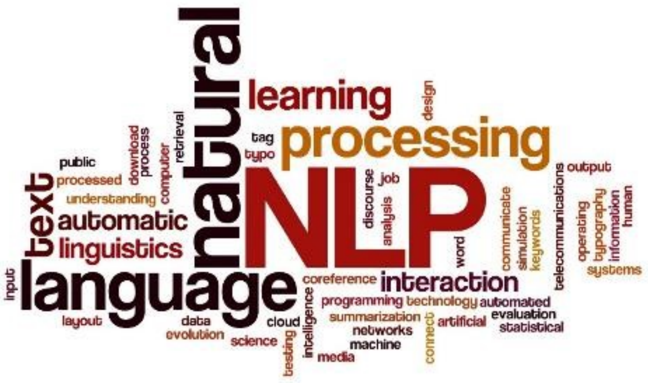
Natural Language Processing (NLP) Application Capstone Project

April 19, 2020

Automated Ticket Assignment



Program	Post Graduate Program (PGP) in Artificial Intelligence and Machine Learning(AIML)
Batch	April 2019 – April 2020
Group	3
Project Guide	Sanjay Tiwary
Program Manager	Anurag Shah
Team Members	Gaurav Walia, Karishma Dcosta, Lavanya Harry Pandian, Kumari Pallavi, Swati Tyagi
Deliverable	Final Report
Submission Date	19th April 2020

Contents

1 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.

2 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.

2.1 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 or poor customer service.

Opportunity for improvement through ML

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

- Reduce or 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

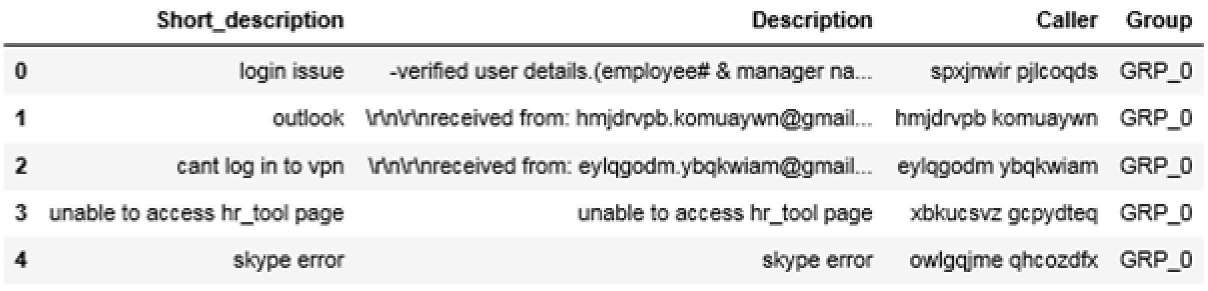
By leveraging the AI technology, we shall build a classifier that can classify the tickets into respective Groups by analyzing the text using NLP techniques in AIML.

2.2 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 contain various languages since the users could potentially be located globally. **Data Requirement** 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. Further, data should be pre-processed to remove words which do not contribute to context (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

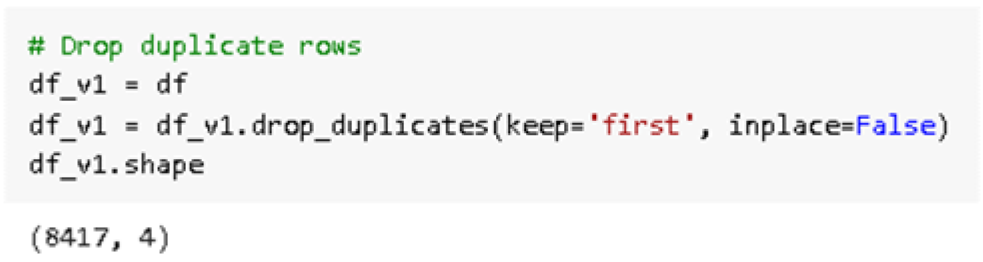
The methods employed to clean our data are used from the NLTK and Regular Expression (re) library.

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.

Class Imbalance

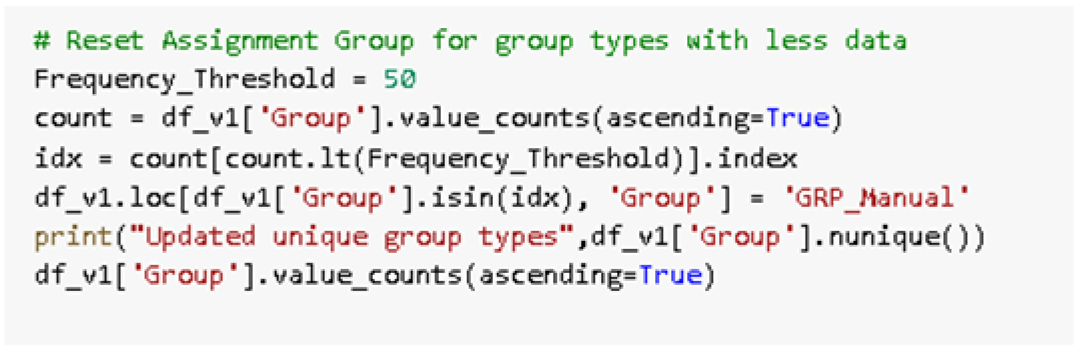
Datasets require proper representation of Class information, i.e equal representation of all Groups. This would enable the modelling algorithms to be trained on equal amounts of data of any given



```
# 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 (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

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.

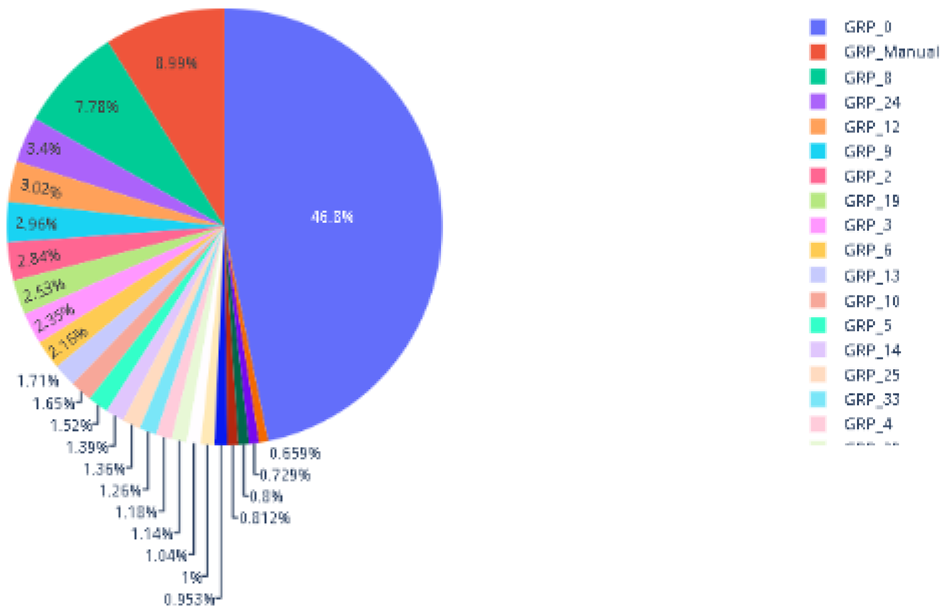
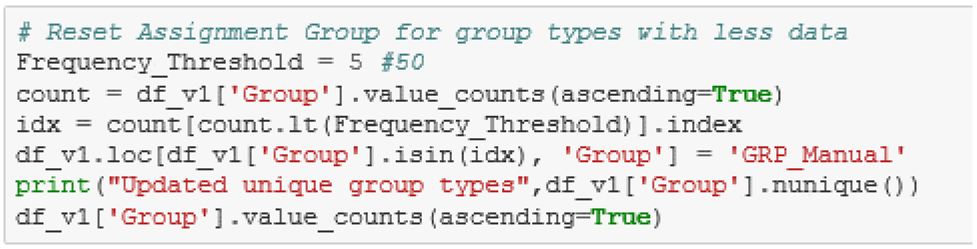


Figure 4: Out of the total 8500 tickets, 47% represents Group_0 tickets



```
# 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)
```

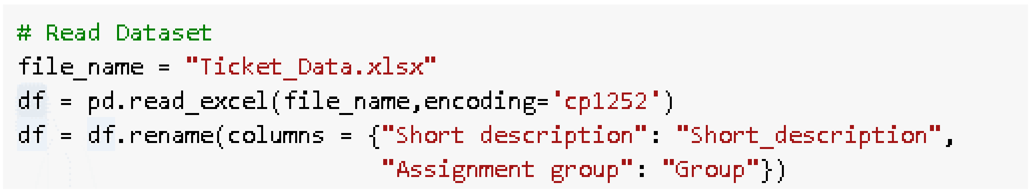
Figure 5: Setting a threshold=5 to control the class imbalance during modelling

3 Summary of the Approach to EDA and Pre-processing

3.1 Analyze and understand the structure of data

Reading the dataset

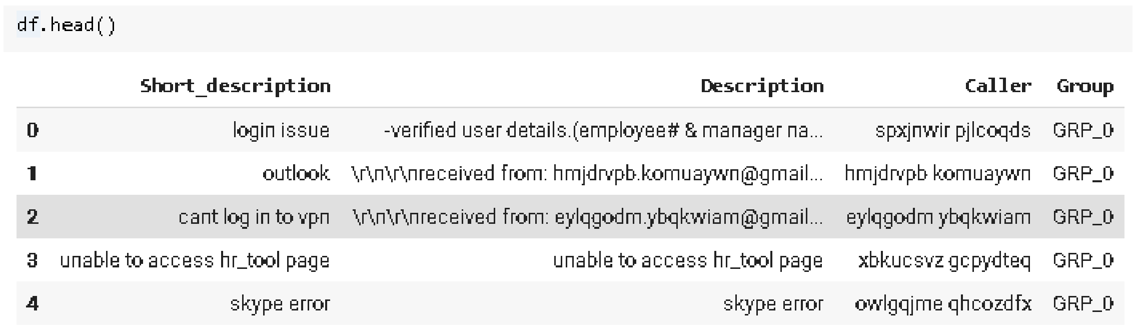
The dataset is located in the google drive and accessed using Pandas library. This file is then stored in a dataframe. While reading the file, 'Assignment group' is renamed to 'Group' and 'Short description' to 'Short_description' as given below.



