.xlabel("Reviews Ratings ")

plt.ylabel("Number of reviews corresponding to the 5 ratings")

plt.show()

print(dataframe['Score'].value\_counts())

In [ ]:

*#The Natural Language Toolkit (NLTK) is a platform used for building Python programs that work*

*#with human language data for applying in statistical natural language processing (NLP). It contains text processing libraries*

*#for tokenization, parsing, classification, stemming, tagging and semantic reasoning.*

**import** **nltk**

**from** **nltk.corpus** **import** stopwords

In [ ]:

*# assign reviews with score > 5 as positive sentiment*

*# score < 5 negative sentiment*

*# remove score = 5*

df = df[df['Score'] != 5]

df['sentiment'] = df['Score'].apply(**lambda** rating : +1 **if** rating > 5 **else** -1)

In [ ]:

df.head()

In [ ]:

!pip install wordcloud

In [ ]:

**from** **wordcloud** **import** WordCloud

The dataset is available in two forms

1.csv file 2.SQLite Database

In order to load the data, I have used the SQLITE dataset as it easier to query the data and visualise the data and results efficiently.

# Wordcloud

Word Cloud is a data visualization technique used for representing text data in which the size of each word indicates its frequency or importance. Significant textual data points can be highlighted using a word cloud.

In [ ]:

{"stream\_name":"stdout","time":"0:00:00.233244","data":"Loading required package: DBI**\n**"},{"stream\_name":"stdout","time":"0:00:00.237744","data":"Loading required package: methods**\n**"},{"stream\_name":"stdout","time":"0:00:00.641376","data":"Loading required package: NLP**\n**"},{"stream\_name":"stdout","time":"0:00:00.783783","data":"Loading required package: RColorBrewer**\n**"},{"stream\_name":"stdout","time":"0:00:34.382261","data":"null device **\n**"},{"stream\_name":"stdout","time":"0:00:34.382390","data":" 1 **\n**"},

In [ ]:

**from** **wordcloud** **import** WordCloud, STOPWORDS

stopwords = set(STOPWORDS)

df2=final\_data

plt.rcParams['figure.figsize']=(8.0,6.0) *#(6.0,4.0)*

figure(num=**None**, figsize=(12, 10), dpi=80, facecolor='w', edgecolor='k')

plt.rcParams['font.size']=12 *#10*

plt.rcParams['savefig.dpi']=100 *#72*

plt.rcParams['figure.subplot.bottom']=.1

**def** show\_wordcloud(data, title = **None**):

wordcloud = WordCloud(

background\_color='white',

stopwords=stopwords,

max\_words=200,

max\_font\_size=40,

scale=3,

random\_state=1 *# chosen at random by flipping a coin; it was heads*

).generate(str(data))

fig = plt.figure(1, figsize=(8, 8))

plt.axis('off')

**if** title:

fig.suptitle(title, fontsize=20)

fig.subplots\_adjust(top=2.3)

plt.imshow(wordcloud)

plt.show()

show\_wordcloud(df2['CleanedText'])

df2.loc[df2['SentimentPolarity'] == 'Positive']['CleanedText']

# Reading/Selecting Data

In [ ]:

*# using the SQLite Table to read data.*

con = sqlite3.connect('database.sqlite')

*#filtering only positive and negative reviews i.e.*

*# not taking into consideration those reviews with Score=3 or regarding them as neutral*

*# SELECT \* FROM Reviews WHERE Score != 3 LIMIT 500000, will give top 500000 data points*

*# you can change the number to any other number based on your computing power*

filtered\_data = pd.read\_sql\_query(""" SELECT \* FROM Reviews WHERE Score != 3 LIMIT 5000""", con)

**def** partition(x):

**if** x < 3:

**return** 0

**return** 1

*#changing reviews with score less than 3 to be positive and vice-versa*

actualScore = filtered\_data['Score']

positiveNegative = actualScore.map(partition)

filtered\_data['Score'] = positiveNegative

print("Number of data points in our data", filtered\_data.shape)

filtered\_data.head()

# Data Cleaning: Deleting duplicate values

It is observed that the reviews data have many duplicate entries. Hence it was necessary to remove duplicates in order to get unbiased results for the analysis of the data.

Helpfulness Numerator: Number of Peoples who found the review helpful to them.

Helpfulness Denominator: Number of Peoples indicated whether they found the review helpful or not.

In [ ]:

evaluate = pd.read\_sql\_query("""

SELECT \*

FROM Reviews

WHERE Score != 3 AND UserId="AR5J8UI46CURR"

ORDER BY ProductID

""", con)

evaluate.head()

In [ ]:

**from** **IPython.display** **import** Image

**from** **IPython.core.display** **import** HTML

Image(url= "https://miro.medium.com/max/756/1\*EyU641UXrBiTrdf10OlORg.png", width=600, height=600)

In [ ]:

**from** **IPython.display** **import** Image

**from** **IPython.core.display** **import** HTML

Image(url= "https://miro.medium.com/max/756/1\*qO48JZ3MamjxmusG3Qot9w.png", width=600, height=600)

We can see that in these two images the brand name brand is same for the products, only flavor is different.

so we have to make a code where the method ensures that there is only one representative for each product and deduplication without sorting would lead to possibility of different representatives still existing for the same product.

we can observe above that the same user has multiple reviews of the product with the same values for HelpfulnessDenominator,HelpfulnessNumerator, Score, Time, Text and Summary and on doing analysis it was found that

ProductId(B000HDOPZG),(B000PAQ75C) was Loacker Quadratini Vanilla Wafer Cookies(Pack of 8) and so on

In order to reduce redundancy and make it les complex it was decided to eliminate the rows having same parameters(except ProductId).

