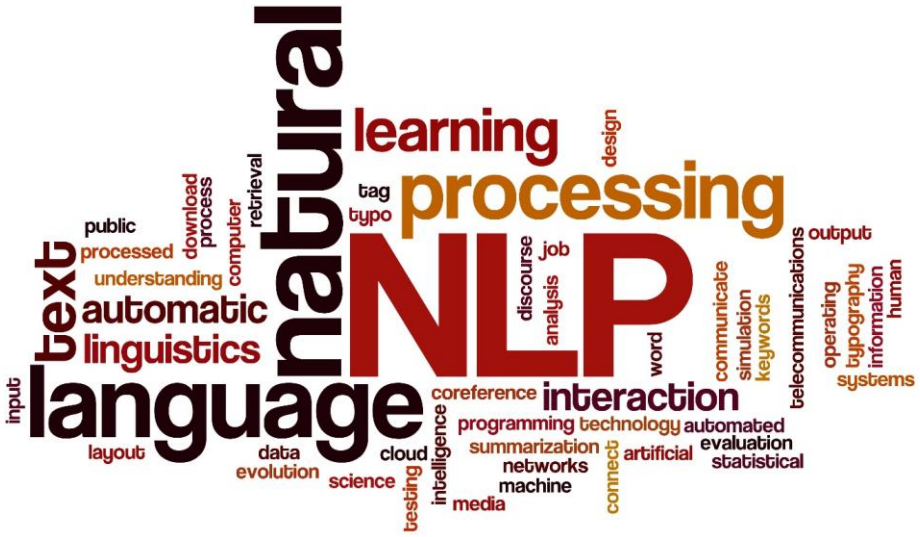
Natural Language Processing (NLP) Application Capstone Project

Automated Ticket Assignment



Program: Post Graduate Program (PGP) in Artificial Intelligence and Machine Learning(AIML)

Batch: April 2019 – April 2020

Group: 3

Deliverable - Milestone - 1

Team members:

- Gaurav Walia
- Karishma Dcosta
- Lavanya Harry Pandian
- Pallavi Kumari
- Swati Tyagi

Project Guide: Sanjay Tiwary

Program Manager: Anurag Shah

Date of Submission: 29-March-2020

Project Goal

One of the key activities of any IT function is to ensure there is no impact to the Business operations through Incident Management process. An incident is an unplanned interruption to an IT service or reduction in the quality of an IT service that affects the Users and the Business. The main goal of the Incident Management process is to provide a quick fix / workarounds or solutions that resolves the interruption and restores the service to its full capacity to ensure no business impact.
These incidents are recorded as tickets that are created by various stakeholders (Business Users, IT Users and Monitoring Tools) within IT Service Management Tool and are assigned to Service Desk teams (L1 / L2 teams).

The *goal* of this project is to build a classifier that can classify the tickets by analyzing the text using Natural Language Processing(NLP) techniques in AIML.

greatlearning
Learning for Life

Contents

Milestone 1 Detailed Report:

1. Summary of problem statement, data and findings

In this section, we describe the problem statement: explaining the current situation, opportunities for improvement and data findings: data requirement, size and source of data with its challenges and techniques to overcome the same.

Problem Statement

Current Situation –

Given the data that is collected from the IT Service Management Tool, the issues are recorded as tickets and are assigned to respective groups based on the type of issues that need to be addressed. Assigning the incidents to the appropriate group has critical importance to provide improved user satisfaction while ensuring better allocation of support resources, thus maintaining the organization's efficiency in the service.
However, the assignment of incidents to appropriate IT groups is still a manual process in many of the IT organizations.

Manual assignment of incidents is time consuming and requires human efforts. There may be mistakes due to human errors and resource consumption is carried out ineffectively because of the misaddressing. On the other hand, manual assignment increases the response and resolution times which result in user satisfaction deterioration / poor customer service.

Opportunity for improvement –

This manual process can be improvised using machine learning based systems such as automatic ticket classification mechanisms that would:

- Reduce/remove the count for human error
- Use the allocated resources efficiently
- Ensure efficient ticket classification
- Provide quick solutions and turn-around times for the organization as a whole.

By leveraging the AI technology, we shall *build a classifier that can classify the tickets into respective Groups by analyzing the text using NLP techniques in AIML.*

Data Findings

In Natural Language Processing (NLP), most of the data in the form of documents and text contain many words that are redundant for text classification, such as stop-words, misspellings, slangs; and also contain various languages since the users could potentially be located globally.

Data Requirement –

In order to understand the tickets, we require the past ticket information comprising of the ticket summary/ title which captures the essence of the issue, the detailed description for additional details, the user information and the group assigned to the respective tickets. Additional information such as separate fields for timestamp, geographic location of user, etc.,would be useful in understanding the traffic and geo of the tickets logged to assign resources as per the demand of the tickets.

The Dataset used for the project can be referred from the following location in an excel format(*.xlsx):
https://drive.google.com/drive/u/0/folders/1xOCdNI2R5hiodskIJbj-QySMQs6ccehL

Source of data and challenges –

The data that has been captured by the IT Management System Tools is unclean. The data requires to be devoid from noise, punctuations, misspellings, htmls, etc. as a start and further pre-process the data to be able to remove words which do not contribute to meaning (stop-words) and extract meaningful words (tokens) to feed the data to modelling algorithms.

	Short_description	Description	Caller	Group
0	login issue	-verified user details.(employee# & manager na...	spxjnwir pjlcoqds	GRP_0
1	outlook	\r\n\r\nreceived from: hmjdrvpb.komuaywn@gmail...	hmjdrvpb komuaywn	GRP_0
2	cant log in to vpn	\r\n\r\nreceived from: eylqgodm.ybqkwiam@gmail...	eylqgodm ybqkwiam	GRP_0
3	unable to access hr_tool page	unable to access hr_tool page	xbkucsvz gcpydteq	GRP_0
4	skype error	skype error	owlgqjme qhcozdfx	GRP_0

Figure 1 Raw data read from the Excel file with 4 columns

The methods employed to clean our data are used from the NLTK and Regular Expression (re) library.

Size of the data-

Duplicate entries - The dataset consists of 8500 entries of tickets. On analyzing for duplicate entries across all the 4 columns, 83 duplicates were observed and removed thus leading to unique 8417 values in the dataset.

```
# Drop duplicate rows
df_v1 = df
df_v1 = df_v1.drop_duplicates(keep='first', inplace=False)
df_v1.shape
```

```
(8417, 4)
```

Figure 2 dropping duplicate entries in the dataframe

Class Imbalance – Datasets require proper representation of Class information, i.e equal representation of all Groups. This would enable the modelling algorithms to be trained on equal amounts of data of any given class (Group). However, this is not the case, and we observe data imbalance in the dataset.

Given the Group information, the Unique Group Count = 74. However, there is a class imbalance w.r.t the representation of the groups in the data. Out of the total 8500 tickets, about 47% represents Group_0 tickets.

One way to control the group data and maintain imbalance is by setting a 'Threshold' value which filters out the minority Group data. The threshold value is set at default 50.

```
# Reset Assignment Group for group types with less data
Frequency_Threshold = 50
count = df_v1['Group'].value_counts(ascending=True)
idx = count[count.lt(Frequency_Threshold)].index
df_v1.loc[df_v1['Group'].isin(idx), 'Group'] = 'GRP_Manual'
print("Updated unique group types",df_v1['Group'].nunique())
df_v1['Group'].value_counts(ascending=True)
```

Figure 3 Setting a threshold at 50 to control the class imbalance during modelling

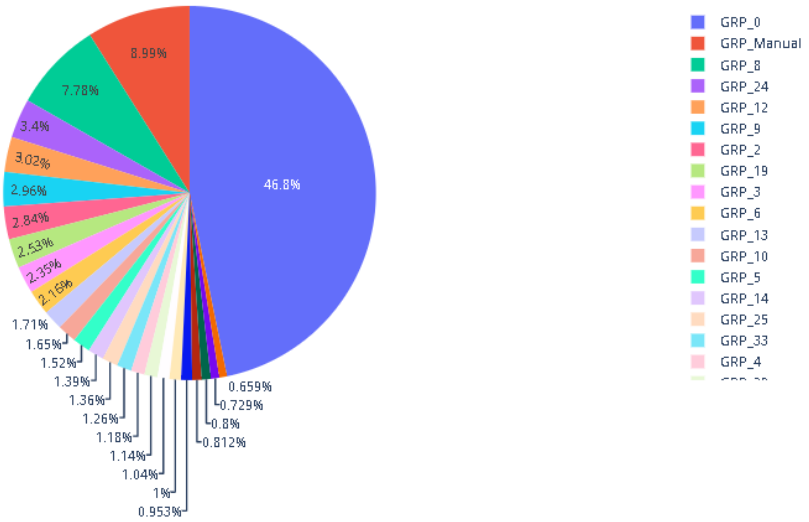


Figure 4 Out of the total 8500 tickets, 47% represents Group_0 tickets

We can tune this Threshold value based on the business requirements to filter the appropriate information into our dataset.