```
# 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

Viewing dataframe with dataframe.head()



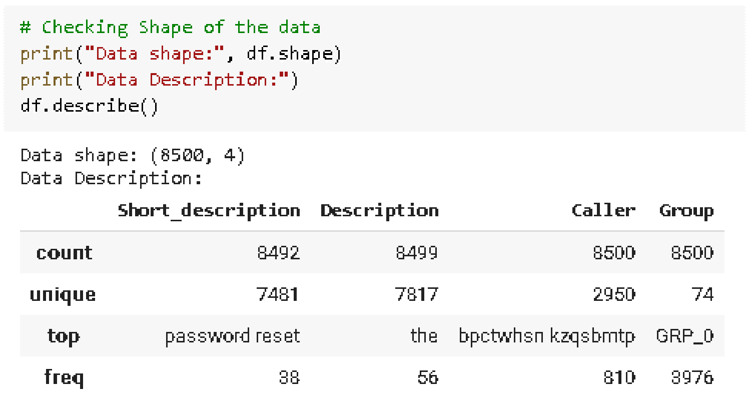
```
df.head()
```

	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 analyze the shape of the data to get a basic understanding of the data and features by using dataframe.shape and dataframe.describe() respectively.

- 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 denotes the unique values in each column Eg. There are 74 unique groups in the dataframe.
- Displays the top word or 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()
```

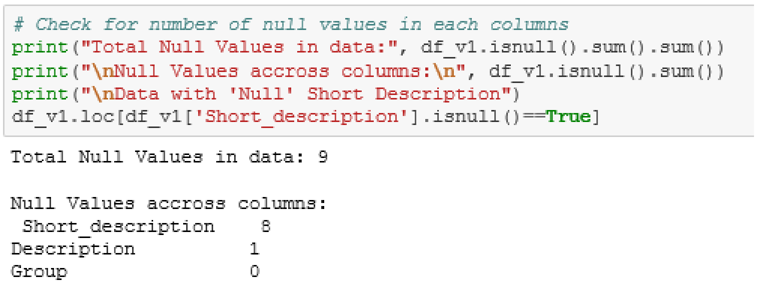
```
Data shape: (8500, 4)
Data Description:
```

	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

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 a stop word 'the', which will be processed 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]
```

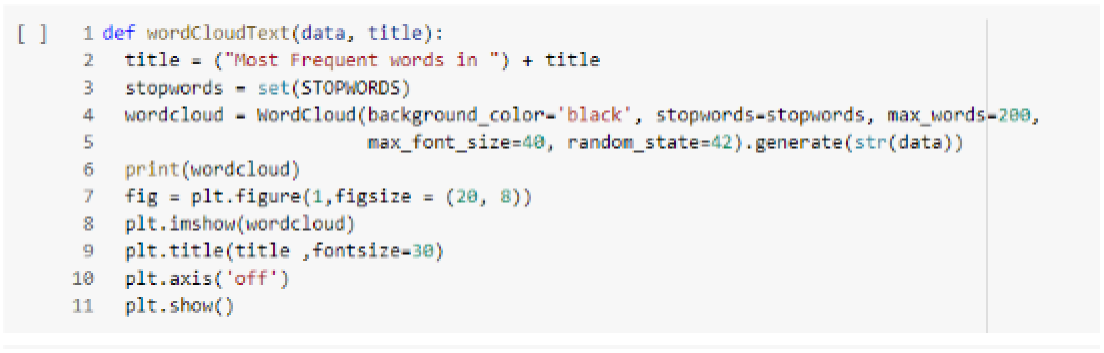
```
Total Null Values in data: 9

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

Figure 9: Checking for missing values and imputing data in the dataset columns

3.2 Visualize Data

Visuals are a great way to analyze data and get an idea about the data that we are handling. There are many packages in python that help visualize data. We have chosen Word Clouds and defined a function to analyze:

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



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

Figure 10: Word Cloud function

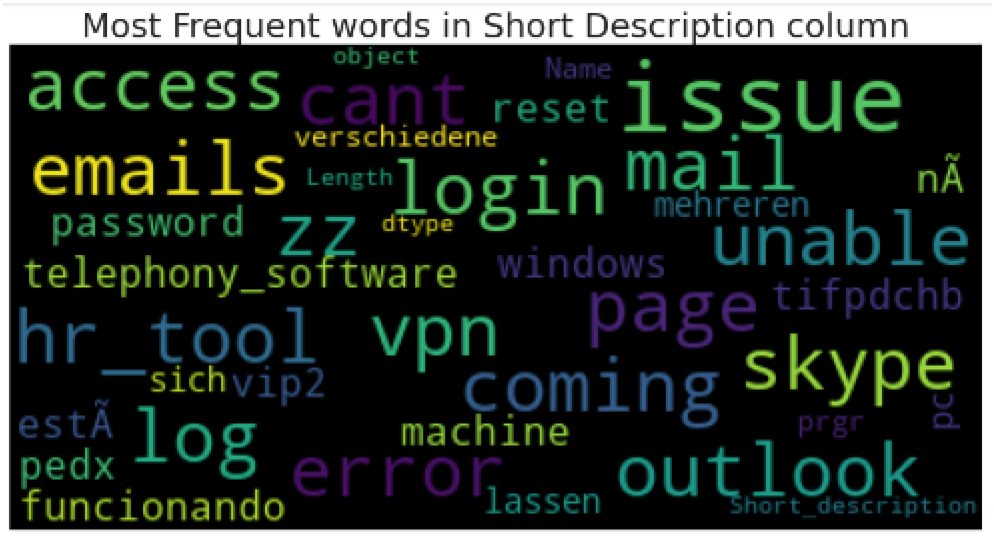


Figure 11: Most frequent words in Short$_descriptionbeforecleaningdata$

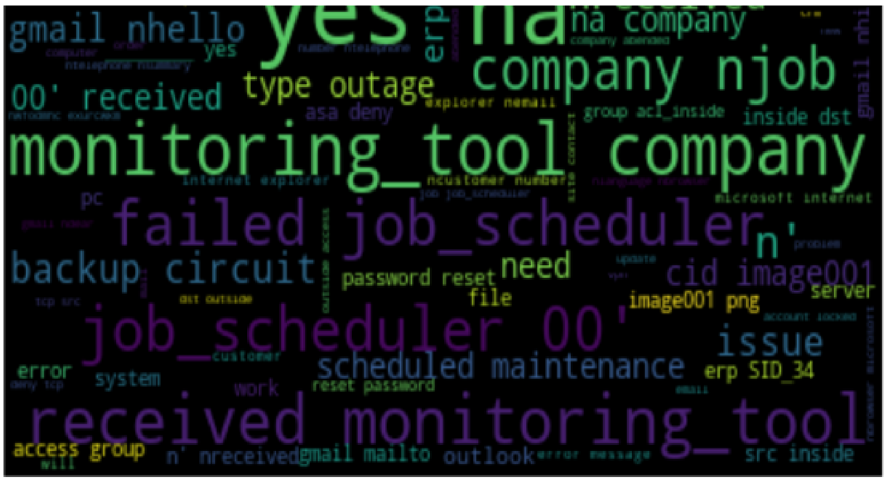


Figure 12: Most frequent words in Description before cleaning data

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.

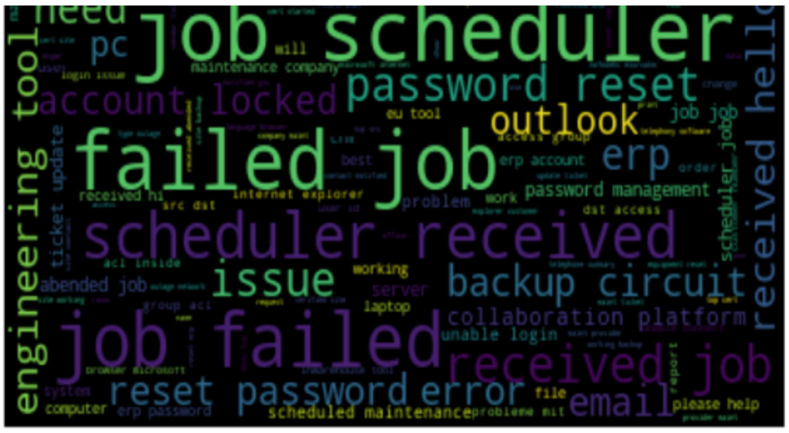
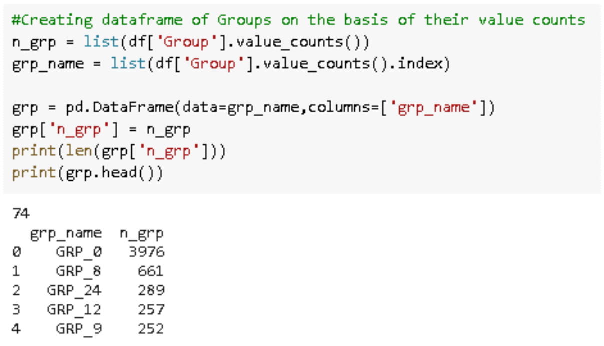


Figure 13: Clean data in Summary field

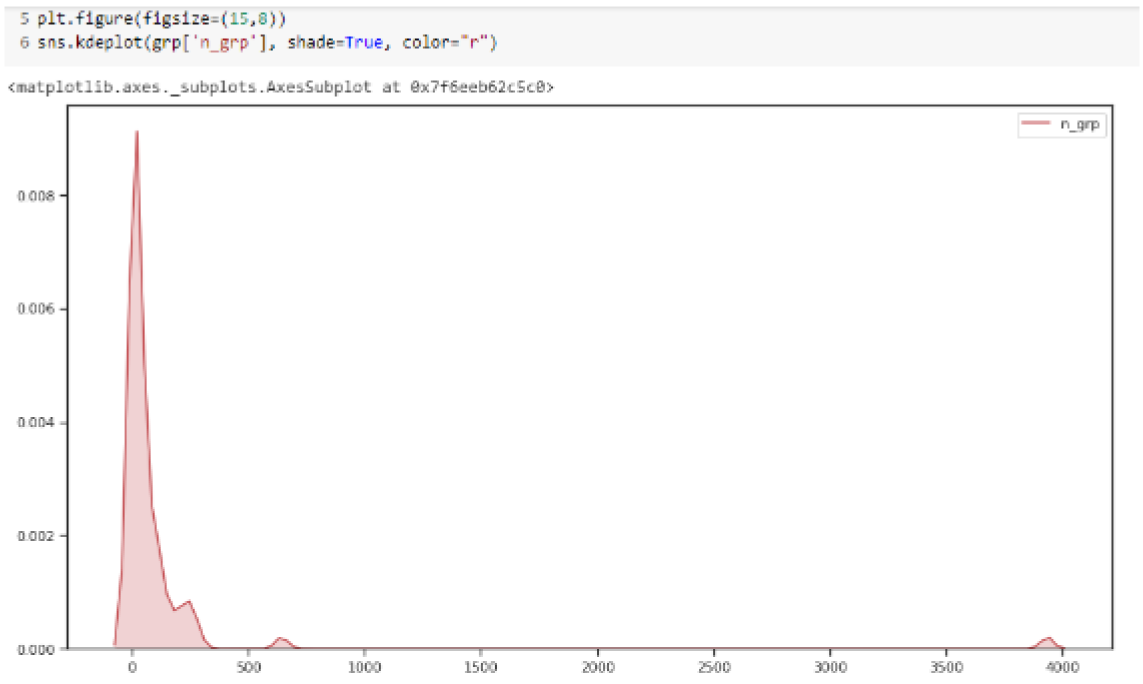


