**Phase 5: Project Documentation & Submission**

**1. Introduction:**

In today's highly competitive business landscape, understanding customer sentiments and feedback is crucial for companies aiming to gain a competitive edge and improve their products and services. Customer feedback is a valuable source of information that can provide insights into the strengths and weaknesses of both your own and competing products. Sentiment analysis, a subfield of Natural Language Processing (NLP), plays a pivotal role in extracting these insights by automatically categorizing text data into positive, negative, or neutral sentiments.

**1.1 Problem Statement:**

The primary objective of this project is to perform sentiment analysis on customer feedback data to gain valuable insights into competitor products. Specifically, we aim to:

Automatically classify customer feedback as positive, negative, or neutral.

Identify the key strengths and weaknesses of competing products based on sentiment patterns.

Provide actionable recommendations to improve our own product offerings.

**1.2 Design Thinking Process:**

In our quest to achieve these goals, we have adopted a design thinking approach that guides our project through stages of empathy, definition, ideation, prototyping, and testing. This process ensures that our analysis is not only technically sound but also aligns with the needs and perspectives of our customers.

**1.3 Phases of Development:**

To address the problem at hand, we have divided the development process into several distinct phases:

Data Collection: We have utilized a publicly available dataset, the Twitter Airline Sentiment dataset from Kaggle, containing tweets related to airline experiences. This dataset is a valuable source of customer feedback.

Data Preprocessing: Before performing sentiment analysis, we have taken steps to clean and preprocess the data. This includes handling missing data, removing duplicates, and cleaning the text.

Feature Extraction: We have employed various techniques such as Bag of Words (BoW), TF-IDF vectorization, and word embeddings to convert the text data into numerical features that machine learning models can understand.

Model Development: We have selected an appropriate machine learning algorithm for sentiment analysis, which will be trained on the preprocessed data.

Model Evaluation: To ensure the model's effectiveness, we will evaluate its performance using relevant metrics and visualize the results.

Conclusion and Recommendations: Finally, we will conclude our analysis, present the key findings, and provide recommendations for improving our product offerings based on customer sentiment insights.

In this project, we aim to demonstrate the power of NLP and sentiment analysis in helping companies make data-driven decisions that can lead to improved products and enhanced customer satisfaction. By understanding customer sentiments, we can unlock the potential for innovation and growth, making our products more competitive in the market.

**DESIGN THINKING PROCESS**

1. Empathize:

In the first stage of the design thinking process, it's essential to empathize with the target audience, which, in this case, includes the customers providing feedback, as well as the company seeking to understand customer sentiments.

Conduct user interviews or surveys to understand customer pain points, expectations, and the context in which feedback is provided.

Gather feedback from different channels, such as social media, customer reviews, and direct interactions.

2. Define:

In this stage, define the problem and frame it in a way that guides the project's direction.

Clearly articulate the problem statement: "How can we gain insights into competitor products by analyzing customer sentiment in an efficient and meaningful way?"

Establish goals and success criteria: Define what success looks like for the project, such as the accuracy of sentiment classification or the identification of actionable insights.

Create user personas: Develop detailed personas representing typical customers and their feedback patterns.

3. Ideate:

This stage is about generating creative solutions to the problem. Brainstorm various ideas on how to approach sentiment analysis and gain insights.

Organize brainstorming sessions with a diverse team to generate ideas for analyzing customer feedback.

Encourage free thinking without constraints. Consider techniques like mind mapping, brainstorming, and role-playing.

Document all ideas, even unconventional ones.

4. Prototype:

Create tangible representations of potential solutions. In this context, this stage involves creating initial models or methods for sentiment analysis.

Develop prototype sentiment analysis models using a subset of the dataset.

Experiment with different feature extraction techniques, algorithms, and pre-processing methods.

Visualize and interpret preliminary results to see which methods show promise.

5. Test:

Testing involves gathering feedback on the prototypes to understand their effectiveness and iterate on the solution.

Collect feedback from potential end-users (e.g., internal stakeholders or subject matter experts).

Conduct usability tests to ensure the sentiment analysis models align with the goals defined earlier.

Iterate on the prototypes based on feedback and test results, refining the models and analysis process.

6. Implement:

Once a viable solution has been identified through testing, it's time to implement it at scale.

Develop a production-ready sentiment analysis pipeline.

Deploy the solution to analyze a larger dataset, which may include real-time feedback data.

Establish monitoring and reporting mechanisms to track the performance and insights generated.

7. Learn and Improve:

Continuously learn from the implementation and make improvements based on ongoing feedback and new customer feedback data.

Monitor the performance of the sentiment analysis system over time.

Gather feedback from users and stakeholders to make improvements.

Update the system based on changing customer behavior and feedback trends.

**Phases of development**

1. Data Collection:

In this initial phase, you'll gather the necessary data for sentiment analysis. For your project, you are using the "Twitter Airline Sentiment" dataset from Kaggle, which contains tweets related to airline experiences. Ensure that you download and organize this dataset for further analysis.

2. Data Preprocessing:

Once the data is collected, the next step is to clean and preprocess it. This phase includes tasks such as:

Handling missing data: Address any null values or missing data points.