In [ ]:

*#Sorting data according to ProductId in ascending order*

sorted\_data=filtered\_data.sort\_values('ProductId', axis=0, ascending=**True**, inplace=**False**, kind='quicksort', na\_position='last')

In [ ]:

*#Deduplication of entries*

final\_result=sorted\_data.drop\_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first', inplace=**False**)

final\_result.shape

In [ ]:

*#how much % of data still remains*

(final\_result['Id'].size\*1.0)/(filtered\_data['Id'].size\*1.0)\*100

In [ ]:

final\_result = final\_result[final\_result.HelpfulnessNumerator<=final\_result.HelpfulnessDenominator]

*## the value of HelpfulnessNumerator is greater than HelpfulnessDenominator which is not possible hence these rows are removed from calcualtions.*

In [ ]:

*#the number of entries left*

print(final\_result.shape)

*#How many positive and negative reviews are present*

final\_result['Score'].value\_counts()

In [ ]:

stop=set(stopwords.words('english'))*#set of stop words*

sno=nltk.stem.SnowballStemmer('english') *#set of snow ball stemmers in english*

**def** cleanhtml(sentence): *#function to clean html tags in a sentence*

cleannr=re.compile('<.\*?>')

comptext=re.sub(cleannr,'',sentence)

**return** comptext

**def** cleanpunc(sentence) : *#function to clean punctuation in the sentence*

cleaned=re.sub(r'[? | ! | **\'** |" | #]',r'',sentence)

cleaned=re.sub(r'[. | , | ) | ( | \ | / ]' ,r' ',sentence)

**return** cleaned

print(stop)

print('\*\*' \* 50)

print(sno.stem('taste'))

In [ ]:

*#We will check the distribution of stemmed word lengths across the whole review dataset to understand what is the length of the maximum number of words we will consider for the word to be relevant.*

*#In other words we will keep only those words which has a length less than that of a speicific length (we will obtain this specific length from the histogram).*

total\_words = []

**for** review **in** tqdm(final\_data['Text'].values):

filtered\_sentence=[]

review = decontracted(review)

review = removeNumbers(review)

review = removeHtml(review)

review = removeURL(review)

review = removePunctuations(review)

review = removePatterns(review)

**for** cleaned\_words **in** review.split():

**if**((cleaned\_words **not** **in** custom\_stopwords)):

stemed\_word=(sno.stem(cleaned\_words.lower()))

total\_words.append(stemed\_word)

total\_words = list(set(total\_words)) *#Get list of unique words.*

*#A list to hold the length of each words used in all the reviews used across the whole dataset.*

dist = []

**for** i **in** tqdm(total\_words):

length = len(i)

dist.append(length)

*# matplotlib histogram to see the distribution of the length of words*

plt.figure(figsize=(20,10))

plt.hist(dist, color = 'red', edgecolor = 'blue', bins =90)

plt.title('Distribution of the length of Words across all reviews.')

plt.xlabel('Word Lengths')

plt.ylabel('Number of Words')

In [ ]:

*# remove urls from text python: https://stackoverflow.com/a/40823105/4084039*

*# printing some random reviews*

sent\_0 = final\_result['Text'].values[0]

print(sent\_0)

print("="\*50)

sent\_1000 = final\_result['Text'].values[1000]

print(sent\_1000)

print("="\*50)

sent\_1400 = final\_result['Text'].values[1400]

print(sent\_1400)

print("="\*50)

sent\_4900 = final\_result['Text'].values[4900]

print(sent\_4900)

print("="\*50)

In [ ]:

*# https://stackoverflow.com/a/47091490/4084039*

**import** **re**

**def** deactivate(phrase):

*# specific*

phrase = re.sub(r"won't", "will not", phrase)

phrase = re.sub(r"can\'t", "can not", phrase)

*# general*

phrase = re.sub(r"n\'t", " not", phrase)

phrase = re.sub(r"\'re", " are", phrase)

phrase = re.sub(r"\'s", " is", phrase)

phrase = re.sub(r"\'d", " would", phrase)

phrase = re.sub(r"\'ll", " will", phrase)

phrase = re.sub(r"\'t", " not", phrase)

phrase = re.sub(r"\'ve", " have", phrase)

phrase = re.sub(r"\'m", " am", phrase)

**return** phrase

In [ ]:

*# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all-tags-from-an-element*

**from** **bs4** **import** BeautifulSoup

soup = BeautifulSoup(sent\_0, 'lxml')

text = soup.get\_text()

print(text)

print("="\*50)

soup = BeautifulSoup(sent\_1000, 'lxml')

text = soup.get\_text()

print(text)

print("="\*50)

soup = BeautifulSoup(sent\_1400, 'lxml')

text = soup.get\_text()

print(text)

print("="\*50)

soup = BeautifulSoup(sent\_4900, 'lxml')

text = soup.get\_text()

print(text)

In [ ]:

sent\_1400 = deactivate(sent\_1400)

print(sent\_1400)

print("="\*50)

In [ ]:

*#remove words with numbers python: https://stackoverflow.com/a/18082370/4084039*

sent\_1400 = re.sub("\S\*\d\S\*", "", sent\_1400).strip()

print(sent\_1400)

In [ ]:

*#remove spacial character: https://stackoverflow.com/a/5843547/4084039*

sent\_1400 = re.sub('[^A-Za-z0-9]+', ' ', sent\_1400)

print(sent\_1400)

In [ ]:

*# https://gist.github.com/sebleier/554280*

*# we are removing the words from the stop words list: 'no', 'nor', 'not'*

*# <br /><br /> ==> after the above steps, we are getting "br br"*

*# we are including them into stop words list*

*# instead of <br /> if we have <br/> these tags would have revmoved in the 1st step*

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",\

"you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', \

'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their',\

'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', \

'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', \

'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', \

'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after',\

'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further',\

'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more',\

'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \

's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', \

've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn',\

"hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn',\

"mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", \

'won', "won't", 'wouldn', "wouldn't"])

In [ ]:

*#code for step by step implementing text preprocess*

*#this code is takes time because it needs to run on 500 k sentences.*

*## Combining all the above stundents*

**from** **tqdm** **import** tqdm *# tqdm is for printing the status bar*

i=0

str1=' '

preprocessed\_reviews=[]

all\_positive\_words=[] *#store words from +ve reviews here*

all\_negative\_words=[] *# store words from -ve reviews here*

s=''

**for** sentance **in** tqdm(final\_result['Text'].values):

filtered\_sentence=[]