```
#Creating dataframe of Groups on the basis of their value counts
n_grp = list(df['Group'].value_counts())
grp_name = list(df['Group'].value_counts().index)

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 14: Top 5 Groups with their frequencies

The frequency distribution of the Groups present in the dataframe is plotted as below.
After classifying the less assigned groups into one group, Pie plot is further used to understand the distribution of each class in the dataset and found that GRP_0 takes almost 48% of the ticket assignment.



```
5 plt.figure(figsize=(15,8))
6 sns.kdeplot(grp['n_grp'], shade=True, color="r")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6eeb62c5c0>
```

Figure 15: kde representing the frequency distribution of the groups

The "Column – Caller" is anonymously provided in the dataset and hence it is difficult to comprehend. As seen in Figure 17 Caller Data anonymously provided in the dataset

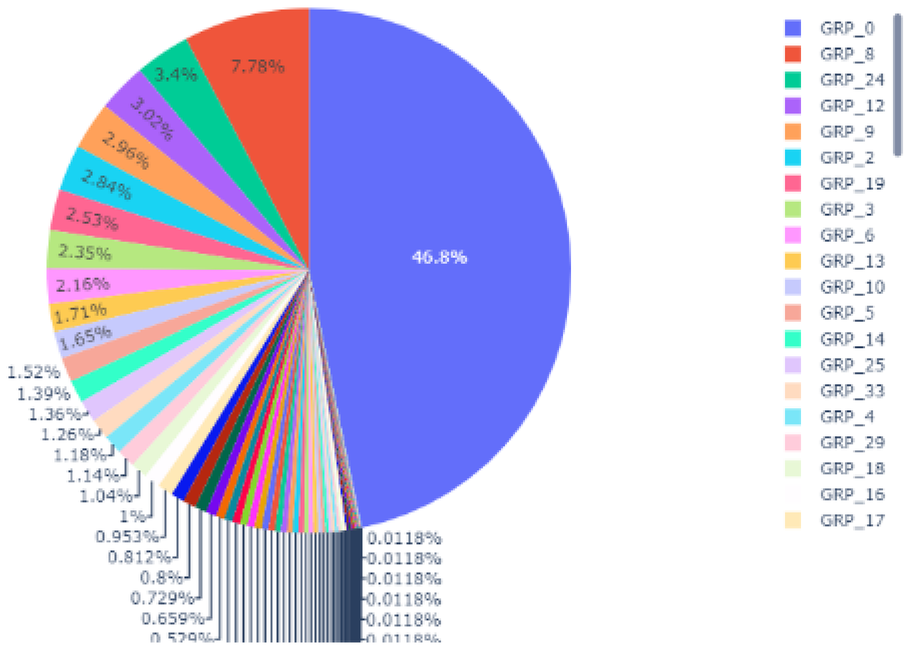
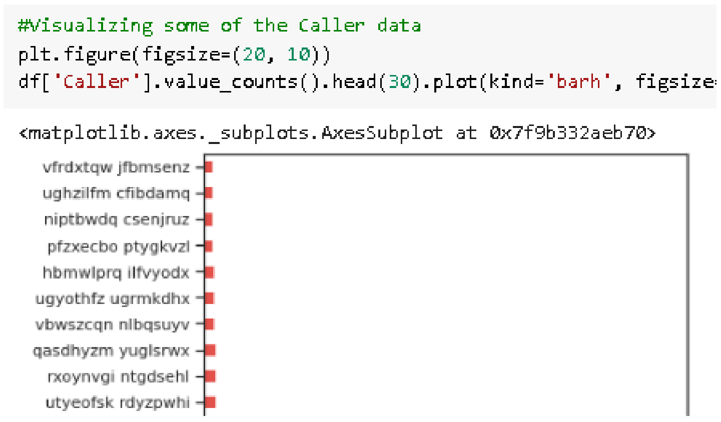


Figure 16: pie chart representation of Groups with Threshold VALUE = 50

The Caller data is then renamed with the word ‘Caller’ followed by an incremental number depending on the frequency of assignment. Eg: Caller1, Caller2, etc., Caller 1 has maximum ticket logged. The top 10 Caller information is displayed as below.

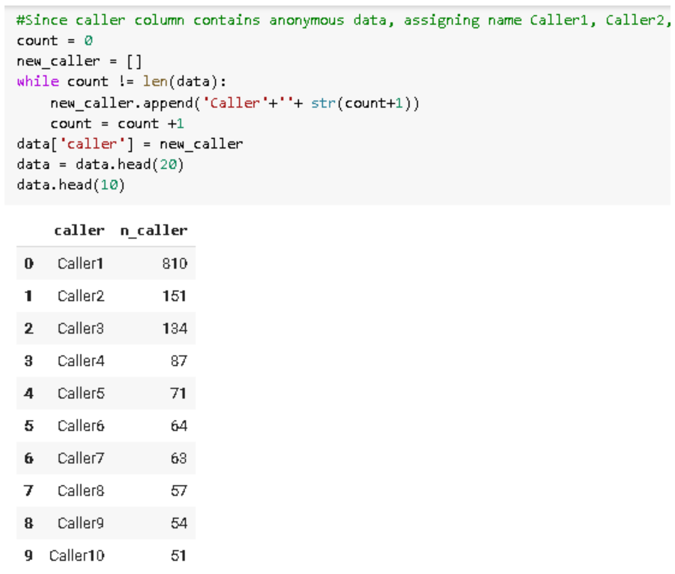


```
#Visualizing some of the Caller data
plt.figure(figsize=(20, 10))
df['Caller'].value_counts().head(30).plot(kind='barh', figsize
```

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

Figure 17: Caller Data anonymously provided in the dataset

This graph depicts the Caller data with frequency of tickets logged.



```
#Since caller column contains anonymous data, assigning name Caller1, Caller2,
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 18: Caller frequency

Analysing the caller data, we see that one caller has reported 810 tickets, and the distribution is skewed. This could be a batch job or an anomaly and should be considered for discussion with domain expert. We are not taking up further analysis of caller for this project, for now.

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

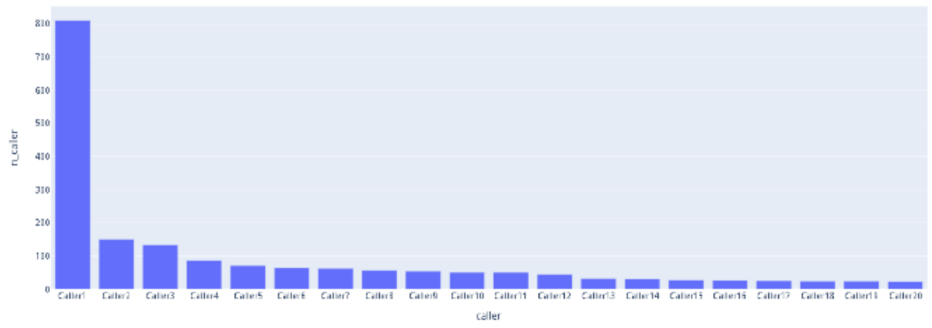
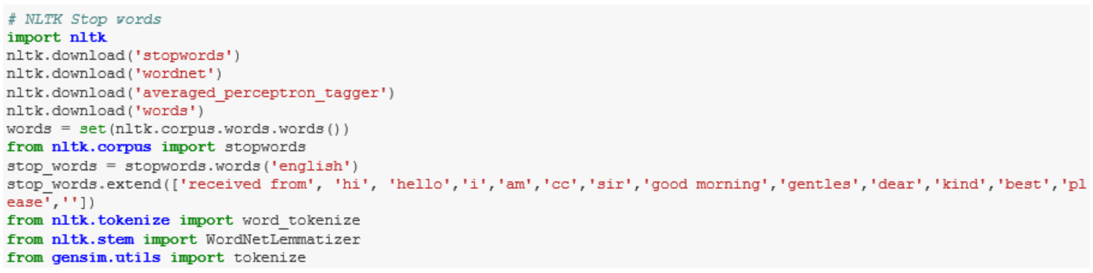


Figure 19: Barplot representing the Caller Frequency Data for Top 20 Callers

4 Data 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
```