Removing duplicates: Eliminate duplicate entries to ensure data integrity.

Text cleaning: This involves tasks like removing special characters, converting text to lowercase, and eliminating noise from the text data.

Tokenization and stop word removal: Break text into words (tokens) and remove common, uninformative words (stop words).

Handling imbalanced classes: If the dataset has imbalanced class distributions (e.g., more positive reviews than negative ones), you may need to address this issue to avoid bias in the analysis.

3. Feature Extraction:

Feature extraction is a critical step in NLP. You can employ various techniques to convert text data into numerical features that can be used for machine learning:

Bag of Words (BoW): Create a vocabulary of words and count how often each word appears in a document.

TF-IDF (Term Frequency-Inverse Document Frequency) vectorization: Weigh words based on their importance in a document relative to the entire corpus.

Word embeddings: Use pre-trained word embeddings like Word2Vec or GloVe to represent words as dense vectors.

Feature engineering: Create custom features, such as sentiment scores or other domain-specific indicators.

4. Model Development:

Choose an appropriate machine learning algorithm or model for sentiment analysis. Common choices include:

Naive Bayes

Support Vector Machine (SVM)

Recurrent Neural Networks (RNNs) or Long Short-Term Memory (LSTM) networks

Pre-trained models like BERT

Train the selected model on the preprocessed and feature-engineered data.

5. Model Evaluation:

Once the model is trained, assess its performance using relevant evaluation metrics. Common evaluation metrics for sentiment analysis include:

Accuracy: Overall model accuracy in classifying sentiments.

Precision: Ability to correctly identify positive/negative sentiments.

Recall: Ability to capture all positive/negative sentiments.

F1-score: The harmonic mean of precision and recall.

ROC AUC (Receiver Operating Characteristic Area Under the Curve): For binary sentiment classification.

Visualize the model's performance, and use techniques like confusion matrices or ROC curves to assess its effectiveness.

6. Conclusion and Recommendations:

In the final phase, summarize the key findings from the sentiment analysis.

Provide actionable recommendations based on the insights gained from customer feedback. These recommendations can help companies identify strengths and weaknesses in competing products and improve their own offerings.

Reflect on the project's success and any limitations encountered during the development process.

**Dataset Description:**

Title: Twitter Airline Sentiment

Source: Kaggle

Link: Twitter Airline Sentiment Dataset

Overview:

The Twitter Airline Sentiment dataset is a valuable collection of tweets related to customer feedback on various airlines. It was originally created for a sentiment analysis competition on Kaggle. This dataset contains a wealth of information in the form of tweets, including both the text content and associated sentiment labels.

Key Characteristics:

Size: The dataset comprises a substantial number of tweets, making it suitable for training and evaluating machine learning models for sentiment analysis.

Attributes:

tweet\_id: A unique identifier for each tweet.

airline\_sentiment: The sentiment label associated with each tweet, categorized as either positive, negative, or neutral.

airline\_sentiment\_confidence: A confidence score indicating the degree of confidence in the sentiment label assigned to the tweet.

negativereason: If the sentiment is negative, this field provides additional information about the reason for the negative sentiment.

negativereason\_confidence: A confidence score for the negative reason assigned.

airline: The name of the airline mentioned in the tweet.

airline\_sentiment\_gold: A gold standard sentiment label.

name: The Twitter handle of the user who posted the tweet.

text: The content of the tweet.

tweet\_created: The timestamp of when the tweet was created.

retweet\_count: The number of times the tweet has been retweeted.

tweet\_coord: The coordinates of the tweet's location, if available.

tweet\_location: The location provided in the user's Twitter profile.

user\_timezone: The user's time zone.

Usefulness:

This dataset is valuable for sentiment analysis and provides insights into how customers express their feelings and opinions about different airlines. It can be used to train machine learning models to automatically classify tweets into positive, negative, or neutral sentiments, and to identify the reasons behind negative sentiments.

Challenges:

The dataset may contain noise, including misspelled words, slang, and other unstructured text data.

Class imbalance is a common challenge in sentiment analysis datasets, as there may be fewer negative or neutral tweets compared to positive ones.

Handling the different airline names and their variations can be challenging, as users may mention airlines using abbreviations or nicknames.

Relevance to the Problem:

The Twitter Airline Sentiment dataset is highly relevant to the problem of sentiment analysis on customer feedback to gain insights into competitor products. It provides a real-world collection of tweets reflecting customer sentiments about airlines, which can serve as a foundation for training and evaluating sentiment analysis models to understand and analyze customer sentiments effectively.

data preprocessing steps

1. Data Cleaning:

Handling Missing Data: Check for any missing values in the dataset and decide on an appropriate strategy for handling them. You can remove rows with missing data or impute missing values if necessary.

Removing Duplicates: Check for and remove duplicate entries in the dataset to avoid bias in the analysis.

2. Text Cleaning:

Lowercasing: Convert all text to lowercase to ensure consistent word representation. This helps in treating "Good" and "good" as the same word.

Removing Special Characters: Eliminate special characters, punctuation, and symbols, which may not carry meaningful sentiment information.