*#print(sentance)*

sentance=cleanhtml(sentance)*#removing the html tags*

sentance = re.sub(r"http\S+", "", sentance)

sentance = BeautifulSoup(sentance, 'lxml').get\_text()

sentance = deactivate(sentance)

sentance = re.sub("\S\*\d\S\*", "", sentance).strip()

sentance = re.sub('[^A-Za-z]+', ' ', sentance)

*# https://gist.github.com/sebleier/554280*

*# sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not in stopwords)*

**for** w **in** sentance.split():

**for** cleaned\_words **in** cleanpunc(w).split():

**if**((cleaned\_words.isalpha()) & (len(cleaned\_words) > 2)):

**if** (cleaned\_words.lower() **not** **in** stopwords):

s=(sno.stem(cleaned\_words.lower())).encode('utf8')

filtered\_sentence.append(s)

**if** (final\_result['Score'].values[i]) == 1 :

all\_positive\_words.append(s) *#list all the positive words*

**if** (final\_result['Score'].values[i]) == 0 :

all\_negative\_words.append(s) *#list all the negative words*

**else**:

**continue**

**else**:

**continue**

*#print(filtered\_sentence)*

str1=b" ".join(filtered\_sentence) *#final string of the filtered sentence*

preprocessed\_reviews.append(str1.strip())

i+=1

In [ ]:

final\_result['preprocessed\_reviews']=preprocessed\_reviews *#adding cleaned text to the*

final\_result['preprocessed\_reviews']=final\_result['preprocessed\_reviews'].str.decode("utf-8")

final\_result['preprocessed\_reviews'].iloc[0]

In [ ]:

final\_result.to\_csv('preprocessed\_reviews',index=**False**)

In [ ]:

*#conn=sqlite3.connect('final.sqlite')*

*#c=conn.cursor()*

*#conn.text\_factory=str*

*#final.to\_sql('pre\_text',conn,schema=None,if\_exists='replace')*

In [ ]:

*#code for step by step implementing text preprocess*

*#this code is takes time because it needs to run on 50000 sentences.*

**from** **tqdm** **import** tqdm *# tqdm is for printing the status bar*

i=0

str1=' '

preprocessed\_summary=[]

all\_positive\_summary=[] *#store words from +ve reviews here*

all\_negative\_summary=[] *# store words from -ve reviews here*

s=''

**for** sentance **in** tqdm(final\_result['Summary'].values):

filtered\_summary=[]

*#print(sentance)*

sentance=cleanhtml(sentance)*#removing the html tags*

sentance = re.sub(r"http\S+", "", sentance)

sentance = BeautifulSoup(sentance, 'lxml').get\_text()

sentance = deactivate(sentance)

sentance = re.sub("\S\*\d\S\*", "", sentance).strip()

sentance = re.sub('[^A-Za-z]+', ' ', sentance)

*# https://gist.github.com/sebleier/554280*

*# sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not in stopwords)*

**for** w **in** sentance.split():

**for** cleaned\_words **in** cleanpunc(w).split():

**if**((cleaned\_words.isalpha()) & (len(cleaned\_words) > 2)):

**if** (cleaned\_words.lower() **not** **in** stopwords):

s=(sno.stem(cleaned\_words.lower())).encode('utf8')

filtered\_summary.append(s)

**if** (final\_result['Score'].values[i]) == 1 :

all\_positive\_summary.append(s) *#list all the positive words*

**if** (final\_result['Score'].values[i]) == 0 :

all\_negative\_summary.append(s) *#list all the negative words*

**else**:

**continue**

**else**:

**continue**

*#print(filtered\_sentence)*

str1=b" ".join(filtered\_summary) *#final string of the filtered sentence*

preprocessed\_summary.append(str1.strip())

i+=1

In [ ]:

final\_result['preprocessed\_summary']=preprocessed\_summary *#adding cleaned text to the*

final\_result['preprocessed\_summary']=final\_result['preprocessed\_summary'].str.decode("utf-8")

final\_result['preprocessed\_summary'].iloc[0]

In [ ]:

final\_result.to\_csv('preprocessed\_data.csv',index=**False**)

In [ ]:

*#store all in to database for future*

*#conn=sqlite3.connect('preprocessed.sqlite')*

*#c=conn.cursor()*

*#conn.text\_factory=str*

*#final.to\_sql('preprocessed\_data',conn,schema=None,if\_exists='replace')*

In [ ]:

all\_positive\_summary[0]

In [ ]:

all\_positive\_words[0:10]

In [ ]:

all\_negative\_words[0:10]

In [ ]:

all\_negative\_summary[0:10]

In [ ]:

df.Score.value\_counts().sort\_index().plot.bar(alpha=0.7, grid=**True**, color = 'orange', width = 0.9)

plt.xlabel('Score')

plt.ylabel('Number Of Reviews')

plt.title('Distribution of reviews over each score')

plt.show()

It is clear from the plot that more than 80000 reviews have given 4 score and around 30000 have given 2 scores.

In [ ]:

*# How many empty length texts are present in the dataset()*

df[df['Text']==0].Text.count()

this means there are no empty rows of text content

In [ ]:

**import** **re**

**from** **nltk.corpus** **import** stopwords

stopwords\_en = set(stopwords.words('english'))

ps = PorterStemmer()

In [ ]:

**def** clean\_text(text):

text = text.lower() *#converting text to lowercase*

text = ' '.join([i **for** i **in** nltk.word\_tokenize(text) **if** i **not** **in** stopwords\_en **and** i **not** **in** string.punctuation]) *#stopword and punct removal*

text = re.sub('[^a-z]+', ' ', text) *#removal of anything other than English letters*

text = ' '.join([ps.stem(i) **for** i **in** nltk.word\_tokenize(text)]) *#stemming*

**return** text

# Data Preprocessing

Here we will apply all the basic steps that I have discussed above and finally we collect the words used to describe positive and negative reviews

In [ ]:

*# printing some random reviews*

sent\_0 = final\_result['Text'].values[0]

print(sent\_0)

print("="\*50)

sent\_1001 = final\_result['Text'].values[1000]

print(sent\_1001)

print("="\*50)

sent\_1400 = final\_result['Text'].values[1500]

print(sent\_1400)

print("="\*50)

sent\_4560 = final\_result['Text'].values[4900]

print(sent\_4560)

print("="\*50)