Figure 20: Utilizing the NLTK and re Library to clean the data

4.1 Addressing Noise

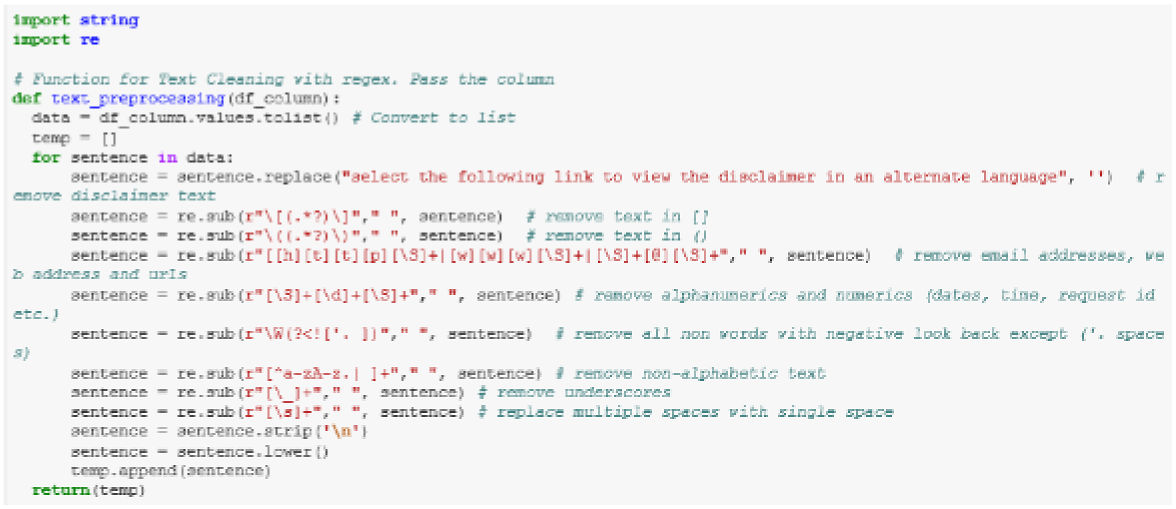
Stop-words

The text dataset includes many words such as 'a', 'about', 'about', 'after', 'again',.., which do not add any significance in model building. The most common technique to deal with these words is to remove them from the texts and documents.

Punctuations removal

Text documents generally contains characters like punctuations or special characters and they are not necessary for text mining or classification purposes. Even though punctuation is critical to understand the meaning of the sentence, 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 21: Noise Removal from text

Capitalization

Sentences can contain a mixture of uppercase and lowercase words. In our dataset, we do not want to create different tokens for capitalized words. Hence we change the words to follow lowercase this brings all words in a document in same space. In some cases where capitalization 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, named identity, slang or abbreviation converters can be applied.

Concatenation of all text to a new field

‘Summary’. The preprocessed data is then concatenated into a single column called ‘Summary’.

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.

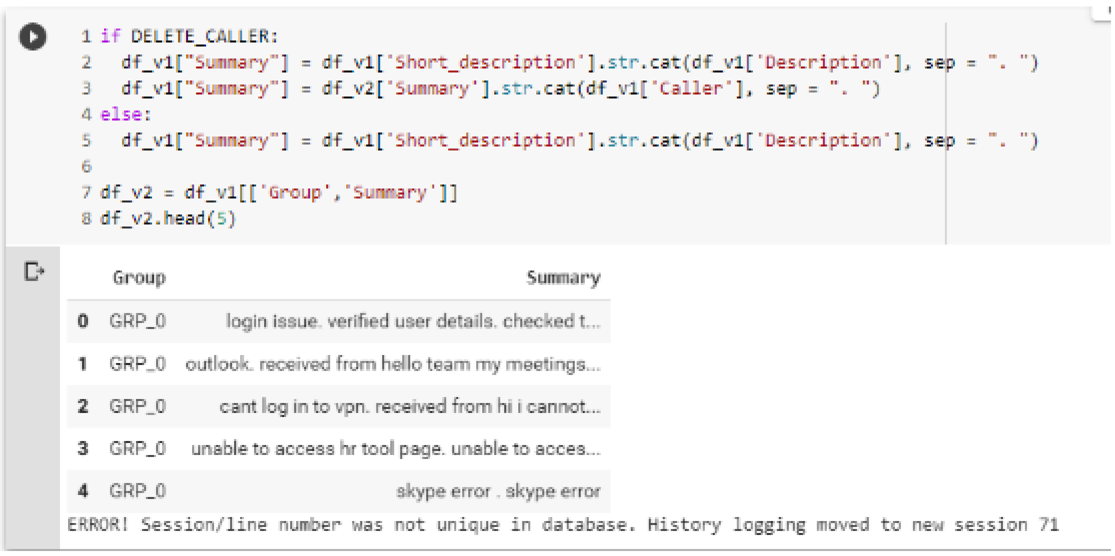
4.2 Create Word Vocabulary Tokens

Tokenization

Tokenization refers to text segmentation or lexical analysis where the large chunk of text or sentence is split into words, phrases, symbols, or 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’

Lemmatization

Text lemmatization is the process of eliminating redundant prefix or suffix of a word and extracting

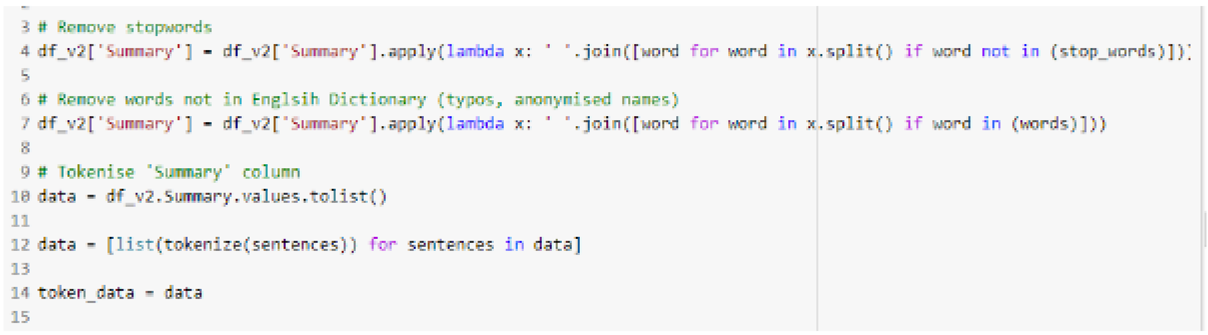


```python
if DELETE_CALLER:
  df_v1["Summary"] = df_v1['Short_description'].str.cat(df_v1['Description'], sep = ". ")
  df_v1["Summary"] = df_v2['Summary'].str.cat(df_v1['Caller'], sep = ". ")
else:
  df_v1["Summary"] = df_v1['Short_description'].str.cat(df_v1['Description'], sep = ". ")

df_v2 = df_v1[['Group','Summary']]
df_v2.head(5)
```

	Group	Summary
0	GRP_0	login issue. verified user details. checked t...
1	GRP_0	outlook. received from hello team my meetings...
2	GRP_0	cant log in to vpn. received from hi i cannot...
3	GRP_0	unable to access hr tool page. unable to acces...
4	GRP_0	skype error . skype error

```
ERROR! Session/line number was not unique in database. History logging moved to new session 71
```

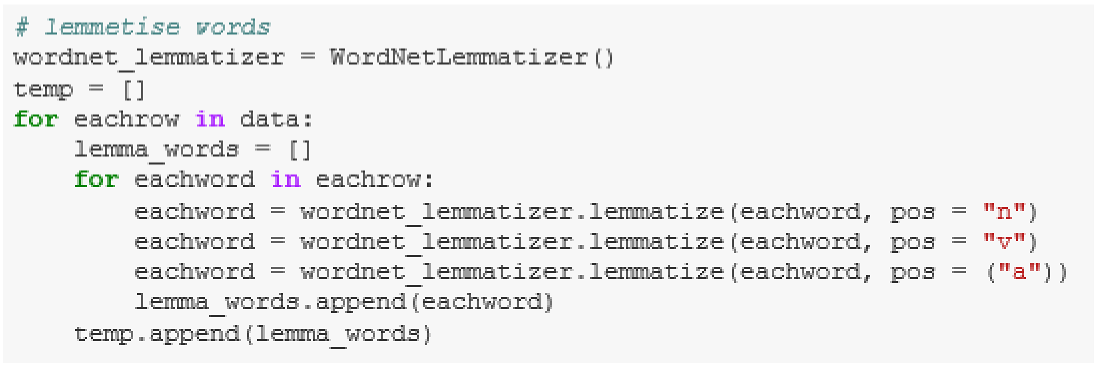
Figure 22: Concatenation of text into a new field – ‘Summary’



```python
# 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]

token_data = data

```

Figure 23: Utilizing Tokenize to create word tokens

the base word (lemma).
Eg: 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 24: Lemmatizing words

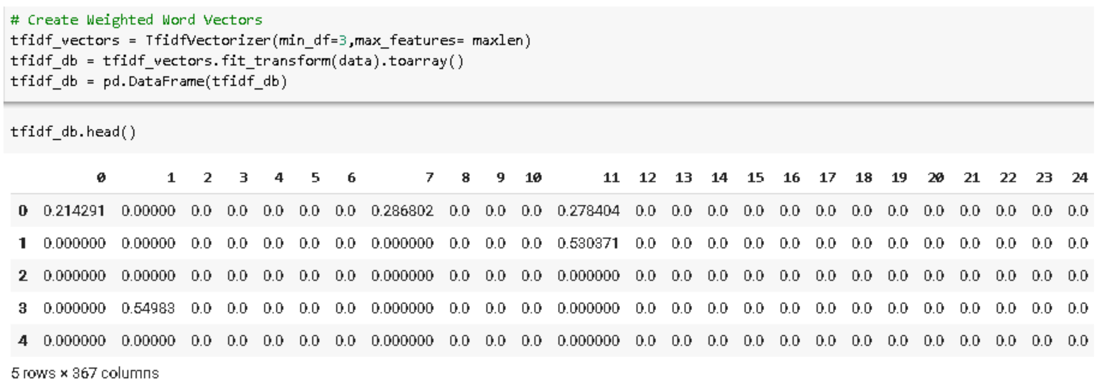
Weighted Words

The most basic form of weighted word feature extraction is TF (Term Frequency), 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 logarithmic 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 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 consumable 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)
```

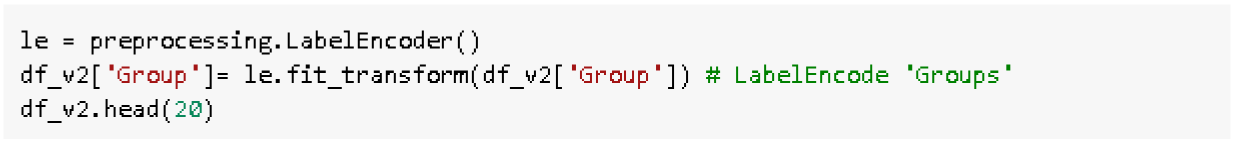
```
tfidf_db.head()
```

```
          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

5 Train Test Split and Evaluation

Before we proceed to build the model, we encode the Group information labelled as GRP_X as 0,1. . . using label encoder(). This field represents the Target column.