However, there are drawbacks to this method which urges us to focus on sampling techniques. Sampling techniques would enable us to down sample the majority classes or/and upsample the minority classes.

```
# Reset Assignment Group for group types with less data
Frequency_Threshold = 5 #50
count = df_v1['Group'].value_counts(ascending=True)
idx = count[count.lt(Frequency_Threshold)].index
df_v1.loc[df_v1['Group'].isin(idx), 'Group'] = 'GRP_Manual'
print("Updated unique group types",df_v1['Group'].nunique())
df_v1['Group'].value_counts(ascending=True)
```

Figure 5 Setting a threshold = 5 to control the class imbalance during modelling

2. Summary of the Approach to EDA and Pre-processing

Analyze and understand the structure of data

Reading the dataset – The dataset is located in the google drive as describe above and accessed using Pandas library. This file is then stored in a dataframe. While reading the file, 'Assignment group' is renamed to 'Group' and 'Short description' to 'Short_description'.

```
# Read Dataset
file_name = "Ticket_Data.xlsx"
df = pd.read_excel(file_name,encoding='cp1252')
df = df.rename(columns = {"Short description": "Short_description",
                          "Assignment group": "Group"})
```

Figure 6 Reading the dataset using Pandas library

Inorder to get a glimpse of the dataframe, we use dataframe.head()

```
df.head()
```

	Short_description	Description	Caller	Group
0	login issue	-verified user details.(employee# & manager na...	spxjnwir pjlcoqds	GRP_0
1	outlook	\r\n\r\nreceived from: hmjdrvpb.komuaywn@gmail...	hmjdrvpb komuaywn	GRP_0
2	cant log in to vpn	\r\n\r\nreceived from: eylqgodm.ybqkwiam@gmail...	eylqgodm ybqkwiam	GRP_0
3	unable to access hr_tool page	unable to access hr_tool page	xbkucsvz gcpydteq	GRP_0
4	skype error	skype error	owlgqjme qhcozdfx	GRP_0

Figure 7 Getting a preview of the dataframe

We need to analyze the shape of the data and get a basic notion of the data columns. We can use dataframe.shape and dataframe.describe() respectively to achieve the tasks.

- There are 4 columns namely Short_description, Description, Caller and Group. The total number of entries are 8500 in the dataframe.
- The count is different for each column, indicating missing values at first glimpse.
- Unique captures the unique description/ content available in each column. Eg. There are 74 unique groups in the dataframe.
- Top identifies the top word/content in the columns
- Frequency captures the frequency at which the top word/content appears in the columns.

```
# Checking Shape of the data
print("Data shape:", df.shape)
print("Data Description:")
df.describe()
```

```
Data shape: (8500, 4)
Data Description:
```

	Short_description	Description	Caller	Group
count	8492	8499	8500	8500
unique	7481	7817	2950	74
top	password reset	the	bpctwhsn kzqsbmtp	GRP_0
freq	38	56	810	3976

Figure 8 Checking shape and basic description of the data

Visualize data

Visuals are a great way to analyze data and get an idea about the data that we are handling. We resorted to using Word Clouds to analyze the following:

- Most frequent words in raw Short_description
- Most frequent words in raw Description
- Clean data in Summary field

```
def wordCloudText(df_column):
  title = ("Most Frequent words in ") + df_column.name
  stopwords = set(STOPWORDS)
  wordcloud = WordCloud(background_color='black', stopwords=stopwords, max_words=200,
                        max_font_size=40, random_state=42).generate(str(df_column.values.tolist()))
  print(wordcloud)
  fig = plt.figure(1,figsize = (20, 8))
  plt.imshow(wordcloud)
  plt.title(title ,fontsize=30)
  plt.axis('off')
  plt.show()
```

```
wordCloudText(df.Short_description)
```

```
<wordcloud.wordcloud.WordCloud object at 0x7f9b3897c6a0>
```

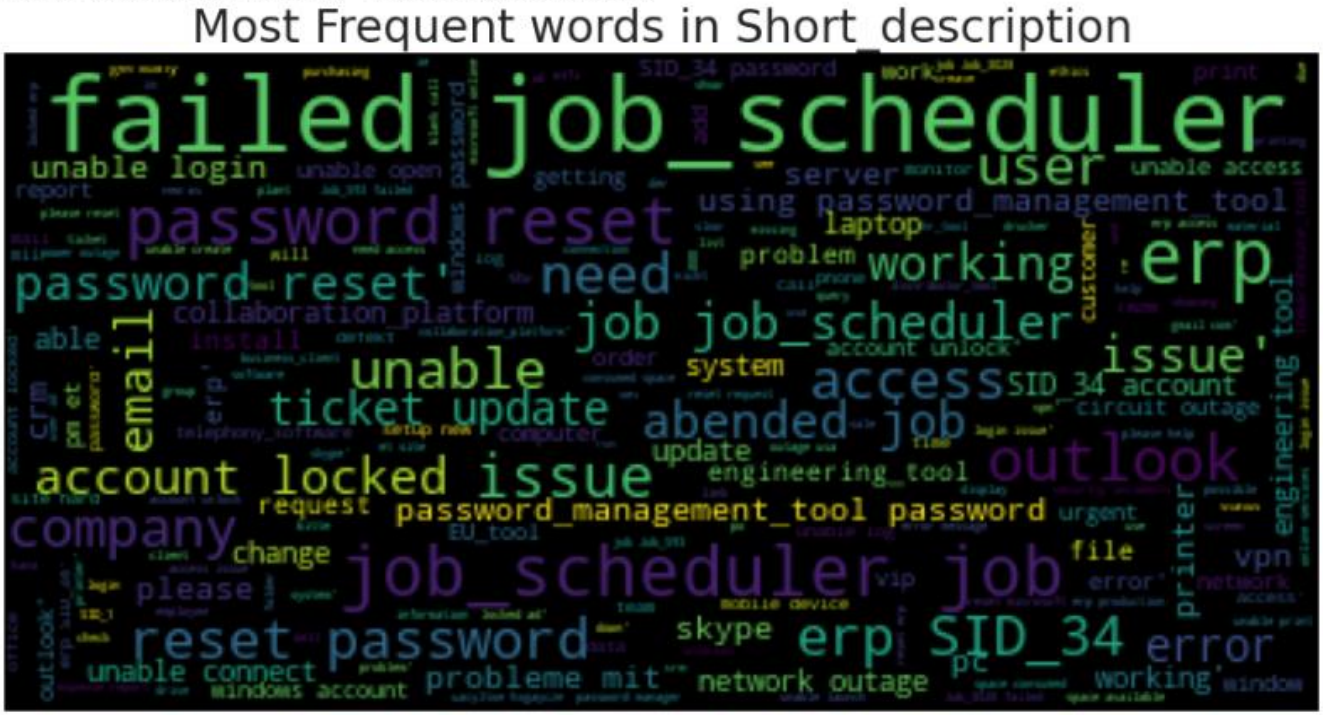


Figure 9 Most frequent words in Short_description

```
wordCloudText(df.Description)
```

```
<wordcloud.wordcloud.WordCloud object at 0x7f9b387b42e8>
```

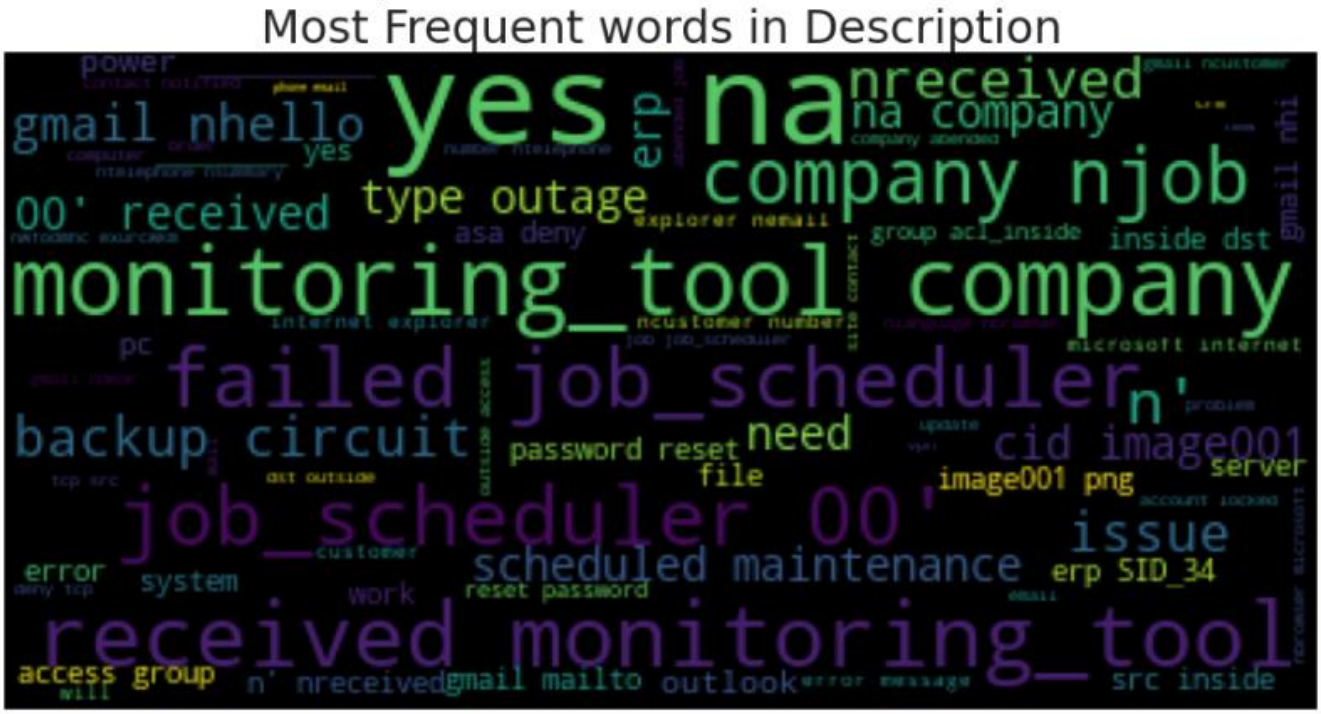


Figure 10 Most frequent words in Description

```
wordCloudText(df_v2.Summary)
```

```
<wordcloud.wordcloud.WordCloud object at 0x7f4c21cd6cc0>
```