In [ ]:

**import** **re**

**def** deactivate(phrase):

*# specific*

phrase = re.sub(r"won't", "will not", phrase)

phrase = re.sub(r"can\'t", "can not", phrase)

*# general*

phrase = re.sub(r"n\'t", " not", phrase)

phrase = re.sub(r"\'re", " are", phrase)

phrase = re.sub(r"\'s", " is", phrase)

phrase = re.sub(r"\'d", " would", phrase)

phrase = re.sub(r"\'ll", " will", phrase)

phrase = re.sub(r"\'t", " not", phrase)

phrase = re.sub(r"\'ve", " have", phrase)

phrase = re.sub(r"\'m", " am", phrase)

**return** phrase

In [ ]:

*# we are removing the words from the stop words list: 'no', 'nor', 'not'*

*# <br /><br /> ==> after the above steps, we are getting "br br"*

*# we are including them into stop words list*

*# instead of <br /> if we have <br/> these tags would have removed in the 1st step*

In [ ]:

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",\

"you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', \

'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their',\

'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', \

'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', \

'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', \

'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after',\

'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further',\

'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more',\

'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \

's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', \

've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn',\

"hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn',\

"mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", \

'won', "won't", 'wouldn', "wouldn't"])

In [ ]:

*# Data cleaning for all the reviews*

**from** **bs4** **import** BeautifulSoup *#Beautiful Soup is a Python library for pulling data out of HTML and XML files. It works with your favorite parser to provide idiomatic ways of navigating, searching, and modifying the parse tree.*

*#It commonly saves programmers hours or days of work.*

**from** **tqdm** **import** tqdm

pp\_reviews = []

*# tqdm is for printing the status bar*

**for** sentence **in** tqdm(final\_result['Text'].values):

sentence = re.sub(r"http\S+", "", sentence) *#remove urls*

sentence = BeautifulSoup(sentence, 'lxml').get\_text() *#remove xml tags*

sentence = deactivate(sentence)

sentence = re.sub("\S\*\d\S\*", "", sentence).strip() *#remove words with numbers*

sentence = re.sub('[^A-Za-z]+', ' ', sentence) *#remove special characters*

sentence = ' '.join(e.lower() **for** e **in** sentence.split() **if** e.lower() **not** **in** stopwords)

pp\_reviews.append(sentence.strip())

In [ ]:

pp\_reviews[0]

In [ ]:

len(pp\_reviews)

# Featurization

In [ ]:

**from** **numpy** **import** zeros

**import** **keras**

**import** **tensorflow** **as** **tf**

**from** **keras.preprocessing.text** **import** Tokenizer

**from** **keras.preprocessing.sequence** **import** pad\_sequences

**from** **keras.models** **import** Sequential

**from** **keras.callbacks** **import** EarlyStopping

**from** **keras.layers.embeddings** **import** Embedding

**from** **keras.layers** **import** SpatialDropout1D

In [ ]:

*# prepare tokenizer*

t = Tokenizer()

t.fit\_on\_texts(pp\_reviews)

vocab\_size = len(t.word\_index) + 1

In [ ]:

*# integer encode the documents*

code\_docs = t.texts\_to\_sequences(pp\_reviews)

In [ ]:

*# pad documents to a max length of max words*

max\_length = 150

pad\_docs = pad\_sequences(code\_docs, maxlen=max\_length, padding='post')

In [ ]:

label = final\_result['Score']

In [ ]:

pad\_docs[1500]

# Prepare Model

In [ ]:

**import** **re**

**from** **nltk.corpus** **import** stopwords

stopwords\_en = set(stopwords.words('english'))

ps = PorterStemmer()

In [ ]:

**def** clean\_text(text):

text = text.lower() *#converting text to lowercase*

text = ' '.join([i **for** i **in** nltk.word\_tokenize(text) **if** i **not** **in** stopwords\_en **and** i **not** **in** string.punctuation]) *#stopword and punct removal*

text = re.sub('[^a-z]+', ' ', text) *#removal of anything other than English letters*

text = ' '.join([ps.stem(i) **for** i **in** nltk.word\_tokenize(text)]) *#stemming*

**return** text

In [ ]:

*# Apply the cleanup and create a new column*

df['FilterText'] = df['Text'].apply(**lambda** x: clean\_text(x))

df.head(3)

In [ ]:

**def** partition(val):

**if**(val>2):

**return** 1

**return** 0

df['Positivity']=df['Score'].apply(**lambda** x: partition(x))

df.head(3)

In [ ]:

required\_columns = ['FilterText', 'Positivity']

df = df[required\_columns]

df.head()

In [ ]:

df.Positivity.value\_counts().plot.bar(alpha=0.5, grid=**True**)

plt.title('Distribution Of Positive & Negative Feedback')

plt.ylabel('Counts')

plt.show()

Around 120000 people have given positive reviews(1) and around 80000 people have given negative reviews.

In [ ]:

*# Create Train/Test data split model*

**from** **sklearn.model\_selection** **import** train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df['FilterText'], df['Positivity'], test\_size=0.25, random\_state=42, shuffle=**True**, stratify=df['Positivity'])

print("Train Set Size = **{}\n**Test Set Size = **{}**".format(X\_train.shape[0], X\_test.shape[0]))

Long short-term memory (LSTM) is an artificial recurrent neural network (RNN) architecture used in the field of deep learning. LSTM networks are well-suited to classifying, processing and making predictions based on time series data, since there can be lags of unknown duration between important events in a time series.