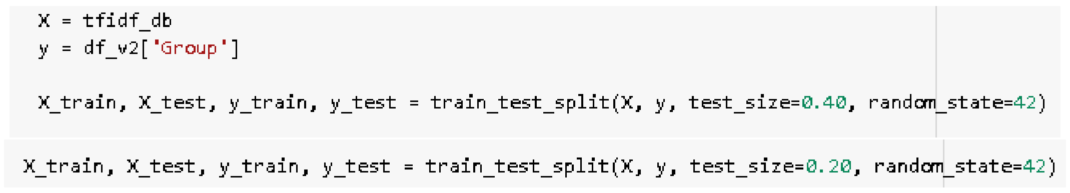


```
le = preprocessing.LabelEncoder()
df_v2['Group']= le.fit_transform(df_v2['Group']) # LabelEncode 'Groups'
df_v2.head(20)
```

Figure 26: Encoding the Group Data using LabelEncoder

Train Test split

We then map the X(input data) to the vectorized data and y(target) to 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 data size.



```
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).

Deciding Evaluation Criteria

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

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

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

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

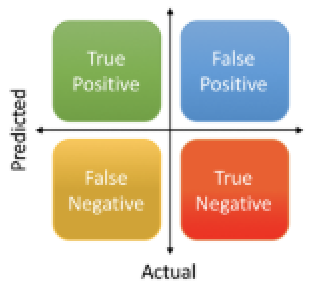


Figure 28: Recall, precision, and confusion matrix

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:

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

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.

6 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, Random Forests, and Ensemble. 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 -Long Short-Term Memory (LSTM), Recurrent Convolutional Neural Networks (RCNN), 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.

6.1 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.

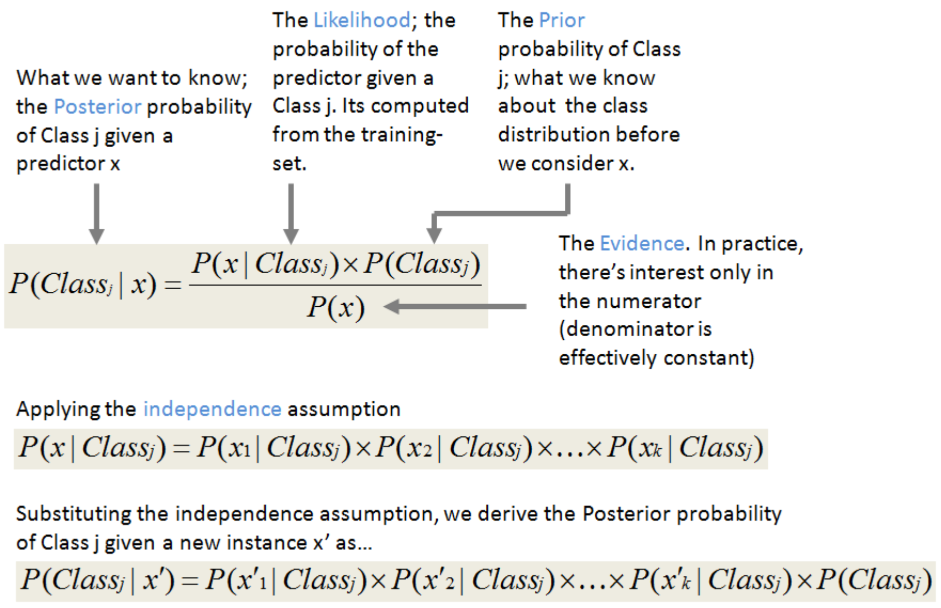

What we want to know; the Posterior probability of Class j given a predictor x
The Likelihood; the probability of the predictor given a Class j. Its computed from the training-set.
The Prior probability of Class j; what we know about the class distribution before we consider x.
The Evidence. In practice, there's interest only in the numerator (denominator is effectively constant)
$P(Class_j \mid x) = \frac{P(x \mid Class_j) \times P(Class_j)}{P(x)}$
Applying the independence assumption
$P(x \mid Class_j) = P(x_1 \mid Class_j) \times P(x_2 \mid Class_j) \times \ldots \times P(x_k \mid Class_j)$
Substituting the independence assumption, we derive the Posterior probability of Class j given a new instance x' as...
$P(Class_j \mid x') = P(x'_1 \mid Class_j) \times P(x'_2 \mid Class_j) \times \ldots \times P(x'_k \mid Class_j) \times P(Class_j)$

Figure 29: Explanation of Naive Bayes, Source: http://shatterline.com/blog/2013/09/12/not-so-naive-classification-with-the-naive-bayes-classifier/

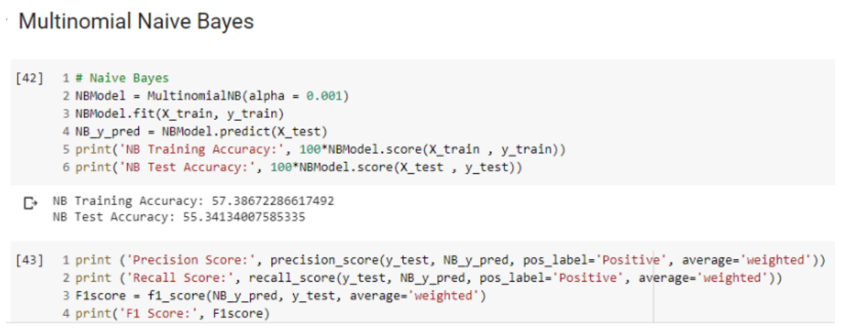
Pros:

- 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.



Multinomial Naive Bayes

```
[42]
# Naive Bayes
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: 57.38672286617492
NB Test Accuracy: 55.34134007585335
```

```
[43]
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)
```

Figure 30: Implementation of Naive Bayes

Result: We can see that the Training accuracy is 57% and testing accuracy is 55% with Naive Bayes Model. The model is able to predict True Positives and False Negatives equally.

6.2 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.

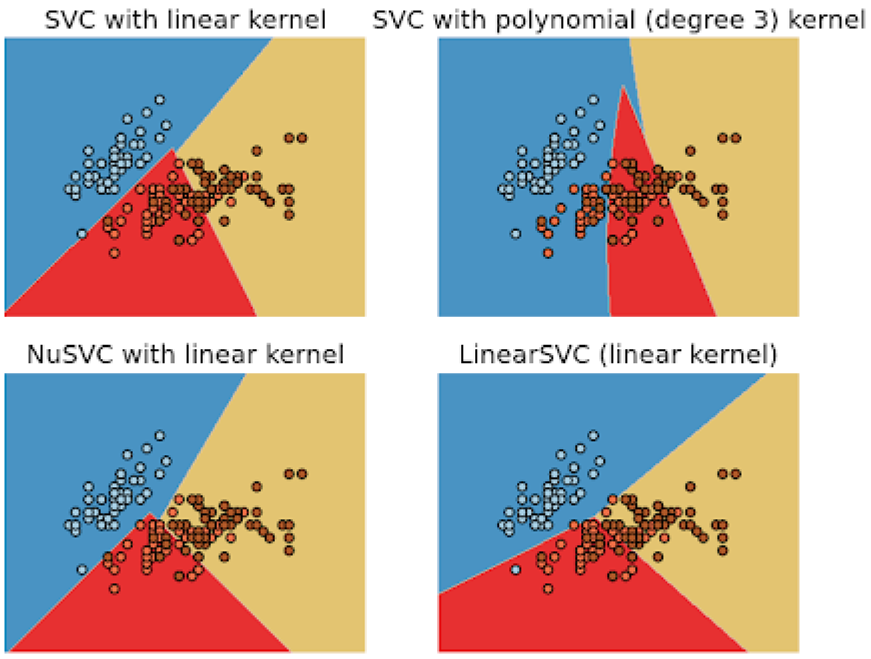


Figure 31: Visual representation of SVC Source: http://scikit-learn.sourceforge.net/0.8/modules/svm.html

We use linear kernel in SVC for this problem as the text are transformed into higher dimensions by using TF IDF and the data in higher dimensions are linearly separable. The linear kernel equation for predicting a new input using the dot product between the input (x) and each support vector (xi) is calculated as f(x) = B(0) + sum(ai * (x,xi))

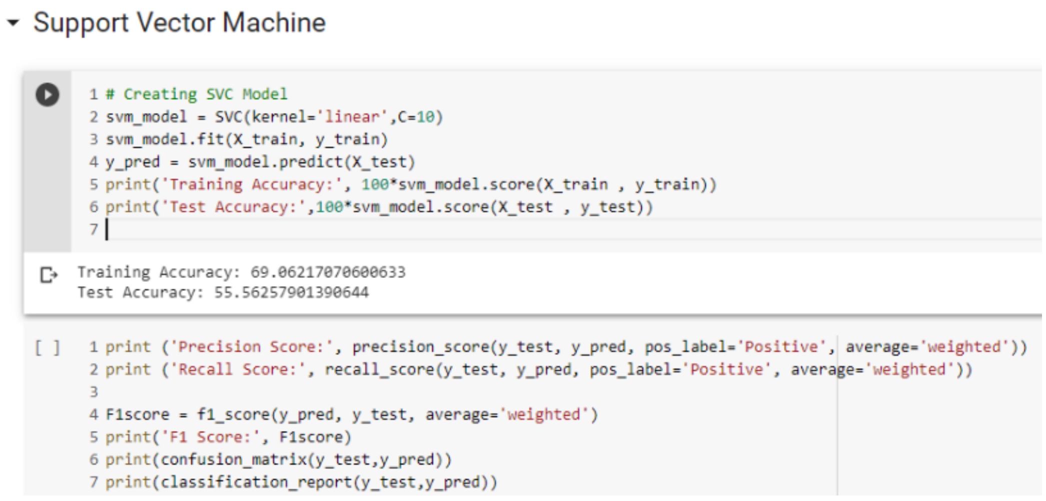
Pros:

- 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


Support Vector Machine