Tokenization: Split the text into individual words or tokens. This is a fundamental step for creating the bag of words representation.

Removing Stop Words: Exclude common and irrelevant words (stop words) such as "the," "is," "and," which don't contribute much to sentiment analysis.

3. Text Normalization:

Lemmatization or Stemming: Reduce words to their base or root form. For example, "running," "ran," and "runs" may be transformed to "run." You can choose either lemmatization or stemming based on your requirements.

4. Handling Contractions and Slang:

Expand contractions: Convert contractions like "don't" to "do not" for better consistency in text analysis.

Address Slang: If the dataset contains slang words or domain-specific jargon, consider mapping them to their standard equivalents.

5. Removing HTML Tags and URLs:

If your dataset includes text from web sources, remove any HTML tags and URLs that might not be relevant for sentiment analysis.

6. Handling Emoticons and Emojis:

Decide whether to convert emoticons and emojis into textual representations (e.g., ":)" to "smile") or remove them, based on their relevance to the analysis.

7. Handling Non-Standard Characters:

Check for and address any non-standard characters, special symbols, or encoding issues in the text data.

8. Text Length and Noise Reduction:

Consider removing very short or very long texts that might not provide meaningful sentiment information.

Address noise in the text, such as excessive spacing or repetitive characters.

9. Data Labeling:

If your dataset includes labels (e.g., positive, negative, neutral), ensure that they are consistent and well-defined. You may need to encode them into numerical values (e.g., 0 for negative, 1 for neutral, 2 for positive).

10. Balancing Class Distribution:

If the dataset has imbalanced class distribution (e.g., significantly more positive samples than negative ones), consider techniques such as oversampling, undersampling, or using class-weighted models to balance the classes.

11. Save Preprocessed Data:

After all the preprocessing steps, save the cleaned and transformed data for future analysis and model training.

feature extraction techniques.

Bag of Words (BoW):

BoW is a simple and widely used technique where each document (customer feedback) is represented as a vector of word frequencies.

It involves creating a vocabulary of unique words in the entire corpus of text data and counting the frequency of each word in each document.

The resulting feature matrix is typically very high-dimensional.

Term Frequency-Inverse Document Frequency (TF-IDF):

TF-IDF is another popular technique that takes into account the importance of words in a document relative to the entire corpus.

It calculates a weight for each word in a document, where words that are common in a specific document but rare in the overall corpus are given higher weights.

This reduces the dimensionality compared to BoW while preserving important information.

Word Embeddings:

Word embeddings, such as Word2Vec, GloVe, and FastText, are pre-trained models that map words into dense vector representations.

These embeddings capture semantic relationships between words and are effective for capturing context and meaning in text.

Word embeddings can be used to create document-level features by aggregating word vectors within a document.

Doc2Vec:

Doc2Vec is an extension of Word2Vec that can learn vector representations of entire documents.

Each document is represented by a dense vector that encapsulates its meaning.

These document vectors can be used as features for sentiment analysis.

N-grams:

N-grams represent contiguous sequences of n words from a document.

For example, bigrams (n=2) would represent pairs of adjacent words in the text.

N-grams can capture local word order and dependencies in the text.

Sentiment Lexicons:

Sentiment lexicons are dictionaries that contain lists of words and their associated sentiment polarities (positive, negative, neutral).

You can use sentiment lexicons to create features based on the presence and count of positive and negative words in a document.

Topic Modeling:

Topic modeling techniques like Latent Dirichlet Allocation (LDA) can be used to extract topics from the text data.

The topic distribution for each document can serve as a feature for sentiment analysis.

Word Frequency Features:

You can create features based on word frequencies, such as the total word count, average word length, and the number of exclamation points or emojis in a document.

These features can provide insights into the writing style and emotional intensity of the text.

Custom Feature Engineering:

Depending on the specific goals of your analysis, you can engineer custom features. For example, you might create features based on the presence of specific keywords or phrases related to competitor products.

The various steps involved in the **Machine Learning Pipeline** are:

* Import Necessary Dependencies
* Read and Load the Dataset
* Exploratory Data Analysis
* Data Visualization of Target Variables
* Data Preprocessing
* Splitting our data into Train and Test sets.
* Transforming Dataset using TF-IDF Vectorizer
* Function for Model Evaluation
* Model Building
* Model Evaluation

Let’s get started,

Step-1: Import the Necessary Dependencies

# utilities

**import** re

**import** numpy **as** np

**import** pandas **as** pd

# plotting

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

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

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

# nltk

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

# sklearn

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

**from** sklearn.naive\_bayes **import** BernoulliNB

**from** sklearn.linear\_model **import** LogisticRegression

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

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

**from** sklearn.metrics **import** confusion\_matrix, classification\_report

Step-2: Read and Load the Dataset

# Importing the dataset

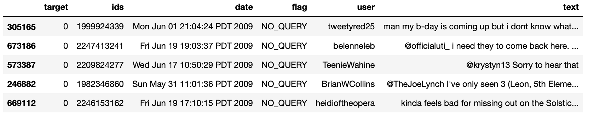
DATASET\_COLUMNS=['target','ids','date','flag','user','text']