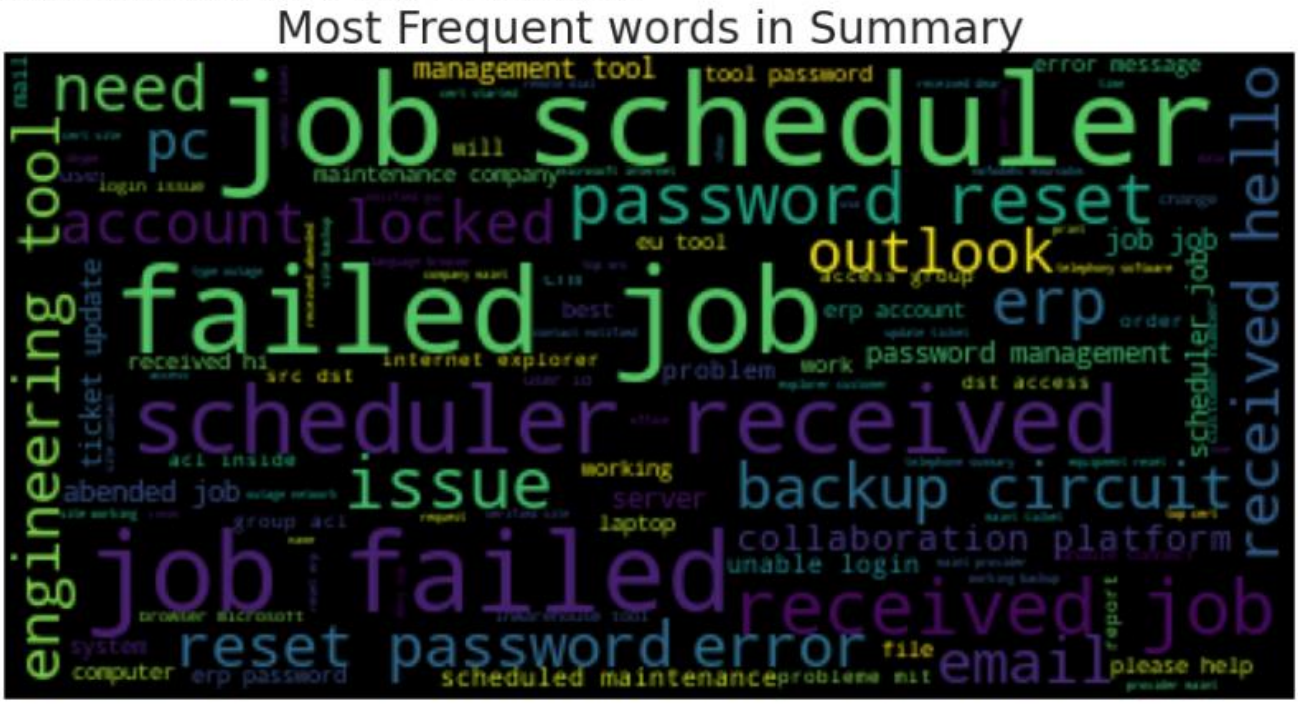


Figure 11 Clean data in Summary field

On finding the counts of each group using dataframe['col'].value_counts(), we observe that the GRP_0 has the highest presence with a frequency of 3976. The top 5 Groups are listed below.

```
#Creating dataframe of Groups on the basis of their value counts
n_grp = list(df['Group'].value_counts())
grp_name = list(df['Group'].value_counts().index)

grp = pd.DataFrame(data=grp_name,columns=['grp_name'])
grp['n_grp'] = n_grp
print(len(grp['n_grp']))
print(grp.head())
```

```
74
  grp_name  n_grp
0    GRP_0   3976
1    GRP_8    661
2   GRP_24    289
3   GRP_12    257
4    GRP_9    252
```

Figure 12 Top 5 Groups with their frequencies

This barplot represents the frequency distribution of the Groups present in the dataframe.

```
plt.figure(figsize=(15,8))
sns.barplot(x="grp_name", y="n_grp", data=grp,palette='plasma')
plt.xticks(rotation=90, ha='right')
plt.tight_layout()
```

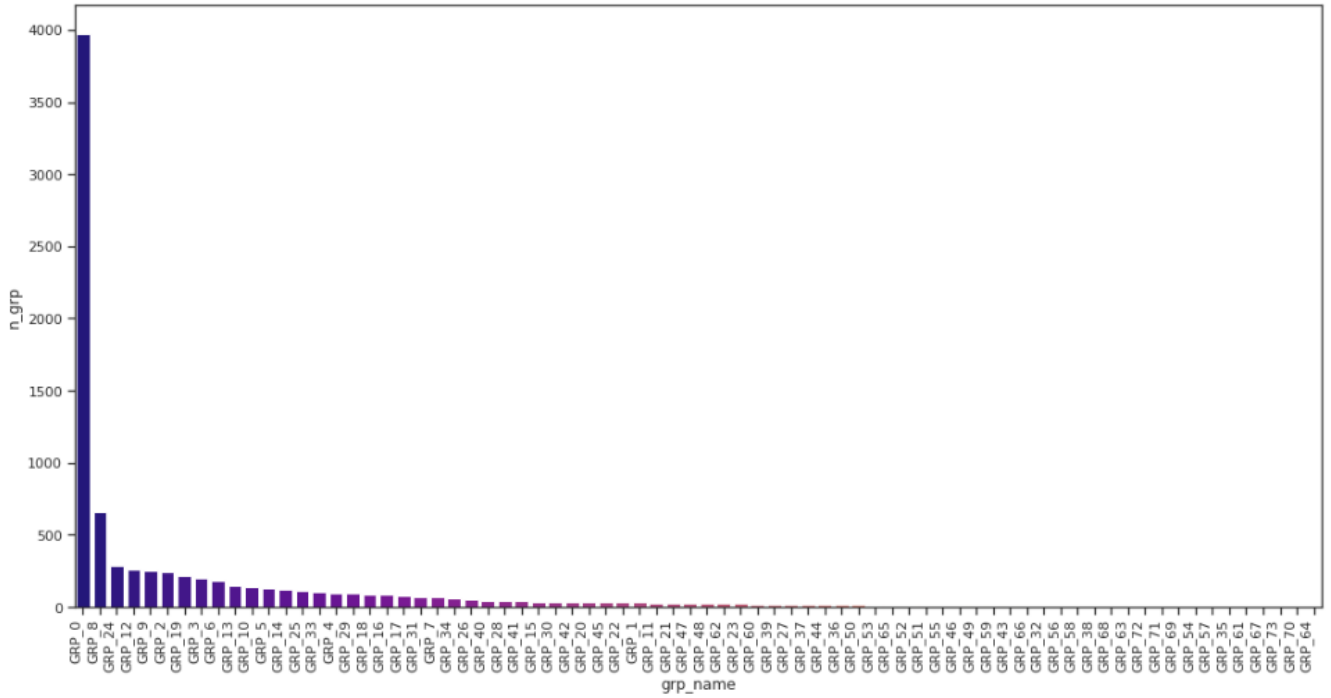


Figure 13 Barplot representing the frequency distribution of the groups

```
#Visualization of ditribution of data after merging of groups have less data than the threshold frequency
fig = px.pie(grp,values='n_grp', names='grp_name')
fig.show()
```

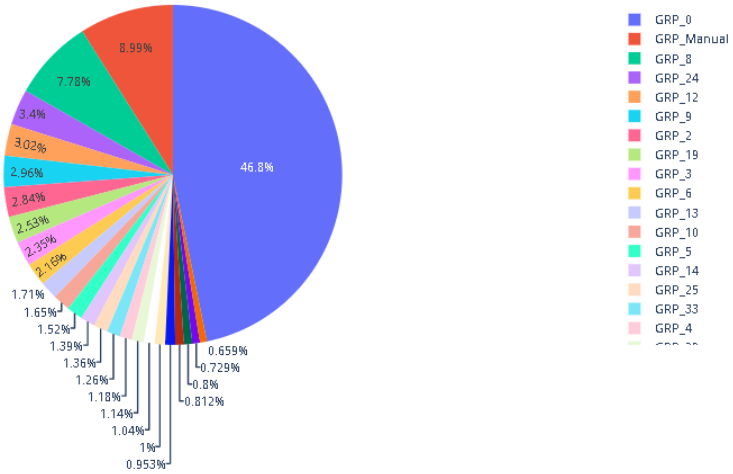


Figure 14 The pie chart showing the representation of Groups present in the data with Threshold VALUE = 50

The “Column – Caller” is anonymously provided in the dataset and hence it is difficult to comprehend. As seen in Figure 15 Caller Data anonymously provided in the dataset.

```
#Visualizing some of the Caller data
plt.figure(figsize=(20, 10))
df['Caller'].value_counts().head(30).plot(kind='barh', figsize
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9b332aeb70>
```