```
# Creating SVC Model
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: 69.06217070600633
Test Accuracy: 55.56257901390644
```

```
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(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))
```

Figure 32: Implementation of SVC

RESULT:
We can see that the Training accuracy is around 69% and testing accuracy is around 55% with SVC. The model is able to predict True Positives and False Negatives equally.

6.3 Decision Tree

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))

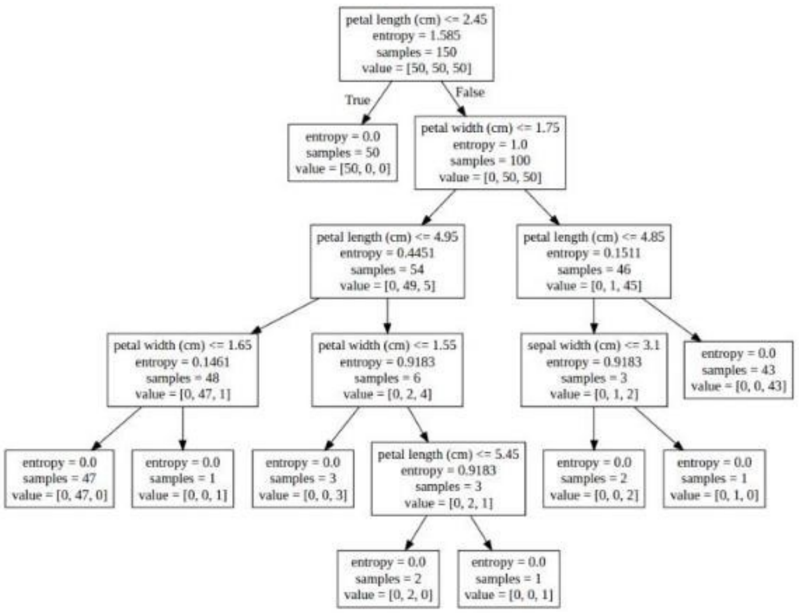


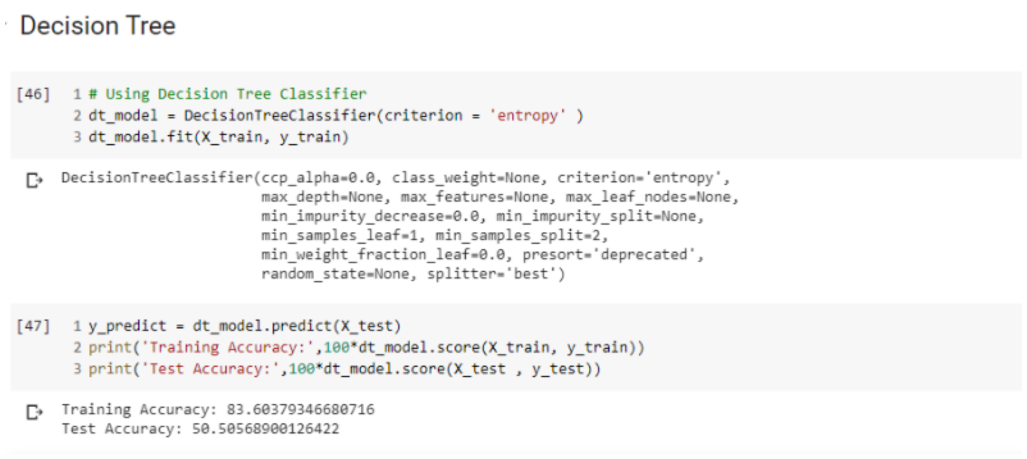
Figure 33: Explanation of Decision Trees source: https://www.kdnuggets.com/2017/05/simplifying-decision-tree-interpretation-decision-rules-python.html

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.



Decision Tree

```
[46]  # Using Decision Tree Classifier
      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')
```

```
[47]  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: 83.60379346680716
Test Accuracy: 50.50568900126422
```

Figure 34: Implementation of Decision Trees

- low prediction accuracy compared to other algorithms and can become complex when there are many class labels.

Result:
We can see that the training accuracy is around 83% and testing accuracy is around 50% with Decision Trees. The model is able to predict True Positives and False Negatives almost equally.

6.4 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.

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:

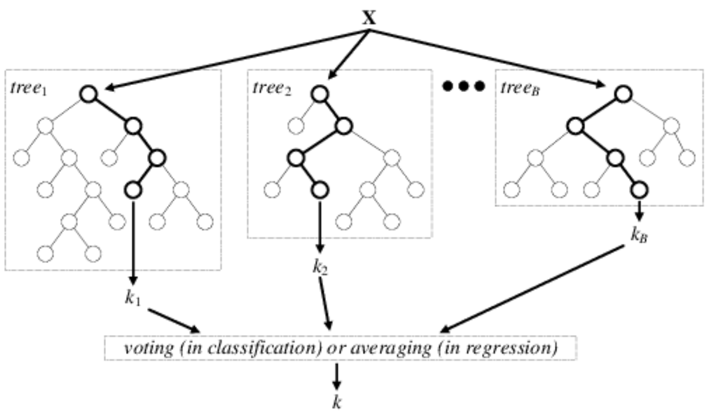
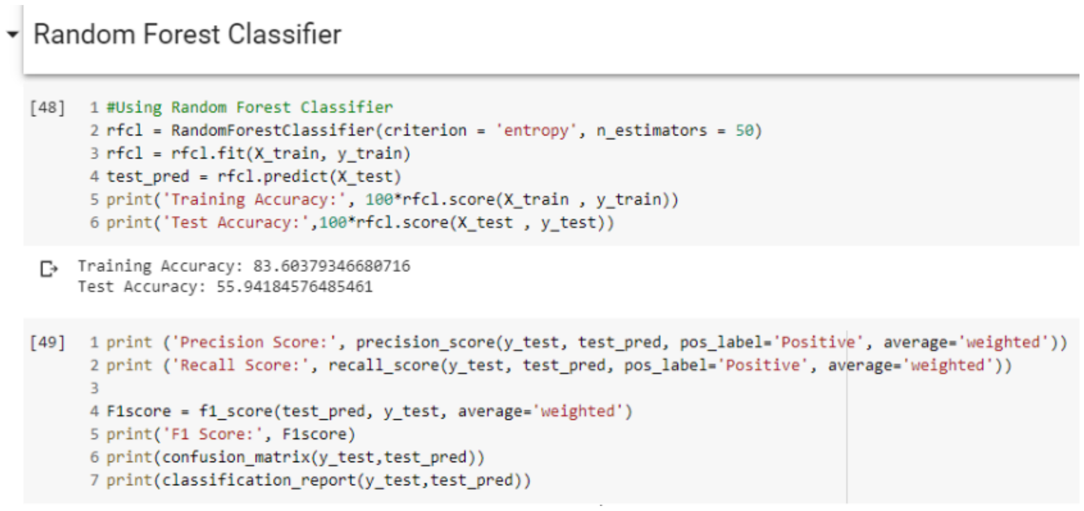


Figure 35: Explanation of Random Forests (Source: https://www.researchgate.net)

- 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.



Random Forest Classifier

```
[48]
#Using Random Forest Classifier
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: 83.60379346680716
Test Accuracy: 55.94184576485461
```