In [ ]:

*#BoW*

count\_vect = CountVectorizer() *#in scikit-learn*

count\_vect.fit(preprocessed\_reviews)

print("some feature names ", count\_vect.get\_feature\_names()[500:510])

print('='\*50)

final\_counts = count\_vect.transform(pp\_reviews)

print("the type of count vectorizer ",type(final\_counts))

print("the shape of out text BOW vectorizer ",final\_counts.get\_shape())

print("the number of unique words ", final\_counts.get\_shape()[1])

In [ ]:

data=pd.read\_csv('preprocessed\_data.csv')

data.head()

In [ ]:

data['preprocessed\_reviews'].isnull().sum()

In [ ]:

*#bi-gram, tri-gram and n-gram*

*#removing stop words like "not" should be avoided before building n-grams*

*# count\_vect = CountVectorizer(ngram\_range=(1,2))*

*# you can choose these numbers min\_df=10, max\_features=5000*

count\_vect = CountVectorizer(ngram\_range=(1,2), min\_df=10, max\_features=5000)

final\_bigram\_counts = count\_vect.fit\_transform(data['preprocessed\_reviews'])[:5000]

print("the type of count vectorizer ",type(final\_bigram\_counts))

print("the shape of out text BOW vectorizer ",final\_bigram\_counts.get\_shape())

print("the number of unique words including both unigrams and bigrams ", final\_bigram\_counts.get\_shape()[1])

# TF-IDF(Term frequency -inverse document frequency):

TF-IDF stands for “Term Frequency — Inverse Document Frequency”. This is a technique to quantify a word in documents, we generally compute a weight to each word which signifies the importance of the word in the document and corpus. This method is a widely used technique in Information Retrieval and Text Mining.

In [ ]:

tf\_idf\_vect = TfidfVectorizer(ngram\_range=(1,2), min\_df=10)

tf\_idf\_vect.fit(data['preprocessed\_reviews'])

print("some sample features(unique words in the corpus)",tf\_idf\_vect.get\_feature\_names()[0:10])

print('='\*50)

final\_tf\_idf = tf\_idf\_vect.transform(data['preprocessed\_reviews'])

print("the type of count vectorizer ",type(final\_tf\_idf))

print("the shape of out text TFIDF vectorizer ",final\_tf\_idf.get\_shape())

print("the number of unique words including both unigrams and bigrams ", final\_tf\_idf.get\_shape()[1])

# word2vec:

This technique is the state of the art algorithm, it consider the semantic meaning of the word. If we give the word it converts in to vectors. It also learns relationship automatically from the text. The output of the word2vec model is Dense vectors.Word2vec model requires large text corpus.

In [ ]:

data=pd.read\_csv('preprocessed\_data.csv')

data['preprocessed\_reviews']=data['preprocessed\_reviews'].fillna(method='bfill')

data['preprocessed\_reviews'].isnull().sum()

In [ ]:

*# Train your own Word2Vec model using your own text corpus*

i=0

list\_of\_sentance=[]

**for** sentance **in** data['preprocessed\_reviews']:

list\_of\_sentance.append(sentance.split())

In [ ]:

list\_of\_sentance[0]

In [ ]:

*# Using Google News Word2Vectors*

*# in this project we are using a pretrained model by google*

*# a pickle file wich contains a dict ,*

*# and it contains all our courpus words as keys and model[word] as values*

*# To use this code-snippet, download "GoogleNews-vectors-negative300.bin"*

*# from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit*

is\_your\_ram\_gt\_16g=**False**

want\_to\_use\_google\_w2v = **False**

want\_to\_train\_w2v = **True**

**if** want\_to\_train\_w2v:

*# min\_count = 5 considers only words that occured atleast 5 times*

w2v\_model=Word2Vec(list\_of\_sentance,min\_count=5, workers=4)

print(w2v\_model.wv.most\_similar('great'))

print('='\*50)

print(w2v\_model.wv.most\_similar('worst'))

**elif** want\_to\_use\_google\_w2v **and** is\_your\_ram\_gt\_16g:

**if** os.path.isfile('GoogleNews-vectors-negative300.bin'):

w2v\_model=KeyedVectors.load\_word2vec\_format('GoogleNews-vectors-negative300.bin', binary=**True**)

print(w2v\_model.wv.most\_similar('great'))

print(w2v\_model.wv.most\_similar('worst'))

**else**:

print("you don't have gogole's word2vec file, keep want\_to\_train\_w2v = True, to train your own w2v ")

In [ ]:

**from** **sklearn.feature\_extraction.text** **import** TfidfVectorizer

tfidf\_vectoriser = TfidfVectorizer()

*# Training % Feature Extraction On Entire Dataset, Used For Cross Validation & Model Comparison*

features = tfidf\_vectoriser.fit\_transform(df['FilterText'])

labels = df['Positivity'].astype(int)

In [ ]:

*# Training On Only Train Set Now*

tfidf\_vectoriser.fit(X\_train)

X\_train\_tf = tfidf\_vectoriser.transform(X\_train)

X\_test\_tf = tfidf\_vectoriser.transform(X\_test)

X\_train\_tf.shape, X\_test\_tf.shape

In [ ]:

**import** **random**

print("Twenty Random Words from Training Set ...**\n**",\*random.sample(tfidf\_vectoriser.get\_feature\_names(),20))

In [ ]:

**from** **sklearn.naive\_bayes** **import** MultinomialNB *#The multinomial Naive Bayes classifier is suitable for classification with discrete features (e.g., word counts for text classification). The multinomial distribution normally requires integer feature counts. However, in practice, fractional counts such as tf-idf may also work. Parameters alphafloat, default=1.0.*

**from** **sklearn.svm** **import** LinearSVC *#The objective of a Linear SVC (Support Vector Classifier) is to fit to the data you provide, returning a "best fit" hyperplane that divides, or categorizes, your data*

**from** **sklearn.model\_selection** **import** cross\_val\_score *#Cross-validation is a statistical method used to estimate the skill of machine learning models.That k-fold cross validation is a procedure used to estimate the skill of the model on new data. There are common tactics that you can use to select the value of k for your dataset.*

**from** **sklearn.linear\_model** **import** LogisticRegression *#Logistic Regression is a Machine Learning classification algorithm that is used to predict the probability of a categorical dependent variable.*

**from** **sklearn.ensemble** **import** RandomForestClassifier *#A random forest classifier. A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.*

In [ ]:

parameters = [

LinearSVC(),

MultinomialNB(),

LogisticRegression(random\_state=0),

RandomForestClassifier(n\_estimators=200, n\_jobs=-1, random\_state=0)