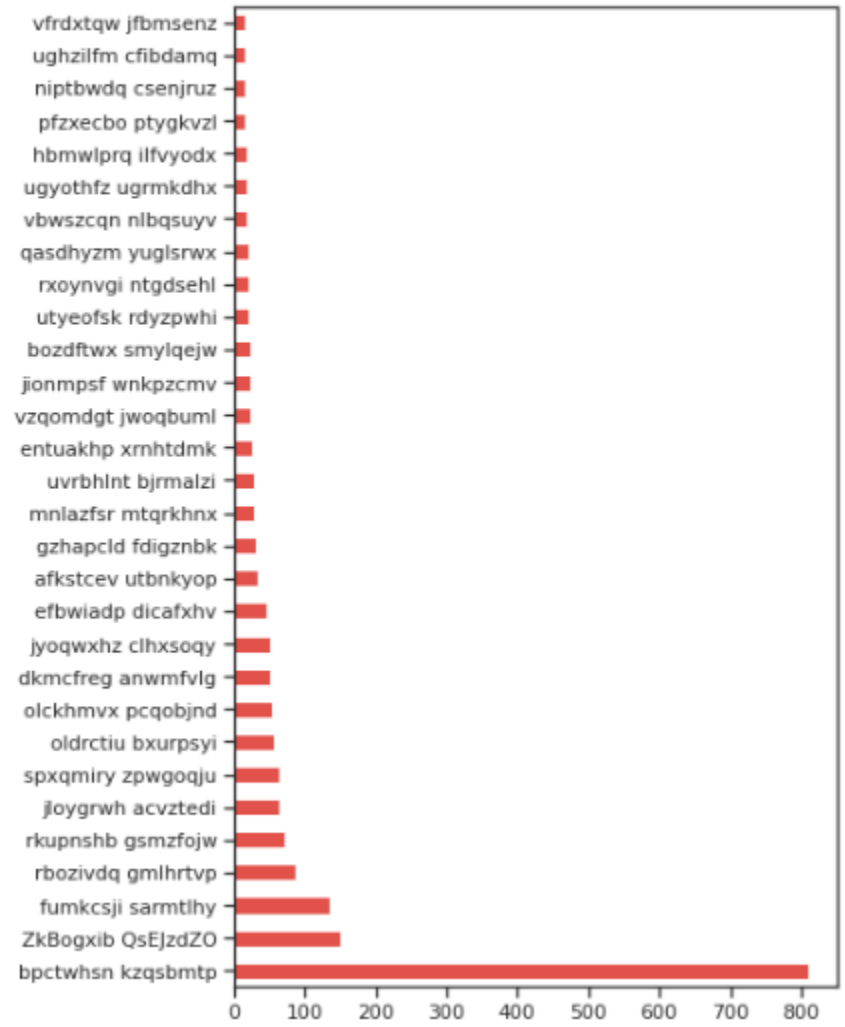


Figure 15 Caller Data anonymously provided in the dataset

The Caller data is then processed by encoding it with a uniform Prefix followed by a number. The data shows that Caller 1 has the highest frequency of 810 tickets logged. The top 10 Caller information is visible below.

```
#Since caller column contains anonymous data, assigning name Caller1, Caller2,..... for better visualization
count = 0
new_caller = []
while count != len(data):
    new_caller.append('Caller'+''+ str(count+1))
    count = count +1
data['caller'] = new_caller
data = data.head(20)
data.head(10)
```

	caller	n_caller
0	Caller1	810
1	Caller2	151
2	Caller3	134
3	Caller4	87
4	Caller5	71
5	Caller6	64
6	Caller7	63
7	Caller8	57
8	Caller9	54
9	Caller10	51

Figure 16 Caller frequency

This graph depicts the frequency distribution of the caller information.

```
#top 20 callers
fig = px.bar(data, x='caller', y='n_caller',hover_data=['n_caller'])
fig.show()
```

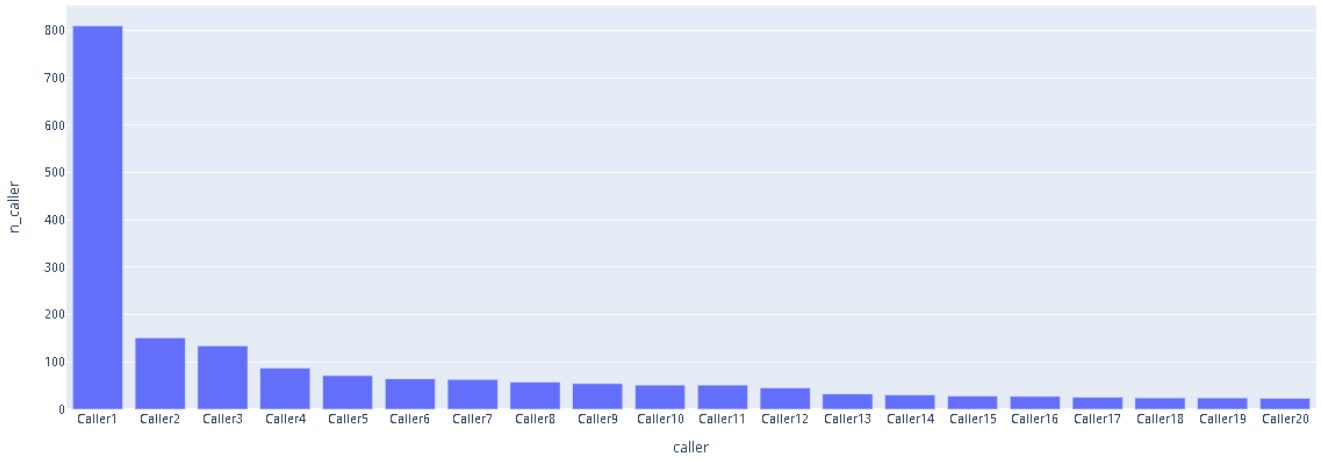


Figure 17 Barplot representing the Caller Frequency Data for Top 20 Callers

Text preprocessing

The methods employed to clean our data are used from the NLTK and Regular Expression (re) library.

```
# NLTK Stop words
import nltk
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')
nltk.download('words')
words = set(nltk.corpus.words.words())
from nltk.corpus import stopwords
stop_words = stopwords.words('english')
stop_words.extend(['received from', 'hi', 'hello','i','am','cc','sir','good morning','gentles','dear','kind','best','pl
ease',''])
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from gensim.utils import tokenize
```

```
import string
import re
```

Figure 18 - Utilizing the NLTK and re Library to clean the data

Stop-words - Text and document classification includes many words which do not contain important significance to be used in classification algorithms, such as {'a', 'about', 'about', 'after', 'again',..}.
The most common technique to deal with these words is to remove them from the texts and documents.

Noise removal - Another issue of text cleaning as a pre-processing step is noise removal. Text documents generally contains characters like punctuations or special characters and they are not necessary for text mining or classification purposes. Although punctuation is critical to understand the meaning of the sentence, but it can affect the classification algorithms negatively.

Noise that is addressed as a part of this section includes – removal of brackets, newline, multi-line spaces, numeric and alpha numeric, non-ascii text, underscores, email addresses, and disclaimers.

```
import string
import re

# Function for Text Cleaning with regex. Pass the column
def text_preprocessing(df_column):
  data = df_column.values.tolist() # Convert to list
  temp = []
  for sentence in data:
      sentence = sentence.replace("select the following link to view the disclaimer in an alternate language", '')  # r
emove disclaimer text
      sentence = re.sub(r"\[(.*?)\]"," ", sentence)  # remove text in []
      sentence = re.sub(r"\((.*?)\)"," ", sentence)  # remove text in ()
      sentence = re.sub(r"[[h][t][t][p][\S]+|[w][w][w][\S]+|[\S]+[@][\S]+"," ", sentence)  # remove email addresses, we
b address and urls
      sentence = re.sub(r"[\S]+[\d]+[\S]+"," ", sentence) # remove alphanumerics and numerics (dates, time, request id
etc.)
      sentence = re.sub(r"\W(?<!['. ])"," ", sentence)  # remove all non words with negative look back except ('. space
s)
      sentence = re.sub(r"[^a-zA-z.| ]+"," ", sentence) # remove non-alphabetic text
      sentence = re.sub(r"[\_]+"," ", sentence) # remove underscores
      sentence = re.sub(r"[\s]+"," ", sentence) # replace multiple spaces with single space
      sentence = sentence.strip('\n')
      sentence = sentence.lower()
      temp.append(sentence)
  return(temp)
```

Figure 19 Noise Removal from text

Capitalization - Sentences can contain a mixture of uppercase and lower case letters. Multiple sentences make up a text document. To reduce the problem space, the most common approach is to reduce everything to lower case. This brings all words in a document in same space, but it often changes the meaning of some words, such as "US" to "us" where first one represents the United States of America and second one is a pronoun. To solve this, slang and abbreviation converters can be applied.

sentence: Cant log into vpn
capitalized text: cant log into vpn

Lemmatization - Text lemmatization is the process of eliminating redundant prefix or suffix of a word and extract the base word (lemma).
word: see or saw
lemma text: see or saw depending on whether the use of the token was as a verb or a noun

```
# lemmetise words
wordnet_lemmatizer = WordNetLemmatizer()
temp = []
for eachrow in data:
    lemma_words = []
    for eachword in eachrow:
        eachword = wordnet_lemmatizer.lemmatize(eachword, pos = "n")
        eachword = wordnet_lemmatizer.lemmatize(eachword, pos = "v")
        eachword = wordnet_lemmatizer.lemmatize(eachword, pos = ("a"))
        lemma_words.append(eachword)
    temp.append(lemma_words)
```

Figure 20 Lemmatizing words

Concatenation of Short_description and Description into a new field – 'Summary'

```
df_v1["Summary"] = df_v1['Short_description'].str.cat(df_v1['Description'], sep = ". ")
df_v2 = df_v1.copy()
df_v2 = df_v1.drop(['Short_description','Description'],axis=1)
df_v2.head(20)
```