```
[49]
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(confusion_matrix(y_test,test_pred))
print(classification_report(y_test,test_pred))
```

Figure 36: Implementation of Random Forests

Result:

We can see that the training accuracy is around 83% and testing accuracy is around 55% with Random Forests. The model is able to predict True Positives and False Negatives almost equally.

6.5 Ensemble

Most of the errors from a model's learning are from three main factors: variance, noise, and bias. Using ensemble methods, we can increase the stability of the final model and reduce the errors mentioned previously. By combining many models, we can (mostly) reduce the variance, even when they are individually not great, as we will not be affected by random errors from a single source. We have used two common ensemble techniques to solve the Automated Ticket Classification problem assigned in this project

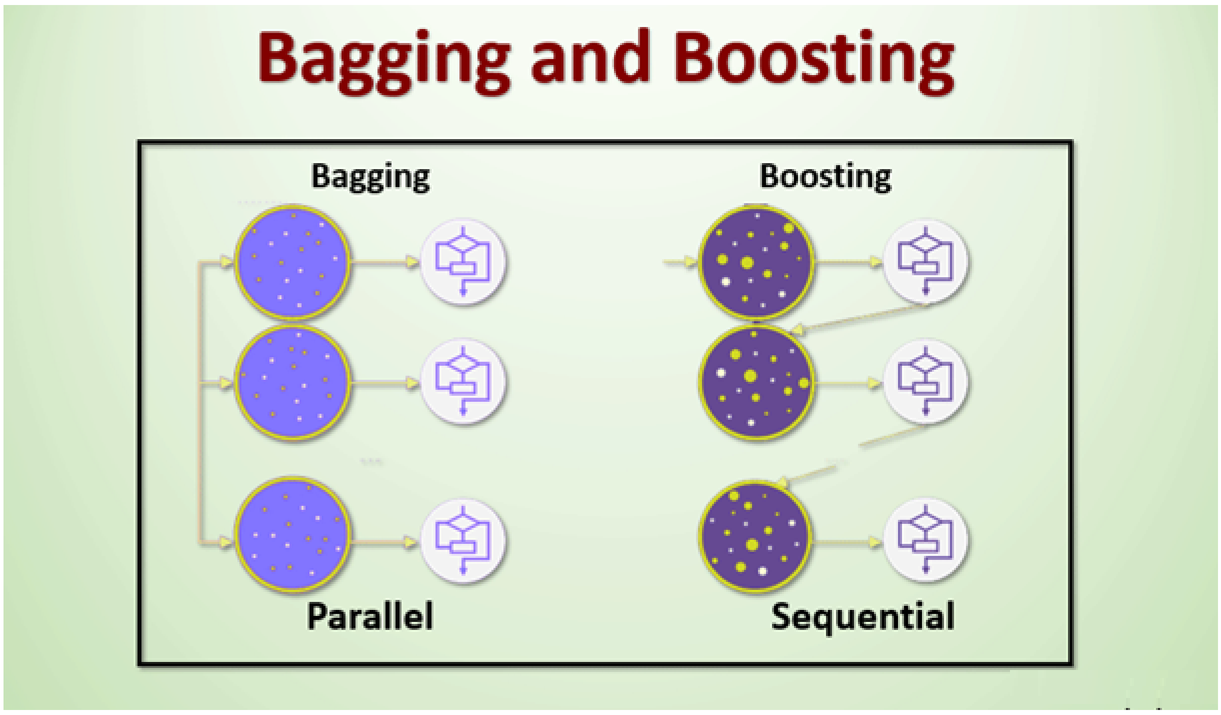
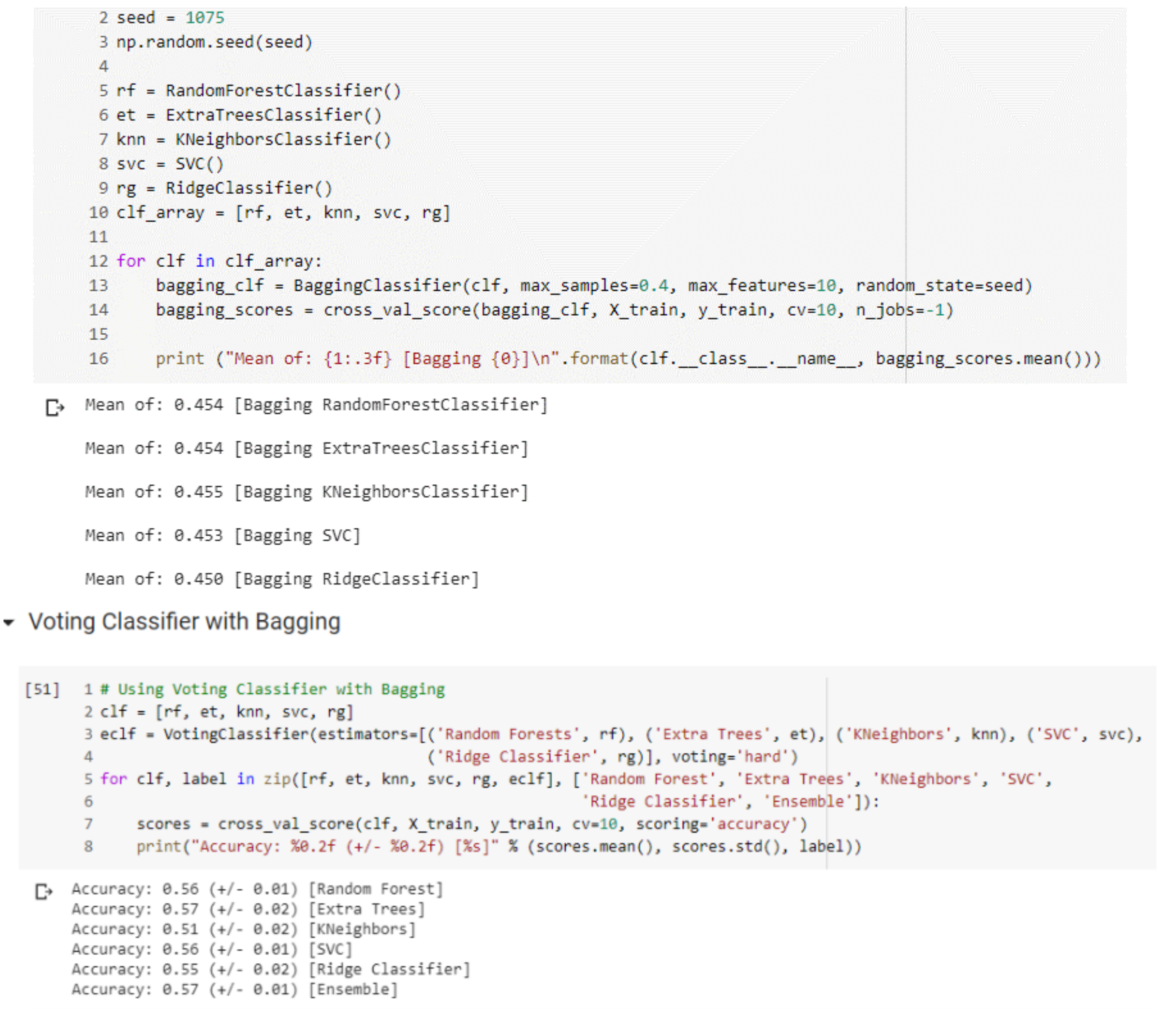


Figure 37: Bagging and Boosting Source: https://www.educba.com/bagging-and-boosting/

Bagging

Bagging is shorthand for the combination of bootstrapping and aggregating. Bootstrapping is a method to help decrease the variance of the classifier and reduce overfitting by resampling data from the training set with the same cardinality as the original set. In bagging there is a tradeoff between base model accuracy and the gain you get through bagging. The aggregation from bagging may improve the ensemble greatly if there is an unstable model. Once the bagging is done, and all the models have been created on (mostly) different data, a weighted average is then used to determine the final score.



```
seed = 1075
np.random.seed(seed)

rf = RandomForestClassifier()
et = ExtraTreesClassifier()
knn = KNeighborsClassifier()
svc = SVC()
rg = RidgeClassifier()
clf_array = [rf, et, knn, svc, rg]

for clf in clf_array:
    bagging_clf = BaggingClassifier(clf, max_samples=0.4, max_features=10, random_state=seed)
    bagging_scores = cross_val_score(bagging_clf, X_train, y_train, cv=10, n_jobs=-1)

    print ("Mean of: {1:.3f} [Bagging {0}]\n".format(clf.__class__.__name__, bagging_scores.mean()))
```

```
Mean of: 0.454 [Bagging RandomForestClassifier]

Mean of: 0.454 [Bagging ExtraTreesClassifier]

Mean of: 0.455 [Bagging KNeighborsClassifier]

Mean of: 0.453 [Bagging SVC]

Mean of: 0.450 [Bagging RidgeClassifier]
```

Voting Classifier with Bagging

```
[51]
# Using Voting Classifier with Bagging
clf = [rf, et, knn, svc, rg]
eclf = VotingClassifier(estimators=[('Random Forests', rf), ('Extra Trees', et), ('KNeighbors', knn), ('SVC', svc),
                                    ('Ridge Classifier', rg)], voting='hard')
for clf, label in zip([rf, et, knn, svc, rg, eclf], ['Random Forest', 'Extra Trees', 'KNeighbors', 'SVC',
                                                     'Ridge Classifier', 'Ensemble']):
    scores = cross_val_score(clf, X_train, y_train, cv=10, scoring='accuracy')
    print("Accuracy: %0.2f (+/- %0.2f) [%s]" % (scores.mean(), scores.std(), label))
```

```
Accuracy: 0.56 (+/- 0.01) [Random Forest]
Accuracy: 0.57 (+/- 0.02) [Extra Trees]
Accuracy: 0.51 (+/- 0.02) [KNeighbors]
Accuracy: 0.56 (+/- 0.01) [SVC]
Accuracy: 0.55 (+/- 0.02) [Ridge Classifier]
Accuracy: 0.57 (+/- 0.01) [Ensemble]
```

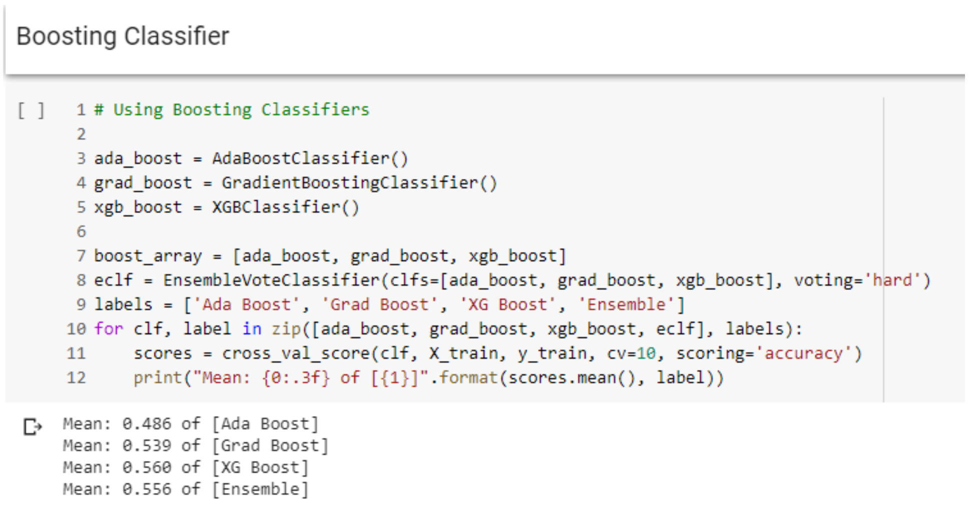
Figure 38: Ensemble model -Bagging

Boosting

The main idea of boosting is to add additional models to the overall ensemble model sequentially. With each iteration of boosting, a new model is created and the new base-learner model is trained (updated) from the errors of the previous learners. The algorithm creates multiple weak models whose output is added together to get an overall prediction and the boosted gradient shifts the current prediction nudging it to the true target. The gradient descent optimization occurs on the output of the various models, and not on their individual parameters.

For Bagging we have used an ensemble of RandomForestClassifier, ExtraTreeClassifier, KNeighborsClassifier, SVM and RidgeClassifier with hard voting, which just need a majority of classifiers to determine what the result could be Please see Figure 38 below. We can see that the training accuracy is around 84% and testing accuracy is around 58% with Random Forests. The model is able to predict True Positives and False Negatives almost equally. We can see that the training accuracy is around 84% and testing accuracy is around 58% with Random Forests. The model is able to predict True Positives and False Negatives almost equally.

Result:
We can see that with Bagging ensemble model, the mean of each classifier used is around 48% and the maximum accuracy when using hard voting is 59% with the chosen classifiers. For Boosting we have used an ensemble of AdaBoostClassifier, GradientBoostingClassifier and XGBClassifier using hard voting from EnsembleVoteClassifier(mlxtend).



Boosting Classifier