DATASET\_ENCODING = "ISO-8859-1"

df = pd.read\_csv('Project\_Data.csv', encoding=DATASET\_ENCODING, names=DATASET\_COLUMNS)

df.sample(5)

**Output:**

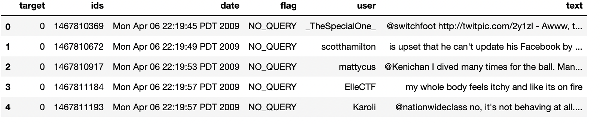


Step-3: Exploratory Data Analysis

***3.1: Five top records of data***

df.head()

**Output:**



***3.2: Columns/features in data***

df.columns

**Output:**

Index(['target', 'ids', 'date', 'flag', 'user', 'text'], dtype='object')

***3.3: Length of the dataset***

print('length of data is', len(df))

**Output:**

length of **data** **is** 1048576

***3.4: Shape of data***

df. shape

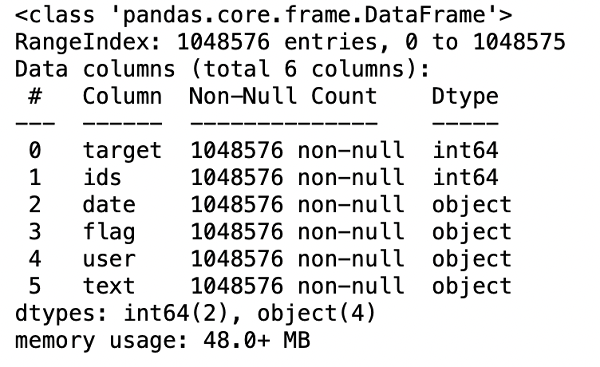
**Output:**

(1048576, 6)

***3.5: Data information***

df.info()

**Output:**



***3.6:*** **Datatypes of all columns**

df.dtypes

**Output:**

target int64

ids int64

date object

flag object

user object

**text** object

dtype: object

***3.7: Checking for null values***

**np**.sum(df.isnull().any(axis=1))

**Output:**

0

***3.8: Rows and columns in the dataset***

print('Count of columns in the data is: ', len(df.columns))

print('Count of rows in the data is: ', len(df))

**Output:**

Count of columns **in** the **data** **is**: 6

Count of rows **in** the **data** **is**: 1048576

***3.9: Check unique target values***

df['target'].unique()

**Output:**

array([0, 4], dtype=int64)

***3.10: Check the number of target values***

df['target'].nunique()

**Output:**

2

Step-4: Data Visualization of Target Variables

# Plotting the distribution for dataset.

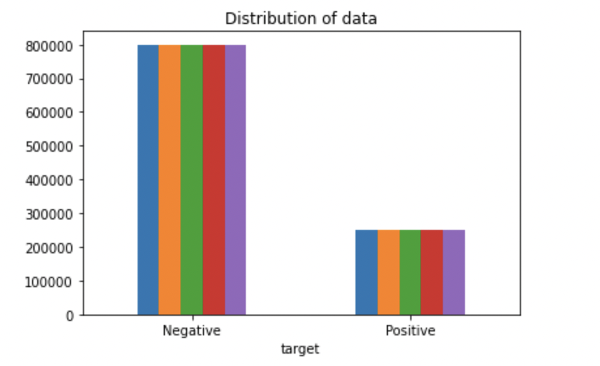
ax = df.groupby('target').count().plot(kind='bar', title='Distribution of data',legend=False)

ax.set\_xticklabels(['Negative','Positive'], rotation=0)

# Storing data in lists.

text, sentiment = list(df['text']), list(df['target'])

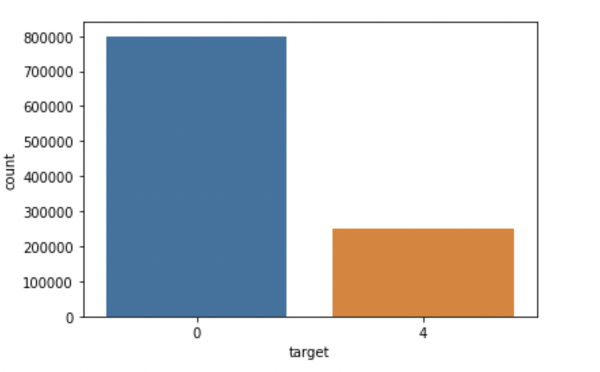
**Output:**



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

sns.countplot(x='target', **data**=df)

**Output:**



Step-5: Data Preprocessing

In the above-given problem statement, before training the model, we performed various pre-processing steps on the dataset that mainly dealt with removing stopwords, removing special characters like emojis, hashtags, etc. The text document is then converted into lowercase for better generalization.

Subsequently, the punctuations were cleaned and removed, thereby reducing the unnecessary noise from the dataset. After that, we also removed the repeating characters from the words along with removing the URLs as they do not have any significant importance.

At last, we then performed **Stemming(reducing the words to their derived stems)** and **Lemmatization(reducing the derived words to their root form, known as lemma)** for better results.