Figure 21 Concatenation of Short_description and Description into a new field – 'Summary'

	Caller	Group	Summary
0	spxjnwir pjlcoqds	GRP_0	login issue. verified user details. checked t...
1	hmjdrvpb komuaywn	GRP_0	outlook. received from hello team my meetings...
2	eylqgodm ybqkwiam	GRP_0	cant log in to vpn. received from hi i cannot...
3	xbkucsvz gcpydteq	GRP_0	unable to access hr tool page. unable to acces...
4	owlgqjme qhcozdfx	GRP_0	skype error . skype error
5	eflahbxn ltdgrvkz	GRP_0	unable to log in to engineering tool and skype...
6	jyoqwxhz clhxsoqy	GRP_1	event the value of mountpoint threshold for . ...
7	eqzibjhw ymebpoih	GRP_0	employment status new non employee . employm...
8	mdbegvct dbvichlg	GRP_0	unable to disable add ins on outlook. unable t...
9	fumkcsji sarmtlhy	GRP_0	ticket update on . ticket update on
10	badgknqs xwelumfz	GRP_0	engineering tool says not connected and unable...
11	dcqsolkx kmsijcuz	GRP_0	hr tool site not loading page correctly. hr to...
12	oblekmrw qltgvspb	GRP_0	unable to login to hr tool to sgxqsuojr xwbeso...
13	iftldbmu fujslwby	GRP_0	user wants to reset the password. user wants t...
14	epwyvjsz najukwho	GRP_0	unable to open payslips . unable to open paysl...
15	fumkcsji sarmtlhy	GRP_0	ticket update on . ticket update on
16	chobktqj qdamxfuc	GRP_0	unable to login to company vpn. received from...
17	sigfdwcj reofwzlm	GRP_3	when undocking pc screen will not come back. w...
18	nqdyowsm yqerwtna	GRP_0	erp account locked. erp account locked
19	ftsqkvre bqzrupic	GRP_0	unable to sign into vpn. unable to sign into vpn

Figure 22 Viewing the dataframe with the new field – 'Summary'

Checking for Missing values - On analyzing the raw data, the fields (Short Description, Description, Caller and Group) were verified for missing values. It was observed that there are 8

missing values in Column - Short Description and 1 missing value in Column – Description and none in Column – Group.

The missing values were addressed by imputing values like space, filler words that would be eradicated in the stop-word removal or text cleaning.

```
# Check for number of null values in each columns
print("Total Null Values in data:", df_v1.isnull().sum().sum())
print("\nNull Values accross columns:\n", df_v1.isnull().sum())
print("\nData with 'Null' Short Description")
df_v1.loc[df_v1['Short_description'].isnull()==True]
```

```
Total Null Values in data: 9

Null Values accross columns:
 Short_description    8
Description           1
Group                 0
```

```
# Check for number of null values in each columns
print("Total Null Values in data:", df_v1.isnull().sum().sum())
print("\nNull Values accross columns:\n", df_v1.isnull().sum())
print("\nData with 'Null' Short Description")
df_v1.loc[df_v1['Short_description'].isnull()==True]
```

```
Total Null Values in data: 9

Null Values accross columns:
 Short_description    8
Description           1
Group                 0
```

Figure 23 Checking for missing values and imputing data in the dataset columns

Language translations – It has been observed that languages besides English are present in the dataset. As a part of the text processing activity, English alone has been considered and any other non-english text is dropped. However, the translation will be addressed as a part of Milestone -2.

Create word vocabulary and tokens -

Tokenization - Tokenization is the process of breaking down a stream of text into words, phrases, symbols, or any other meaningful elements called tokens. The main goal of this step is to extract individual words in a sentence. Along with text classification, in text mining, it is necessary to incorporate a parser in the pipeline which performs the tokenization of the documents; for example:
sentence: cant log into vpn
tokens: {'cant', 'log', 'into', 'vpn'}

```
# Remove stopwords
df_v2['Summary'] = df_v2['Summary'].apply(lambda x: ' '.join([word for word in x.split() if word not in (stop_words)]))

# Remove words not in Englsih Dictionary (typos, anonymised names)
df_v2['Summary'] = df_v2['Summary'].apply(lambda x: ' '.join([word for word in x.split() if word in (words)]))

# Tokenise 'Summary' column
data = df_v2.Summary.values.tolist()

data = [list(tokenize(sentences)) for sentences in data]
```

Figure 24 Utilizing Tokenize to create word tokens

Weighted Words - The most basic form of weighted word feature extraction is TF, where each word is mapped to a number corresponding to the number of occurrences of that word in the whole corpora. Methods that extend the results of TF generally use word frequency as a boolean or logarithmically scaled weighting.
In all weight words methods, each document is translated to a vector (with length equal to that of the document) containing the frequency of the words in that document. Although this approach is intuitive, it is limited by the fact that particular words that are commonly used in the language may dominate such representations.

Term Frequency-Inverse Document Frequency (tf-idf) - Different word embedding procedures have been proposed to translate these unigrams into consummable input for machine learning algorithms. A very simple way to perform such embedding is weighted words term-frequency~(TF) and TF-IDF where each word will be mapped to a number corresponding to the number of occurrence of that word in the whole corpora.
Although tf-idf tries to overcome the problem of common terms in document, it still suffers from some other descriptive limitations. Namely, tf-idf cannot account for the similarity between words in the document since each word is presented as an index.

```
# Create Weighted Word Vectors
tfidf_vectors = TfidfVectorizer(min_df=3,max_features= maxlen)
tfidf_db = tfidf_vectors.fit_transform(data).toarray()
tfidf_db = pd.DataFrame(tfidf_db)
```

```
tfidf_db.head()
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
0	0.214291	0.00000	0.0	0.0	0.0	0.0	0.0	0.286802	0.0	0.0	0.0	0.278404	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.000000	0.00000	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.530371	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.000000	0.00000	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.000000	0.54983	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.000000	0.00000	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 367 columns

Figure 25 Creating weighted vectors of the vocabulary

Build a Classification model – Train and Test the models

Once we create a vectorized form representing the vocabulary, we now encode the Group information labelled as GRP_X as 0,1…using label encoder(). This field represents the Target column.

```
le = preprocessing.LabelEncoder()
df_v2['Group']= le.fit_transform(df_v2['Group']) # LabelEncode 'Groups'
df_v2.head(20)
```

Figure 26 Encoding the Group Data using LabelEncoder

Training the model - We then map the X(input data) and y(target) to the vectorized data and encoded groups in order to build our models. The training and testing datasets are formulated from the X and y data, in a ratio of 60-40 training and test data respectively using the train_test_split() function. The ideal range is 60-40 to 80-20. On changing the *test_size* parameter, we can modify the testing datasize.