]

CV = 5

cv\_df = pd.DataFrame(index=range(CV \* len(parameters)))

entries = []

**for** model **in** parameters:

model\_name = model.\_\_class\_\_.\_\_name\_\_

accuracies = cross\_val\_score(model, features ,labels, scoring='accuracy', cv=CV)

**for** fold\_idx, accuracy **in** enumerate(accuracies):

entries.append((model\_name, fold\_idx, accuracy))

In [ ]:

**from** **sklearn.model\_selection** **import** GridSearchCV

param\_grid = {

'C':[0.5,0.8,1.0,1.5]

}

svm = LinearSVC(max\_iter=1500)

svm\_cv = GridSearchCV(svm, param\_grid, cv=5)

svm\_cv.fit(features, labels)

print("Best Parameters :", svm\_cv.best\_params\_)

print("Amazing Score :",svm\_cv.best\_score\_)

In [ ]:

svm = LinearSVC(C=0.5, max\_iter=2000)

svm.fit(X\_train\_tf, y\_train)

In [ ]:

**from** **sklearn.metrics** **import** classification\_report, accuracy\_score, confusion\_matrix, roc\_auc\_score, roc\_curve

y\_pred = svm.predict(X\_test\_tf)

print(classification\_report(y\_test, y\_pred, target\_names=['Negative','Positive']))

print("Accuracy :",accuracy\_score(y\_test, y\_pred), end='**\n\n**')

conf\_mat = confusion\_matrix(y\_test, y\_pred)

print(conf\_mat)

Preprocessing the data using Spacy and Machine learning model training using sklearn

In this stage, Spacy package of python is used to lemmatize and remove stop words from the obtained dataset.

In [ ]:

**from** **spacy.lang.en.stop\_words** **import** STOP\_WORDS

*# To build a list of stop words for filtering*

stopwords = list(STOP\_WORDS)

print(stopwords)

Thus, the stop words have been enlisted

The data is initially split into test and training datasets prior to feeding into the machine learning pipeline. Then, a class object was defined as 'sent\_predict' is used as the first step of the pipeline which would inherit from the TransformerMixin package and perform the cleaning of data. The second method of the pipeline is to vectorize the cleaned data.

Tokenized words needs to be lemmatized and filtered for pronouns, stopwords and punctuations using the defined method 'my\_tokenizer' For that purpose count vectorizeor and tfidfVectorizer both have been tried subsequently to decide which is better.

Then the third step of the pipeline is the defining of the classifier. In this case, Linear Support Vector Machine classifier was chosen. Other methods could be explored in the furture

In [ ]:

**import** **string**

punctuations = string.punctuation

*# Creating a Spacy Parser*

**from** **spacy.lang.en** **import** English

parser = English()

In [ ]:

**def** my\_tokenizer(sentence):

mytokens = parser(sentence)

mytokens = [ word.lemma\_.lower().strip() **if** word.lemma\_ != "-PRON-" **else** word.lower\_ **for** word **in** mytokens ]

mytokens = [ word **for** word **in** mytokens **if** word **not** **in** stopwords **and** word **not** **in** punctuations ]

**return** mytokens

In [ ]:

**from** **sklearn.metrics** **import** accuracy\_score

**from** **sklearn.base** **import** TransformerMixin

**from** **sklearn.pipeline** **import** Pipeline

In [ ]:

*# Create the pipeline to clean, tokenize, vectorize, and classify using"Count Vectorizor"*

pipe\_countvect = Pipeline([("cleaner", predictors()),

('vectorizer', vectorizer),

('classifier', classifier)])

*# Fit our data*

pipe\_countvect.fit(X\_train,y\_train)

*# Predicting with a test dataset*

sample\_prediction = pipe\_countvect.predict(X\_test)

In [ ]:

*#Custom transformer using spaCy*

**class** **predictors**(TransformerMixin):

**def** transform(self, X, \*\*transform\_params):

**return** [clean\_text(text) **for** text **in** X]

**def** fit(self, X, y, \*\*fit\_params):

**return** self

**def** get\_params(self, deep=**True**):

**return** {}

*# Basic function to clean the text*

**def** clean\_text(text):

**return** text.strip().lower()

In [ ]:

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

**from** **sklearn.metrics** **import** accuracy\_score

**from** **sklearn.base** **import** TransformerMixin

**from** **sklearn.pipeline** **import** Pipeline

**from** **sklearn.svm** **import** LinearSVC

In [ ]:

*# Vectorization*

vectorizer = CountVectorizer(tokenizer = my\_tokenizer, ngram\_range=(1,1))

classifier = LinearSVC()

In [ ]:

*# Using Tfidf*

tfvectorizer = TfidfVectorizer(tokenizer = my\_tokenizer)

In [ ]:

df1 = pd.read\_csv('pp\_reviews.csv')

In [ ]:

**from** **sklearn.model\_selection** **import** train\_test\_split

In [ ]:

*# Features and Labels*

X = df1['Text']

ylabels = df1['Score']

In [ ]:

pipe = Pipeline([("cleaner", predictors()),

('vectorizer', vectorizer),

('classifier', classifier)])

In [ ]:

pipe.fit(X\_train,y\_train)

In [ ]:

sample\_prediction = pipe.predict(X\_test)

In [ ]:

**def** removeHTMLTags(review):

soup = BeautifulSoup(review, 'lxml')

**return** soup.get\_text()

In [ ]:

**def** removeApostrophe(review):

phrase = re.sub(r"won't", "will not", review)

phrase = re.sub(r"can\'t", "can not", review)

phrase = re.sub(r"n\'t", " not", review)

phrase = re.sub(r"\'re", " are", review)

phrase = re.sub(r"\'s", " is", review)

phrase = re.sub(r"\'d", " would", review)

phrase = re.sub(r"\'ll", " will", review)

phrase = re.sub(r"\'t", " not", review)

phrase = re.sub(r"\'ve", " have", review)

phrase = re.sub(r"\'m", " am", review)

**return** phrase

**def** removeAlphaNumericWords(review):

**return** re.sub("\S\*\d\S\*", "", review).strip()

**def** removeSpecialChars(review):

**return** re.sub('[^a-zA-Z]', ' ', review)

In [ ]:

**def** doTextCleaning(review):

review = removeHTMLTags(review)

review = removeApostrophe(review)

review = removeAlphaNumericWords(review)

review = removeSpecialChars(review)

*# Lower casing*

review = review.lower()

*#Tokenization*

review = review.split()

*#Removing Stopwords and Lemmatization*

lmtzr = WordNetLemmatizer()

review = [lmtzr.lemmatize(word, 'v') **for** word **in** review **if** **not** word **in** set(stopwords.words('english'))]

review = " ".join(review)

**return** review

In [ ]:

**def** scorePartition(x):

**if** x < 3:

**return** 0

**return** 1

In [ ]:

final=final\_result.sort\_values(by=['Time'],ascending=**False**)

finalDataPoints=final\_result.head(100000)

x=finalDataPoints["Text"]

y=finalDataPoints["Score"]

x\_tr,x\_test,y\_tr,y\_test=train\_test\_split(x, y, test\_size=0.2,shuffle=**False**)

In [ ]:

*#BOW for unigram*

bow = CountVectorizer()

x\_tr\_uni = bow.fit\_transform(x\_tr)

x\_test\_uni= bow.transform(x\_test)

**from** **sklearn.preprocessing** **import** StandardScaler

x\_tr\_uni = StandardScaler(with\_mean = **False**).fit\_transform(x\_tr\_uni)

x\_test\_uni = StandardScaler(with\_mean = **False**).fit\_transform(x\_test\_uni)

In [ ]:

pip install sklearn 0.21.3

In [ ]:

pip install best\_score

In [ ]:

LR = LogisticRegression(penalty='l2')

C\_value = [{'C': [10\*\*-4, 10\*\*-2, 10\*\*0, 10\*\*2, 10\*\*4]}]

gsv = GridSearchCV(LR,C\_value,cv=5,verbose=1,scoring='f1\_weighted')

gsv.fit(x\_tr\_uni,y\_tr)

print("Best HyperParameter: ",gsv.best\_params\_)

print(gsv.best\_score)

optimal\_C=gsv.best\_score

x=[]

y=[]

plt.figure(figsize=(8,8))

**for** a **in** gsv.cv\_scores\_:

x.append(a[0]['C'])

y.append(a[1])

plt.xlabel("C",fontsize=15)

plt.ylabel("f1\_weighted")

plt.title('f1\_weighted v/s C')

plt.plot(x,y, marker='o', markerfacecolor='red', markersize=10)

plt.show()

In [ ]:

*#Plot the performance of model both on train data and cross validation data for each hyper parameter. Display the confusion matrix as well as ROC Curve*

*#From the heatmaps above, we will select the optimal value of max\_depth to be 25 and min\_samples\_split to be 500.*

max\_depth = 25

min\_samples\_split=500,

best\_estimator = DecisionTreeClassifier(criterion='gini', max\_depth=25, min\_samples\_split=500, random\_state=0, splitter='best')

vectorizationType="TFIDF"

trained\_clf\_TFIDF = performance(best\_estimator, vectorizationType, X\_train, y\_train, X\_test, y\_test, X\_calib, y\_calib, max\_depth, min\_samples\_split)

In [ ]:

Call the function above **and** **pass** a filename onto it.

f\_names=tf\_idf\_obj.get\_feature\_names()

graph=visualize\_tree(trained\_clf\_TFIDF, f\_names, 'TFIDF\_DT.png')

graph

In [ ]:

*#Load the W2V Vectors we had created earlier and standardize them. We will standardize the traina and test data seperately in order to prevent data leakage.*

**import** **pickle**

**with** open('X\_train\_W2V.pkl', 'rb') **as** file:

X\_train = pickle.load(file)

**with** open('X\_test\_W2V.pkl', 'rb') **as** file:

X\_test = pickle.load(file)

**with** open('y\_train\_W2V.pkl', 'rb') **as** file:

y\_train = pickle.load(file)

**with** open('y\_test\_W2V.pkl', 'rb') **as** file:

y\_test = pickle.load(file)

**with** open('X\_calib\_W2V.pkl', 'rb') **as** file:

X\_calib = pickle.load(file)

**with** open('y\_calib\_W2V.pkl', 'rb') **as** file:

y\_calib = pickle.load(file)

print("Shape of the train data matrix: ",X\_train.shape)

print("Shape of the test data matrix: ",X\_test.shape)

print("Shape of the calibration data matrix: ",X\_calib.shape)

*#Perform Grid Search cross validation to obtain the best value of the hyperparameter.*

vectorizationType = "AVG-W2V"

st=datetime.now()

gsearch\_cv = get\_GridSearchCV(vectorizationType, X\_train, y\_train, X\_test, y\_test)

print("**\n**Time taken to complete Hyperparameter Search: ",datetime.now()-st)

In [ ]:

*#Plot the performance of model both on train data and cross validation data for each hyper parameter. Display the confusion matrix as well as ROC Curve*

*#From the heatmaps above, we will select the optimal value of max\_depth to be 10 and min\_samples\_split to be 250.*

max\_depth = 10

min\_samples\_split=250,

best\_estimator = DecisionTreeClassifier(criterion='gini', max\_depth=10, min\_samples\_split=250, random\_state=0, splitter='best')

vectorizationType="AVG-W2V"

trained\_clf\_avg\_w2v = performance(best\_estimator, vectorizationType, X\_train, y\_train, X\_test, y\_test, X\_calib, y\_calib, max\_depth, min\_samples\_split)

In [ ]:

*#Load the TF-IDF W2V Vectors we had created earlier and standardize them. We will standardize the traina and test data seperately in order to prevent data leakage.*

**import** **pickle**

**with** open('X\_train\_TFIDF-W2V.pkl', 'rb') **as** file:

X\_train = pickle.load(file)

**with** open('X\_test\_TFIDF-W2V.pkl', 'rb') **as** file:

X\_test = pickle.load(file)

**with** open('y\_train\_TFIDF-W2V.pkl', 'rb') **as** file:

y\_train = pickle.load(file)