***5.1: Selecting the text and Target column for our further analysis***

data=df[['text','target']]

***5.2: Replacing the values to ease understanding. (Assigning 1 to Positive sentiment 4)***

**data**['target'] = **data**['target'].replace(4,1)

***5.3: Printing unique values of target variables***

data['target'].unique()

**Output:**

array([0, 1], dtype=int64)

***5.4: Separating positive and negative tweets***

data\_pos = data[data['target'] == 1]

data\_neg = data[data['target'] == 0]

***5.5: Taking one-fourth of the data so we can run it on our machine easily***

data\_pos = data\_pos.iloc[:int(20000)]

data\_neg = data\_neg.iloc[:int(20000)]

***5.6: Combining positive and negative tweets***

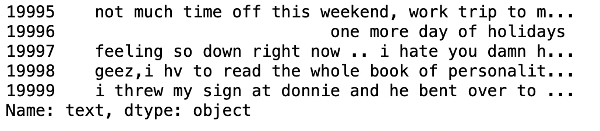
dataset = pd.concat([data\_pos, data\_neg])

***5.7: Making statement text in lowercase***

dataset['text']=dataset['text'].str.lower()

dataset['text'].tail()

**Output:**



***5.8: Defining set containing all stopwords in English.***

stopwordlist = ['a', 'about', 'above', 'after', 'again', 'ain', 'all', 'am', 'an',

'and','any','are', 'as', 'at', 'be', 'because', 'been', 'before',

'being', 'below', 'between','both', 'by', 'can', 'd', 'did', 'do',

'does', 'doing', 'down', 'during', 'each','few', 'for', 'from',

'further', 'had', 'has', 'have', 'having', 'he', 'her', 'here',

'hers', 'herself', 'him', 'himself', 'his', 'how', 'i', 'if', 'in',

'into','is', 'it', 'its', 'itself', 'just', 'll', 'm', 'ma',

'me', 'more', 'most','my', 'myself', 'now', 'o', 'of', 'on', 'once',

'only', 'or', 'other', 'our', 'ours','ourselves', 'out', 'own', 're','s', 'same', 'she', "shes", 'should', "shouldve",'so', 'some', 'such',

't', 'than', 'that', "thatll", 'the', 'their', 'theirs', 'them',

'themselves', 'then', 'there', 'these', 'they', 'this', 'those',

'through', 'to', 'too','under', 'until', 'up', 've', 'very', 'was',

'we', 'were', 'what', 'when', 'where','which','while', 'who', 'whom',

'why', 'will', 'with', 'won', 'y', 'you', "youd","youll", "youre",

"youve", 'your', 'yours', 'yourself', 'yourselves']

***5.9: Cleaning and removing the above stop words list from the tweet text***

STOPWORDS = set(stopwordlist)

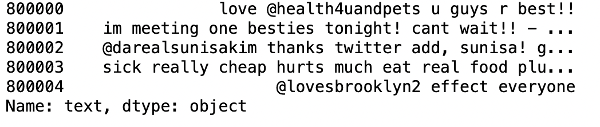
**def** **cleaning\_stopwords**(text):

**return** " ".join([word **for** word **in** str(text).split() **if** word **not** **in** STOPWORDS])

dataset['text'] = dataset['text'].apply(**lambda** text: cleaning\_stopwords(text))

dataset['text'].head()

**Output:**



***5.10: Cleaning and removing punctuations***

**import** string

english\_punctuations = string.punctuation

punctuations\_list = english\_punctuations

**def** **cleaning\_punctuations**(text):

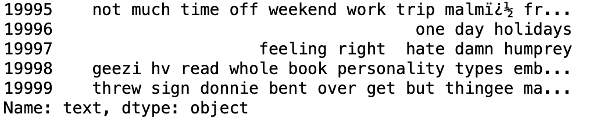
translator = str.maketrans('', '', punctuations\_list)

**return** text.translate(translator)

dataset['text']= dataset['text'].apply(**lambda** x: cleaning\_punctuations(x))

dataset['text'].tail()

**Output:**



***5.11: Cleaning and removing repeating characters***

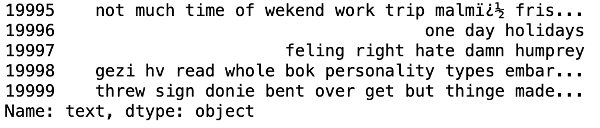
**def** **cleaning\_repeating\_char**(text):

**return** re.sub(r'(.)1+', r'1', text)

dataset['text'] = dataset['text'].apply(**lambda** x: cleaning\_repeating\_char(x))

dataset['text'].tail()

**Output:**



***5.12: Cleaning and removing URLs***

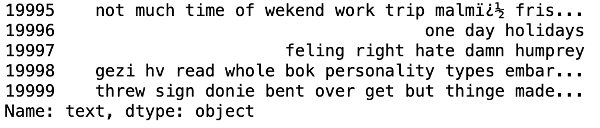
def cleaning\_URLs(data):

return re.sub('((www.[^s]+)|(https?://[^s]+))',' ',data)

dataset['text'] = dataset['text'].apply(lambda x: cleaning\_URLs(x))

dataset['text'].tail()