```
X = tfidf_db
y = df_v2['Group']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.40, random_state=42)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
```

Figure 27 Setting X and y(Target), splitting between training and testing sets

We then fit the training data to the model using model.fit(X_train, y_train). The predicted value of y is obtained using model.predict(X_test).

Evaluation Criteria-

Training and Testing scores are determined based on the training and testing sets respectively for the models that are architected. We use model.score(X,y) to evaluate the same.

Recall and precision and confusion matrix are metrics are used to determine the algorithms response to the multi-classification problem.

Recall (also known as sensitivity) is the fraction of positives events that you predicted correctly as shown below and Precision is the fraction of predicted positives events that are actually positive as shown below. Confusion matrix is the matrix to the right which depicts the Y_actual and Y_predicted.

$$\textbf{Precision} = \frac{\text{True Positive}}{\text{Actual Results}} \quad \text{or} \quad \frac{\text{True Positive}}{\text{True Positive + False Positive}}$$

$$\textbf{Recall} = \frac{\text{True Positive}}{\text{Predicted Results}} \quad \text{or} \quad \frac{\text{True Positive}}{\text{True Positive + False Negative}}$$

$$\textbf{Accuracy} = \frac{\text{True Positive + True Negative}}{\text{Total}}$$

Trade-off

F1 score is the harmonic mean of recall and precision, with a higher score as a better model. The F1 score is calculated using the following formula:

$$\textbf{F1 Score} = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

We have employed weighted-average F1-score, or weighted-F1 wherein we weight the F1-score of each class by the number of samples from that class.

3. Deciding Models and Model Building

The following models are considered as a part of classifying the tickets into their respective groups. In Milestone 1, we employed Traditional classification algorithms, such as the Naïve Bayes, Support Vector Classifier, Decision Trees and Random Forests. For the given problem which is a supervised learning, multi-classification problem, the approach was to tackle the problem using conventional techniques in Milestone 1 and thereby proceeding to Deep Neural networks, such as - a. Long Short-Term Memory (LSTM), b. Recurrent Convolutional Neural Networks (RCNN), c. Random Multimodel Deep Learning (RMDL), etc in Milestone 2. Majority of the time about 60% was spent in Data cleaning activities to prepare the data for modelling.

Below is a comparison of the models implemented along with the pros and cons of each.

Comparison of Models employed:

Model	Training Accuracy (%)	Testing Accuracy (%)	F1 score (weighted)
Naïve Bayes	67	58	0.67
SVC	78	60	0.65
Decision Trees	88	54	0.58
Random Forests	88	60	0.68

Naive Bayes

Naive Bayes is a probabilistic learning algorithm derived from Bayes Theorem. Naive Bayes Model is considered to be extremely fast, reliable, and has stable classification ability relative to other classification algorithms. The algorithm is based on the assumption that each feature in independent of each other while predicting the classification.

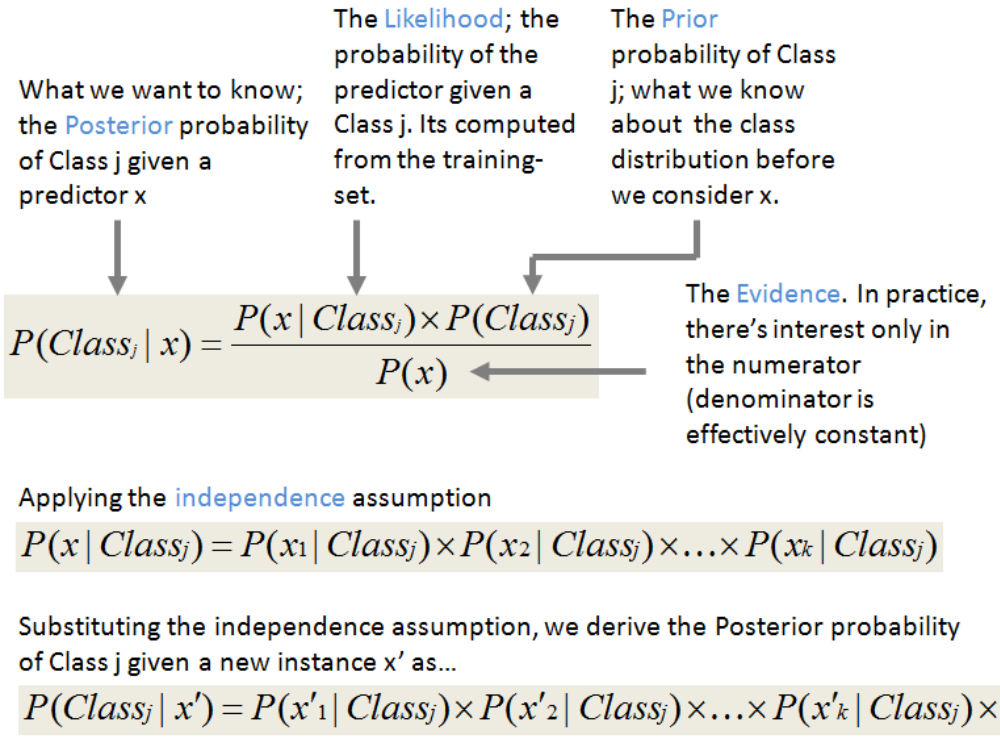


Figure 28 Explanation of Naive Bayes

Pros:

- Simple, fast and well in multi class prediction.
- Performs better with less training data as it assumes feature independence

Cons:

- Bad estimator hence the probability outputs are not taken too seriously.
- Assumptions of independent feature cannot represent real time data.
- Zero frequency - If training data set gets a category not trained on earlier, then model will assign a 0 (zero) probability and will be unable to make a prediction.

The multinomial Naive Bayes classifier is suitable for classification with discrete features (e.g., word counts for text classification), we will build Multinomial Naive Bayes model for our dataset.

```
# We will use multinomialNB for this dataset
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report, precision_score, recall_score
NBModel = MultinomialNB(alpha = 0.001)
NBModel.fit(X_train, y_train)
NB_y_pred = NBModel.predict(X_test)
print('NB Training Accuracy:', 100*NBModel.score(X_train , y_train))
print('NB Test Accuracy:', 100*NBModel.score(X_test , y_test))
```

```
NB Training Accuracy: 67.29411764705883
NB Test Accuracy: 58.735294117647065
```

```
print ('Precision Score:', precision_score(y_test, NB_y_pred, pos_label='Positive', average='weighted'))
print ('Recall Score:', recall_score(y_test, NB_y_pred, pos_label='Positive', average='weighted'))

F1score = f1_score(NB_y_pred, y_test, average='weighted')
print('F1 Score:', F1score)
print(metrics.confusion_matrix(y_test,NB_y_pred))
print(metrics.classification_report(y_test,NB_y_pred))
```

```
Precision Score: 0.5873529411764706
Recall Score: 0.5873529411764706
F1 Score: 0.6697032221054285
```

Figure 29 Implementation of Naive Bayes

Result: We can see that the Training accuracy is 67% and testing accuracy is 58% with Naive Bayes Model. The model is able to predict True Positives and False Negatives equally.

Support Vector Classifier (SVC)

Support Vector Machine (SVM) creates a hyperplane between the classes which acts as decision boundary for each class. Data falling within these boundaries will belong to that particular class.

SVM can classify non-linear data and can capture complex relationships between data points without having to perform difficult transformations. While Naïve Bayes treats the features of dataset as independent, SVM analyses the interactions between each feature to certain degree using Radial Basis Function (RBF).

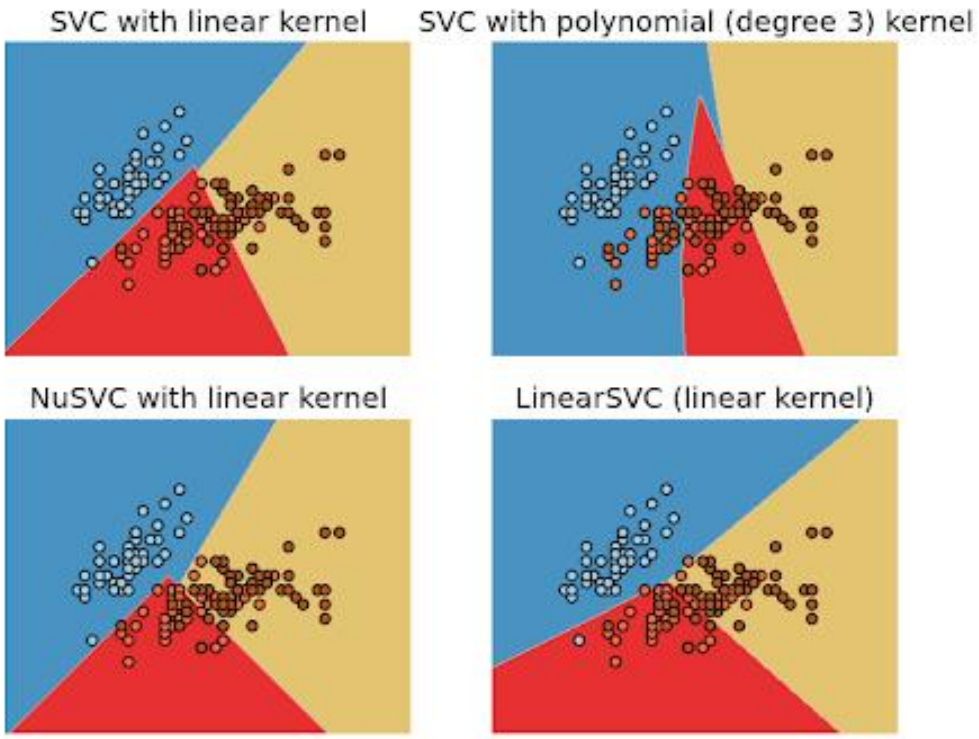


Figure 30 Visual representation of SVC

Gaussian RBF Kernel is a general-purpose kernel commonly used in SVM when there is no prior knowledge about the data. The RBF kernel on two samples x and x', represented as feature vectors in some input space, is defined as

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right)$$

$$\|\mathbf{x} - \mathbf{x}'\|^2$$

may be recognized as the squared Euclidean distance between the two feature vectors. σ is a free parameter.

Pros:

- Less affected by outliers, relatively computationally efficient and accurate than its competitors.
- Effective where number of features are greater than the number of samples.
- Good generalization capabilities which prevents it from over-fitting

Cons:

- Does not perform very well when the data of target classes are overlapping.
- Choosing an appropriate Kernel function for handling the non-linear data could be tricky and complex
- Requires lot of memory size to store all support vectors and takes long time to train on larger dataset

```
# Creating SVC Model
from sklearn import metrics
from sklearn.metrics import accuracy_score,f1_score,confusion_matrix
from sklearn.metrics import classification_report, precision_score, recall_score

X = tfidf_db
y = df_v2['Group']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.40, random_state=42)

svm_model = SVC(kernel='linear',C=10)
svm_model.fit(X_train, y_train)
y_pred = svm_model.predict(X_test)
print('Training Accuracy:', 100*svm_model.score(X_train , y_train))
print('Test Accuracy:',100*svm_model.score(X_test , y_test))
```

```
Training Accuracy: 78.17647058823529
Test Accuracy: 60.029411764705884
```

```
print ('Precision Score:', precision_score(y_test, y_pred, pos_label='Positive', average='weighted'))
print ('Recall Score:', recall_score(y_test, y_pred, pos_label='Positive', average='weighted'))

F1score = f1_score(y_pred, y_test, average='weighted')
print('F1 Score:', F1score)
print(metrics.confusion_matrix(y_test,y_pred))
print(metrics.classification_report(y_test,y_pred))
```