**with** open('y\_test\_TFIDF-W2V.pkl', 'rb') **as** file:

y\_test = pickle.load(file)

**with** open('X\_calib\_TFIDF-W2V.pkl', 'rb') **as** file:

X\_calib = pickle.load(file)

**with** open('y\_calib\_TFIDF-W2V.pkl', 'rb') **as** file:

y\_calib = pickle.load(file)

print("Shape of the train data matrix: ",X\_train.shape)

print("Shape of the test data matrix: ",X\_test.shape)

print("Shape of the calibration data matrix: ",X\_calib.shape)

*#Perform Grid Search cross validation to obtain the best value of the hyperparameter.*

vectorizationType = "TFIDF-W2V"

st=datetime.now()

gsearch\_cv = get\_GridSearchCV(vectorizationType, X\_train, y\_train, X\_test, y\_test)

print("**\n**Time taken to complete Hyperparameter Search: ",datetime.now()-st)

In [ ]:

*#Plot the performance of model both on train data and cross validation data for each hyper parameter. Display the confusion matrix as well as ROC Curve*

*#From the heatmaps above, we will select the optimal value of max\_depth to be 10 and min\_samples\_split to be 500.*

max\_depth = 10

min\_samples\_split=500,

best\_estimator = DecisionTreeClassifier(criterion='gini', max\_depth=10, min\_samples\_split=500, random\_state=0, splitter='best')

vectorizationType="TFIDF-W2V"

trained\_clf\_avg\_w2v = performance(best\_estimator, vectorizationType, X\_train, y\_train, X\_test, y\_test, X\_calib, y\_calib, max\_depth, min\_samples\_split)

In [ ]:

Compare performance **and** display it on a pretty table.

**from** **prettytable** **import** PrettyTable

table = PrettyTable()

table.field\_names = ["Model", "Max Depth", "Min Samples Split", "Accuracy on Test data", "AUC Score Test Data", "No. Of accurate predictions"]

print("Please find below the important metrics for all the models below.**\n**")

file = open('info\_model\_DT.txt', 'r')

file.seek(0)

**for** line **in** file:

table.add\_row(line.split())

print(table)

## Applying various Machine Learning models

### 1.Logistic Regression

#### L1 regulization

#### Conclusion

| **Model** | **hyper parameter** | **F1score test** | **accuracy Test** |
| --- | --- | --- | --- |
| unigram | 1 | 0.293 | 94.723% |
| Bi-gram | 1 | 0.912 | 80.652% |
| Tf-IDF | 1 | 0.833 | 81.262% |
| Av-Word2Vec | 100 | 0.793 | 88.561% |
| Tf-IDF Word2vec | 0.0001 | 0.698 | 74.700% |

#### L2 regulization

#### Conclusion

| **Model** | **hyper parameter** | **F1score test** | **accuracy Test** |
| --- | --- | --- | --- |
| unigram | 0.0001 | 0.777 | 86.7% |
| Bi-gram | 1 | 0.692 | 87.210% |
| Tf-IDF | 10000 | 0.87 | 92.715% |
| Av-Word2Vec | 100 | 0.72 | 88.7% |

### 2.Decision Tree

#### Conclusion

| **Model** | **hyper parameter** | **F1score test** | **accuracy Test** |
| --- | --- | --- | --- |
| unigram | 65 | 0.500 | 84.635% |
| Bi-gram | 71 | 0.423 | 81.275% |
| Tf-IDF | 65 | 0.437 | 82.600% |
| Av-Word2Vec | 8 | 0.980 | 86.75% |
| Tf-IDF Word2vec | 6 | 0.686 | 80.66% |

### 3.RandomForest

#### Conclusion

| **Model** | **Max-Depth** | **n\_estimator** | **F1score test** | **Accuracy Test** |
| --- | --- | --- | --- | --- |
| unigram | 150 | 12 | 0.713 | 87.030% |
| Bi-gram | 150 | 4 | 0.566 | 83.510% |
| Tf-IDF | 120 | 4 | 0.636 | 84.35% |
| Av-Word2Vec | 40 | 16 | 0.461 | 82.305% |
| Tf-IDF Word2vec | 10 | 12 | 0.559 | 84.115% |

### 4.SVM

#### Conclusion

| **Model** | **alpha** | **Test Roc Auc** | **Train Roc Auc** | **F1-score** |
| --- | --- | --- | --- | --- |
| unigram | 0.001 | 0.933 | 0.940 | 0.824 |
| Bi-gram | 0.001 | 0.942 | 0.950 | 0.851 |
| Tf-IDF | 0.00003 | 0.937 | 0.941 | 0.872 |
| Av-Word2Vec | 0.002 | 0.874 | 0.905 | 0.739 |
| Tf-IDF Word2vec | 0.001 | 0.938 | 0.847 | 0.733 |

## **Conclusion**

•**Unigram**-Use NB it give the F1-score of 0.834.  
•**Bigram**-Use NB it give the F1-score of 0.71 or Logistic Regression it give the F1-score of 0.872 with L1 regulization  
•**Tf-idf**-Use GBDT it give the F1-score of 0.812 or Logistic Regression it give the F1-score of 0.826 with L1 regulization  
•**Av-Word2Vec**-Use GBDT it give the F1-score of 0.767 or Logistic Regression it give the F1-score of 0.04437 with L1 regulization  
•**Tf-IDF Word2vec**-Use GBDT it give the F1-score of 0.798

**System Requirements**

**Operating system:** Windows 7 or newer, 64-bitmacOS 10.13+, or Linux, including Ubuntu, RedHat, CentOS 6+, and others.

**System architecture:** Windows- 64-bit x86, 32-bitx86; MacOS- 64-bit x86; Linux- 64-bit x86, 64-bitPower8/Power9.

**Disk Space:** Minimum 5 GB disk space to download and install anaconda distribution.

**RAM**: 2 GB RAM recommended.

**Graphics:** For neural networks in Machine Learning, better graphics cards will yield faster results with some high-end graphics cards created especially for Machine Learning purposes. However, Google Colab or some similar website can be used to perform same tasks using cloud computing, without needing a graphics card.