**Output:**



***5.13: Cleaning and removing numeric numbers***

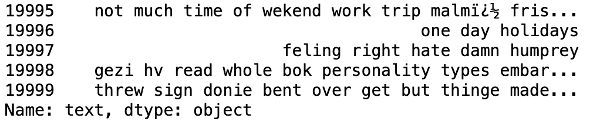
def cleaning\_numbers(data):

return re.sub('[0-9]+', '', data)

dataset['text'] = dataset['text'].apply(lambda x: cleaning\_numbers(x))

dataset['text'].tail()

**Output:**



***5.14: Getting tokenization of tweet text***

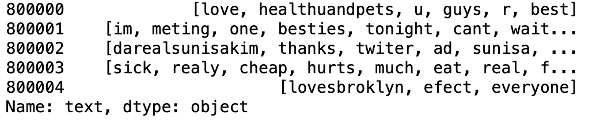
**from** nltk.tokenize **import** RegexpTokenizer

tokenizer = RegexpTokenizer(r'w+')

dataset['text'] = dataset['text'].apply(tokenizer.tokenize)

dataset['text'].head()

**Output:**



***5.15: Applying stemming***

**import** nltk

st = nltk.PorterStemmer()

def stemming\_on\_text(**data**):

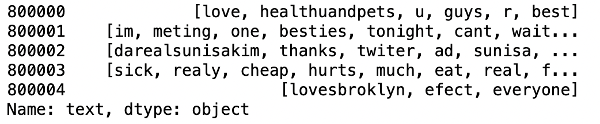
text = [st.stem(word) **for** word **in** **data**]

**return** **data**

dataset['text']= dataset['text'].apply(lambda x: stemming\_on\_text(x))

dataset['text'].head()

**Output:**



***5.16: Applying lemmatizer***

lm = nltk.WordNetLemmatizer()

def lemmatizer\_on\_text(**data**):

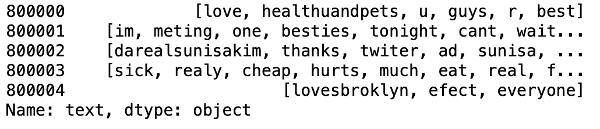
text = [lm.lemmatize(word) **for** word **in** **data**]

**return** **data**

dataset['text'] = dataset['text'].apply(lambda x: lemmatizer\_on\_text(x))

dataset['text'].head()

**Output:**



***5.17: Separating input feature and label***

X=data.text

y=data.target

***5.18: Plot a cloud of words for negative tweets***

data\_neg = data['text'][:800000]

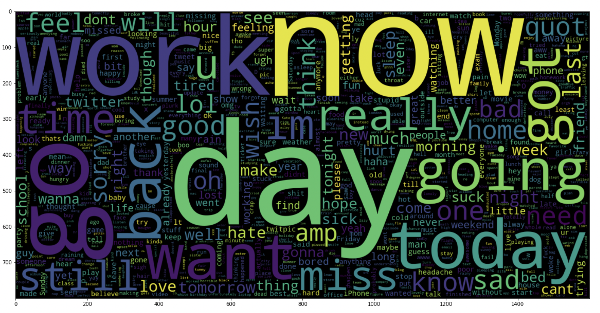
plt.figure(figsize = (20,20))

wc = WordCloud(max\_words = 1000 , width = 1600 , height = 800,

collocations=False).generate(" ".join(data\_neg))

plt.imshow(wc)

**Output:**



***5.19: Plot a cloud of words for positive tweets***

data\_pos = data['text'][800000:]

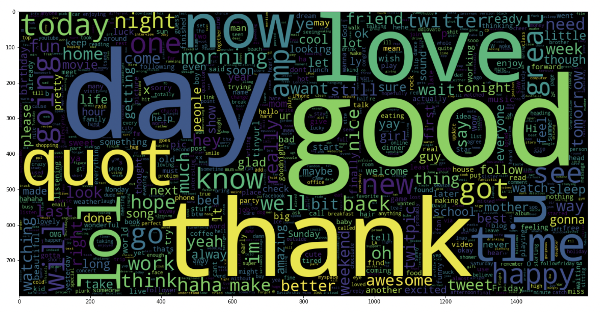
wc = WordCloud(max\_words = 1000 , width = 1600 , height = 800,

collocations=False).generate(" ".join(data\_pos))

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

plt.imshow(wc)

**Output:**



Step-6: Splitting Our Data Into Train and Test Subsets

# Separating the 95% **data** **for** training **data** and 5% **for** testing **data**

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,test\_size = 0.05, random\_state =26105111)

Step-7: Transforming the Dataset Using TF-IDF Vectorizer

***7.1: Fit the TF-IDF Vectorizer***

vectoriser = TfidfVectorizer(ngram\_range=(1,2), max\_features=500000)

vectoriser.fit(X\_train)

print('No. of feature\_words: ', len(vectoriser.get\_feature\_names()))

**Output:**

No. of feature\_words: 500000

***7.2: Transform the data using TF-IDF Vectorizer***

X\_train = vectoriser.transform(X\_train)

X\_test = vectoriser.transform(X\_test)