```
Precision Score: 0.5481669672064655
Recall Score: 0.6002941176470589
F1 Score: 0.6583876015127572
```

Figure 31 Implementation of SVC

Result: We can see that the Training accuracy is around 78% and testing accuracy is around 60% with SVC. The model is able to predict True Positives and False Negatives equally.

Decision Tree (DT)

Decision Tree solves the problem of machine learning by transforming the data into tree representation. Each internal node of the tree denotes an attribute and each leaf node denotes a class label.

Decision tree algorithm can be used to solve both regression and classification problems. Decision Tree creates a training model which can use to predict class by learning decision rules inferred from training data. Decision tree creates a model to predict the labels by learning the decision rule from training data.

The cost functions try to find most homogeneous branches, or branches having groups with similar responses. The mean of responses of the training data inputs of that group is considered as prediction for that group.

Classification: G = sum(pk * (1 — pk))

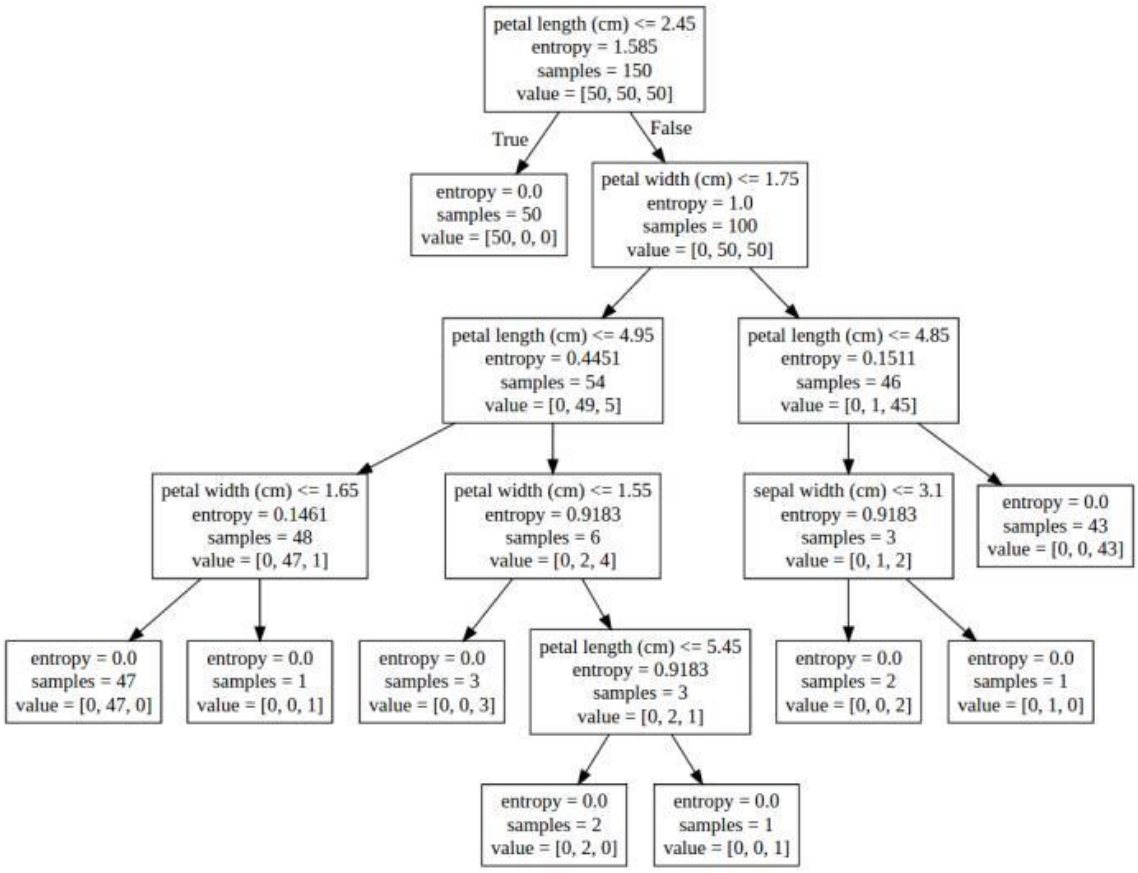


Figure 32 Explanation of Decision Trees

Pros:

- Missing values does not affect decision tree.
- Requires less effort for data preparation during pre-processing, does not need scaling or normalization.
- The Number of hyper-parameters to be tuned is almost null.
- A Decision trees model is very intuitive and Interpretation of a complex Decision Tree model can be simplified by its visualizations

Cons:

- Instable as a small change in the data can cause a large change in the structure of the decision tree.
- For a Decision tree sometimes calculation can go far more complex compared to other algorithms.
- Probability of overfitting is high and training time is more making it expensive and complex.
- low prediction accuracy compared to other algorithms and can become complex when there are many class labels.

```
# Using Decision Tree Classifier
from sklearn.tree import DecisionTreeClassifier
dt_model = DecisionTreeClassifier(criterion = 'entropy' )
dt_model.fit(X_train, y_train)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

```
y_predict = dt_model.predict(X_test)
print('Training Accuracy:',100*dt_model.score(X_train, y_train))
print('Test Accuracy:',100*dt_model.score(X_test , y_test))
```

```
Training Accuracy: 88.29411764705883
Test Accuracy: 53.970588235294116
```

```
print ('Precision Score:', precision_score(y_test, y_predict, pos_label='Positive', average='weighted'))
print ('Recall Score:', recall_score(y_test, y_predict, pos_label='Positive', average='weighted'))

F1score = f1_score(y_predict, y_test, average='weighted')
print('F1 Score:', F1score)
print(metrics.confusion_matrix(y_test,y_predict))
print(metrics.classification_report(y_test,y_predict))
```

```
Precision Score: 0.4853194149941848
Recall Score: 0.5397058823529411
F1 Score: 0.5778675867829679
```

Figure 33 Implementation of Decision Trees

Result: We can see that the training accuracy is around 88% and testing accuracy is around 54% with Decision Trees. The model is able to predict True Positives and False Negatives *almost* equally.

Random Forest (RF)

Random forest classifier creates a number of decision trees from randomly selected subset of training set. It then uses averaging to improve the predictive accuracy and control over-fitting. Random forest applies weight concept, tree with high error rate are given low weight value and vice versa. This would increase the decision impact of trees with low error rate.

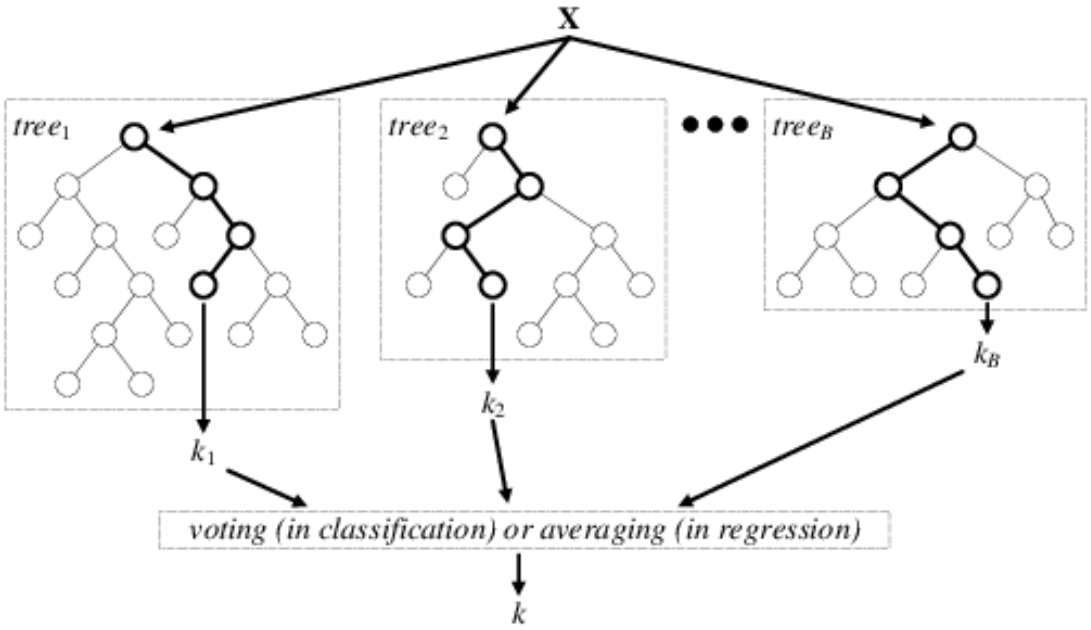


Figure 34 Explanation of Random Forests

Pros:

- Random forest is an accurate and robust method because of the number of decision trees participating in the process.
- It takes the average of all the predictions, which cancels out the biases thereby does not suffer from the overfitting problem.
- Can handle missing values and can be used in both classification and regression problems.

Cons:

- Random forest is slow in generating predictions as it has multiple decision trees. All the trees in the forest must make a prediction then perform voting on it. This whole process is time-consuming.
- The model is difficult to interpret compared to a decision tree.

```
#Using Random Forest Classifier
from sklearn.ensemble import RandomForestClassifier
rfcl = RandomForestClassifier(criterion = 'entropy', n_estimators = 50)
rfcl = rfcl.fit(X_train, y_train)
test_pred = rfcl.predict(X_test)
print('Training Accuracy:', 100*rfcl.score(X_train , y_train))
print('Test Accuracy:',100*rfcl.score(X_test , y_test))
```

```
Training Accuracy: 88.29411764705883
Test Accuracy: 59.35294117647059
```

```
print ('Precision Score:', precision_score(y_test, test_pred, pos_label='Positive', average='weighted'))
print ('Recall Score:', recall_score(y_test, test_pred, pos_label='Positive', average='weighted'))

F1score = f1_score(test_pred, y_test, average='weighted')
print('F1 Score:', F1score)
print(metrics.confusion_matrix(y_test,test_pred))
print(metrics.classification_report(y_test,test_pred))
```

```
Precision Score: 0.5451418333947825
Recall Score: 0.5935294117647059
F1 Score: 0.6762291195123756
```

Figure 35 Implementation of Random Forests

Result: We can see that the training accuracy is around 88% and testing accuracy is around 60% with Random Forests. The model is able to predict True Positives and False Negatives *almost* equally.

4. How to improve your model performance?

- Hyper-parameter tuning for the models using Grid Search for the optimum hyper-parameter values.
- Employ dimensionality reduction techniques since the word matrix is sparse.
- Address class imbalance by employing sampling techniques (upsampling and downsampling) thus reducing data entries to balance the classes out.
- Use enhanced Word embeddings – Word2Ved and Glove

References and sources:

1. Peter F. Brown et al.1990. A Statistical Approach to Machine Translation, Computational Linguistics
2. Kevin Knight, Graehl Jonathan.1992. Machine Transliteration. Computational Linguistics
3. Dekai Wu.1997. Inversion Transduction Grammars and the Bilingual Parsing of Parallel Corpora, Computational Linguistics
4. Ronan Collobert et al.2011. Natural Language Processing (almost) from Scratch, J. of Machine Learning Research
5. Ahmad Aghaebrahimian and Mark Cieliebak 2019. Hyperparameter Tuning for Deep Learning in Natural LanguageProcessing
6. RVK.2019. Always start with “Text EDA In Classification Problem”
7. Susan Li.2019. A Complete Exploratory Data Analysis and Visualization for Text Data
8. Pascal Pompey.2018. The art of deep learning (applied to NLP)
9. Lidan Wang, Minwei Feng, Bowen Zhou, Bing Xiang, Sridhar Mahadevan.2015. Efficient Hyper-parameter Optimization for NLP Applications
10. Datanizing GmbH.2019. Modern Text Mining with Python, Part 2 of 5: Data Exploration with Pandas
11. https://towardsdatascience.com/multi-class-metrics-made-simple-part-ii-the-f1-score-ebe8b2c2ca1
12. https://towardsdatascience.com/precision-vs-recall-386cf9f89488
13. http://shatterline.com/blog/2013/09/12/not-so-naive-classification-with-the-naive-bayes-classifier/
14. http://scikit-learn.sourceforge.net/0.8/modules/svm.html
15. https://www.researchgate.net/figure/Architecture-of-the-random-forest-model_fig1_301638643
16. https://www.datacamp.com/community/tutorials/random-forests-classifier-python#advantages
17. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

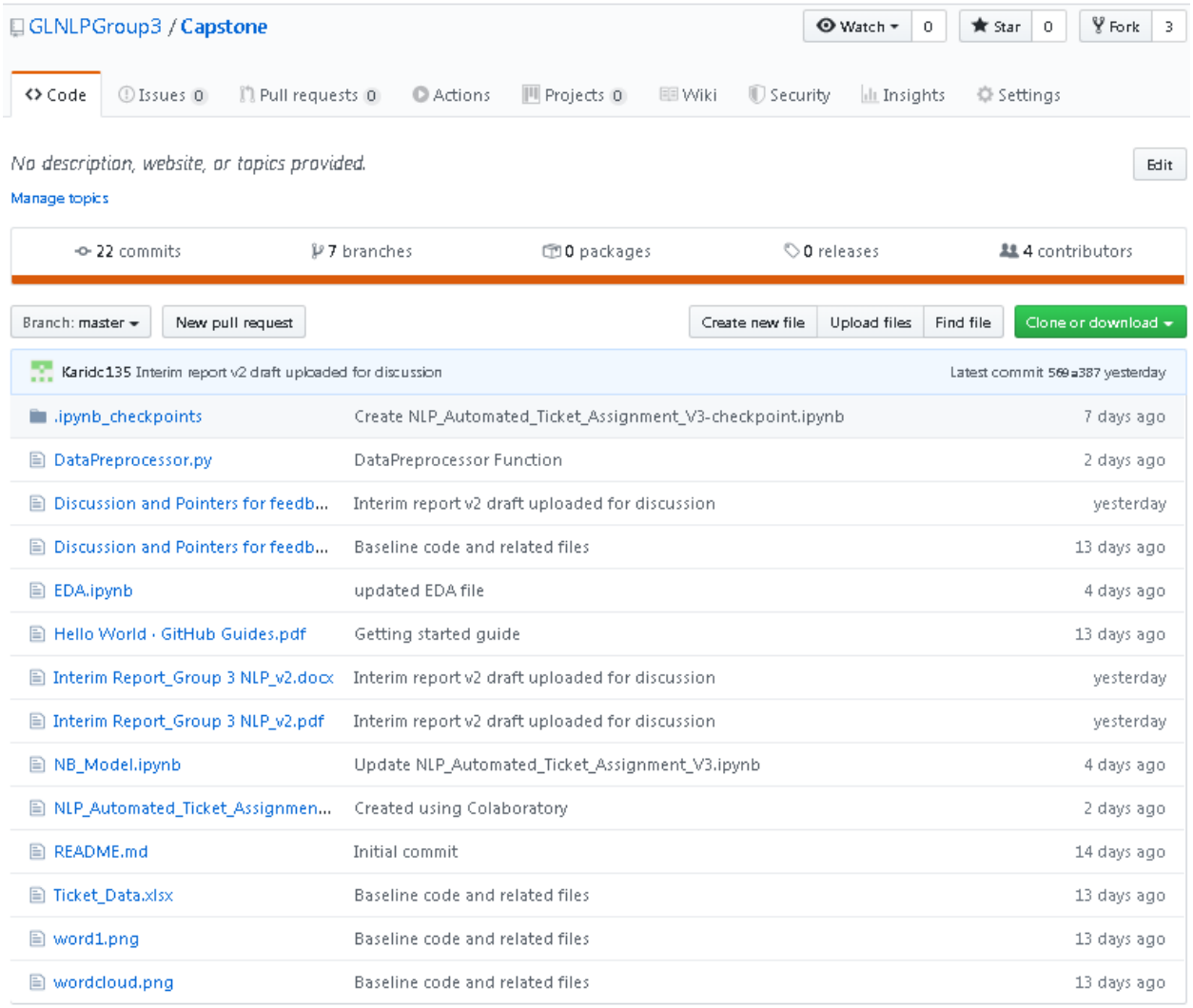
Challenges, Approach and Mitigation:

Item	Challenge	Approach	Mitigation
Hardware	Personal machines used for the project as limited in storage and processing power.	Find easy and free platforms with sufficient hardware and processing support.	The platform which was used to achieve the task was - Google Colab which provides around 15GB of Storage space on the Google Drive as well as its GPU which empowers us to train and test our models effectively.
Data	Data is the most importance piece of the puzzle with regards to Machine Learning(ML) problems. Our observations are as follows:		
Imbalance	Data imbalance was observed with Group_0 followed by group_8 being over-representative in the whole of the dataset.	Sampling techniques would enable us to down sample the majority classes or/and upsample the minority classes.	In milestone 1, we tackled the problem by using a tuned threshold value which allows us to control the entries that enter the training set. However, there are drawbacks to this method which urges us to focus on sampling techniques.
Noisy Data	Data collected from the systems would be noisy with extra characters like punctuations, html texts, special characters etc.	Cleaning data is the primary task to any data modelling problem. We spend a considerable amount of time cleaning the data and preparing it for modelling.	In the view of preparing the data for modelling, we must first clean the data. This has been accomplished using NLTK and RE (regular expressions) Libraries.

Multi-lingual data	Besides English, we observed non-English text in the dataset.	We checked if libraries from Google Translate and other language translation modules would work for our purpose. However, the limitations exceeded the cause. As a part of the text processing activity, English text has been considered and any other non-English text was dropped.	For milestone 1, we considered only English text for the processing and would address the non-English text either through a translation or a mechanism to map the same using word mappings. The findings would be potentially shared in Milestone 2.
Collaboration	The team is located in different parts of the country and the course is completely online which posed a challenge in communication and collaboration.	As a team, the primary goal is to be able to share data between the team, communicate effectively and thereby work together.	We used the following platforms which proved efficient in meeting our team's expectations and goals. a. Github b. Google Drive c. Telephony, Whatsapp groups, etc.

Code and Deliverables:

1. Interim report –
    a. PDF format - Interim Report_Group 3 NLP_Final.pdf
2. Filenames listed, attachments in the Great Learning (GL) portal
    a. NLP_Automated_Ticket_Assignment_Baseline.ipynb
    b. NLP_Automated_Ticket_Assignment_Baseline.html
3. Snapshot of Github Repo (Capstone)- https://github.com/GLNLPGroup3/Capstone



Milestone 2 Plan - Future Scope of this Project

1. Data Imbalance – The method we would employ is upsampling and downsampling techniques. After analyzing the data, it seems obvious that downsampling the majority class of Group_0 an d Group_8 would lead to better results than upsampling the minority groups. However, the results of this are yet to be proved.
2. Utilize enhanced embedding techniques - Word embeddings using Word2vec and Glove to comprehend textual contexts.
3. Employ DeepNets and transfer learning modelling techniques. Among the algorithms to be considered, the following are potential candidates:
    a. Long Short-Term Memory (LSTM)
    b. Recurrent Convolutional Neural Networks (RCNN)
    c. Random Multimodel Deep Learning (RMDL)..etc
4. It has been observed that languages besides English are present in the dataset. For milestone 1, we considered English text for the processing and would address the non-English text either through a translation or a mechanism to map the same using word mappings. The potential findings would be shared in Milestone 2.
5. Modularize the code into packages to maintain code re-usability and independence.
6. Meet the expectations and prepare the deliverables for Milestone 2.

<END OF REPORT>