Step-8: Function for Model Evaluation

After training the model, we then apply the evaluation measures to check how the model is performing. Accordingly, we use the following evaluation parameters to check the performance of the models respectively:

* Accuracy Score
* Confusion Matrix with Plot
* ROC-AUC Curve

**def** **model\_Evaluate**(model):

# Predict values for Test dataset

y\_pred = model.predict(X\_test)

# Print the evaluation metrics for the dataset.

print(classification\_report(y\_test, y\_pred))

# Compute and plot the Confusion matrix

cf\_matrix = confusion\_matrix(y\_test, y\_pred)

categories = ['Negative','Positive']

group\_names = ['True Neg','False Pos', 'False Neg','True Pos']

group\_percentages = ['{0:.2%}'.format(value) **for** value **in** cf\_matrix.flatten() / np.sum(cf\_matrix)]

labels = [f'{v1}n{v2}' **for** v1, v2 **in** zip(group\_names,group\_percentages)]

labels = np.asarray(labels).reshape(2,2)

sns.heatmap(cf\_matrix, annot = labels, cmap = 'Blues',fmt = '',

xticklabels = categories, yticklabels = categories)

plt.xlabel("Predicted values", fontdict = {'size':14}, labelpad = 10)

plt.ylabel("Actual values" , fontdict = {'size':14}, labelpad = 10)

plt.title ("Confusion Matrix", fontdict = {'size':18}, pad = 20)

Step-9: Model Building

In the problem statement, we have used three different models respectively :

* Bernoulli Naive Bayes Classifier
* SVM (Support Vector Machine)
* Logistic Regression

The idea behind choosing these models is that we want to try all the classifiers on the dataset ranging from simple ones to complex models, and then try to find out the one which gives the best performance among them.

***8.1: Model-1***

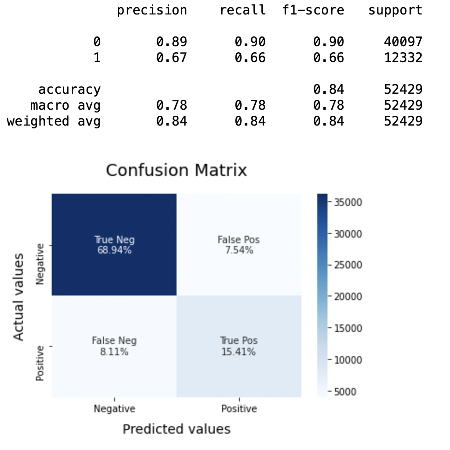
BNBmodel = BernoulliNB()

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

model\_Evaluate(BNBmodel)

y\_pred1 = BNBmodel.predict(X\_test)

**Output:**



***8.2: Plot the ROC-AUC Curve for model-1***

**from** sklearn.metrics **import** roc\_curve, auc

fpr, tpr, thresholds = **roc\_curve**(y\_test, y\_pred1)

roc\_auc = **auc**(fpr, tpr)

plt.**figure**()

plt.**plot**(fpr, tpr, color='darkorange', lw=1, label='ROC curve (area = %0.2f)' % roc\_auc)

plt.**xlim**([0.0, 1.0])

plt.**ylim**([0.0, 1.05])

plt.**xlabel**('False Positive Rate')

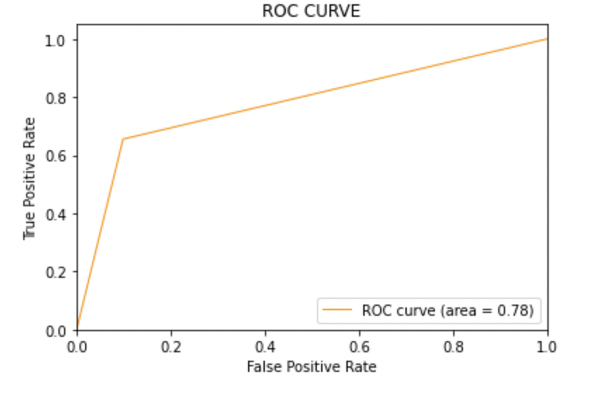
plt.**ylabel**('True Positive Rate')

plt.**title**('ROC CURVE')

plt.**legend**(loc="lower right")

plt.**show**()

**Output:**



***8.3: Model-2:***

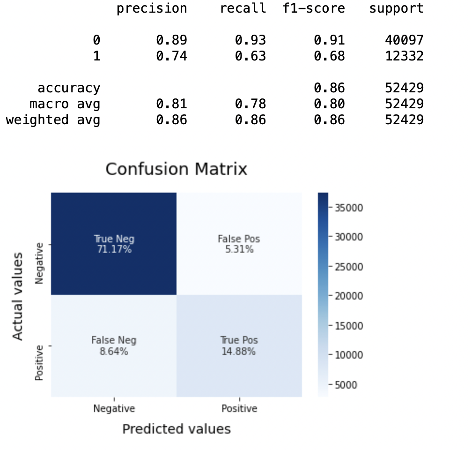
SVCmodel = LinearSVC()

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

model\_Evaluate(SVCmodel)

y\_pred2 = SVCmodel.predict(X\_test)

**Output:**



***8.4: Plot the ROC-AUC Curve for model-2***

**from** sklearn.metrics **import** roc\_curve, auc

fpr, tpr, thresholds = **roc\_curve**(y\_test, y\_pred2)

roc\_auc = **auc**(fpr, tpr)

plt.**figure**()

plt.**plot**(fpr, tpr, color='darkorange', lw=1, label='ROC curve (area = %0.2f)' % roc\_auc)

plt.**xlim**([0.0, 1.0])

plt.**ylim**([0.0, 1.05])

plt.**xlabel**('False Positive Rate')

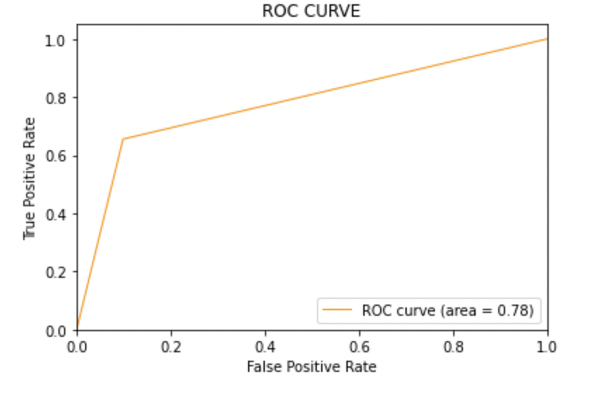
plt.**ylabel**('True Positive Rate')

plt.**title**('ROC CURVE')

plt.**legend**(loc="lower right")

plt.**show**()

**Output:**



***8.5: Model-3***

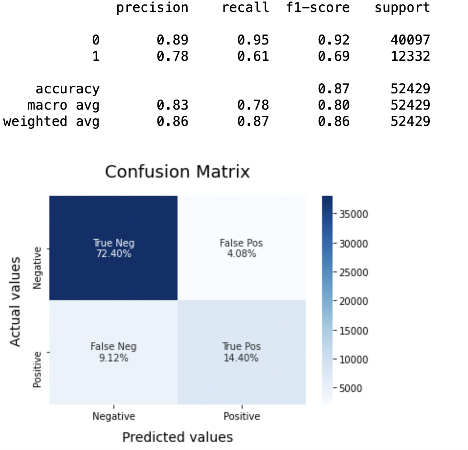
LRmodel = LogisticRegression(C = 2, max\_iter = 1000, n\_jobs=-1)

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

model\_Evaluate(LRmodel)

y\_pred3 = LRmodel.predict(X\_test)

**Output:**



***8.6: Plot the ROC-AUC Curve for model-3***

**from** sklearn.metrics **import** roc\_curve, auc

fpr, tpr, thresholds = **roc\_curve**(y\_test, y\_pred3)

roc\_auc = **auc**(fpr, tpr)

plt.**figure**()

plt.**plot**(fpr, tpr, color='darkorange', lw=1, label='ROC curve (area = %0.2f)' % roc\_auc)

plt.**xlim**([0.0, 1.0])

plt.**ylim**([0.0, 1.05])

plt.**xlabel**('False Positive Rate')

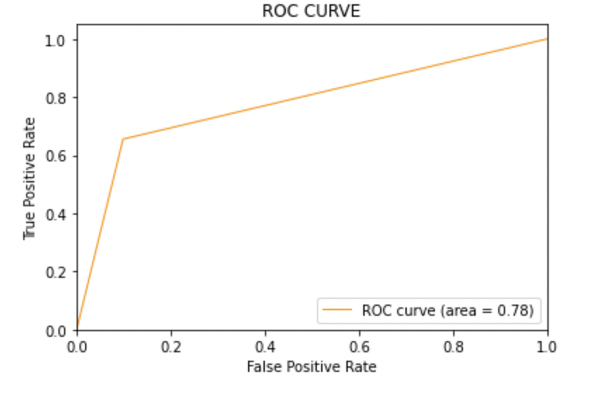
plt.**ylabel**('True Positive Rate')

plt.**title**('ROC CURVE')

plt.**legend**(loc="lower right")

plt.**show**()

**Output:**



Step-10: Model Evaluation

Upon evaluating all the models, we can conclude the following details i.e.

**Accuracy:** As far as the accuracy of the model is concerned, Logistic Regression performs better than SVM, which in turn performs better than Bernoulli Naive Bayes.

**F1-score:** The F1 Scores for class 0 and class 1 are :  
(a) For class 0: Bernoulli Naive Bayes(accuracy = 0.90) < SVM (accuracy =0.91) < Logistic Regression (accuracy = 0.92)  
(b) For class 1: Bernoulli Naive Bayes (accuracy = 0.66) < SVM (accuracy = 0.68) < Logistic Regression (accuracy = 0.69)

**AUC Score:** All three models have the same ROC-AUC score.

We, therefore, conclude that the Logistic Regression is the best model for the above-given dataset.

In our problem statement, **Logistic Regression** follows the principle of **Occam’s Razor,** which defines that for a particular problem statement, if the data has no assumption, then the simplest model works the best. Since our dataset does not have any assumptions and Logistic Regression is a simple model. Therefore, the concept holds true for the above-mentioned dataset.