```
# Using Boosting Classifiers

ada_boost = AdaBoostClassifier()
grad_boost = GradientBoostingClassifier()
xgb_boost = XGBClassifier()

boost_array = [ada_boost, grad_boost, xgb_boost]
eclf = EnsembleVoteClassifier(clfs=[ada_boost, grad_boost, xgb_boost], voting='hard')
labels = ['Ada Boost', 'Grad Boost', 'XG Boost', 'Ensemble']
for clf, label in zip([ada_boost, grad_boost, xgb_boost, eclf], labels):
    scores = cross_val_score(clf, X_train, y_train, cv=10, scoring='accuracy')
    print("Mean: {0:.3f} of [{1}]".format(scores.mean(), label))
```

```
Mean: 0.486 of [Ada Boost]
Mean: 0.539 of [Grad Boost]
Mean: 0.560 of [XG Boost]
Mean: 0.556 of [Ensemble]
```

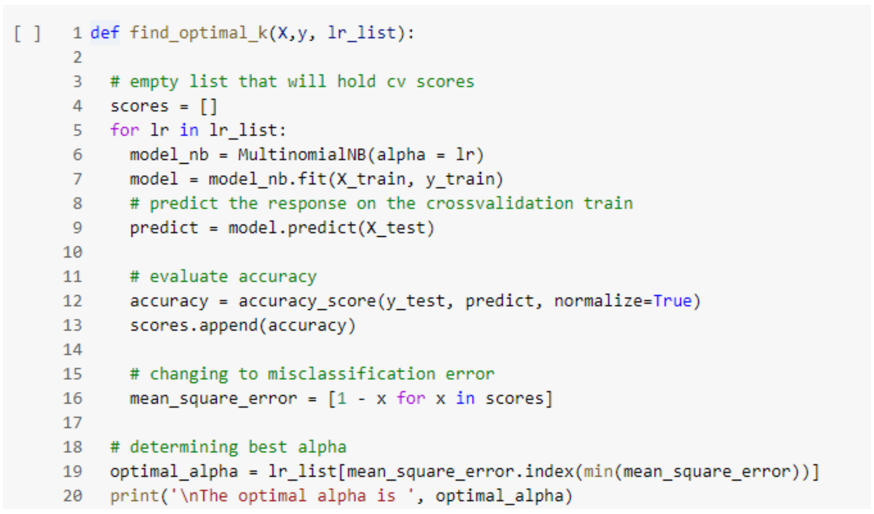
Figure 39: Ensemble model – Boosting

Result:
The maximum accuracy is with 58% with XGBoost classifier. Out of the ensemble models built, bagging with hard voting seem to give slightly higher accuracy of 59

7 Improving Model Performance

Hyperparameters in Machine Learning are user-controlled settings of ML models. These hyperparameters influence how the model's parameters are updated and learned during the training. If the right hyperparameters are set, the model will learn the most optimal weights that it can with a given training algorithm and data. The best hyperparameters are found by trial and error method manually or with Grid Search module

7.1 Naive Bayes



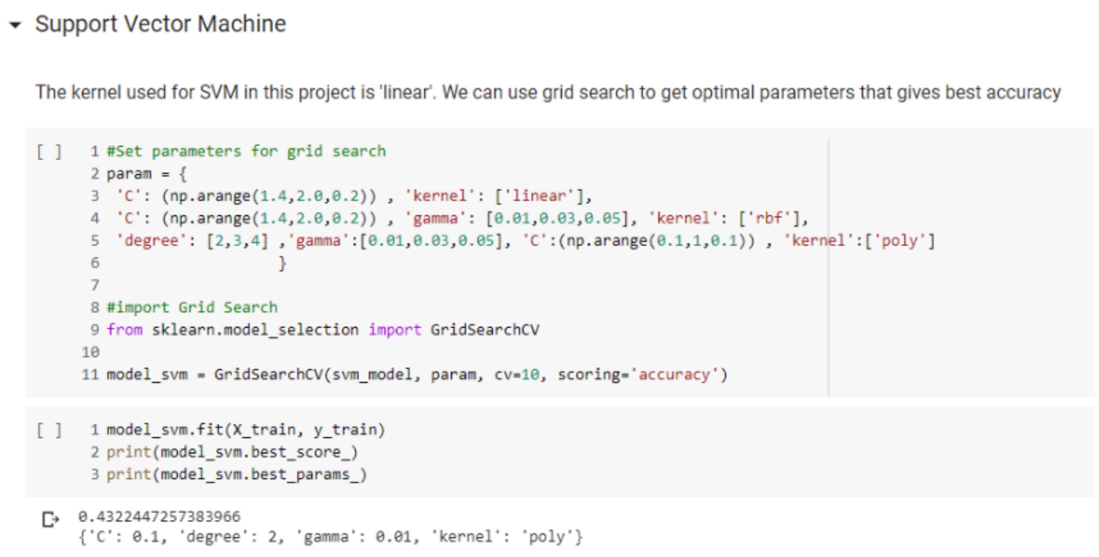
```
[ ] def find_optimal_k(X,y, lr_list):

    # empty list that will hold cv scores
    scores = []
    for lr in lr_list:
      model_nb = MultinomialNB(alpha = lr)
      model = model_nb.fit(X_train, y_train)
      # predict the response on the crossvalidation train
      predict = model.predict(X_test)

      # evaluate accuracy
      accuracy = accuracy_score(y_test, predict, normalize=True)
      scores.append(accuracy)

      # changing to misclassification error
      mean_square_error = [1 - x for x in scores]

    # determining best alpha
    optimal_alpha = lr_list[mean_square_error.index(min(mean_square_error))]
    print('\nThe optimal alpha is ', optimal_alpha)
```

Figure 40: Hyperparameter tuning – Naïve Bayes

Result:

The best alpha found is 0.00001. The best training accuracy is 57% and test accuracy is 55%. With shuffleshift, we found that the least fit time is 0.021 seconds

7.2 Support Vector Classifier



Support Vector Machine

The kernel used for SVM in this project is 'linear'. We can use grid search to get optimal parameters that gives best accuracy

```
#Set parameters for grid search
param = {
 'C': (np.arange(1.4,2.0,0.2)) , 'kernel': ['linear'],
 'C': (np.arange(1.4,2.0,0.2)) , 'gamma': [0.01,0.03,0.05], 'kernel': ['rbf'],
 'degree': [2,3,4] ,'gamma':[0.01,0.03,0.05], 'C':(np.arange(0.1,1,0.1)) , 'kernel':['poly']
                  }

#import Grid Search
from sklearn.model_selection import GridSearchCV

model_svm = GridSearchCV(svm_model, param, cv=10, scoring='accuracy')
```

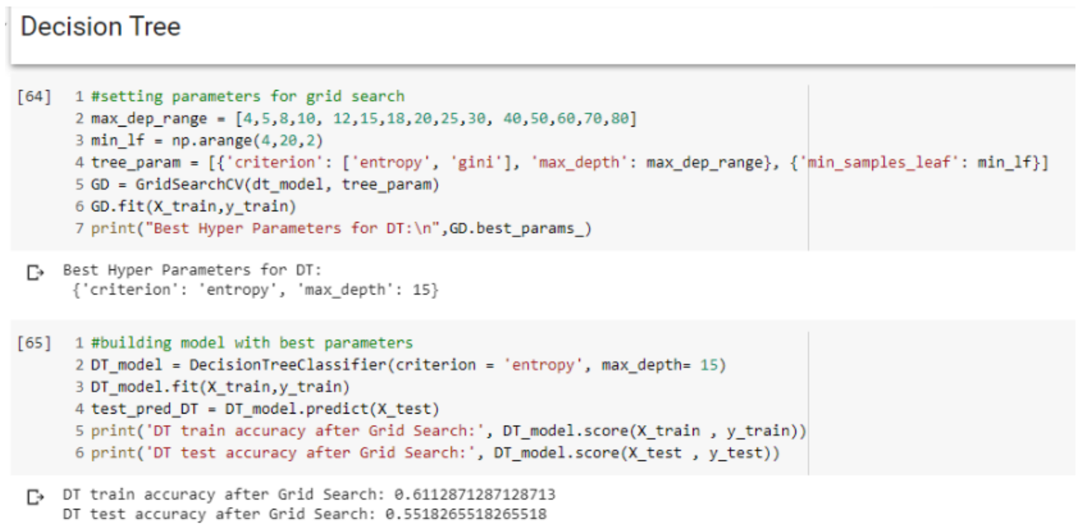
```
model_svm.fit(X_train, y_train)
print(model_svm.best_score_)
print(model_svm.best_params_)
```

```
0.4322447257383966
{'C': 0.1, 'degree': 2, 'gamma': 0.01, 'kernel': 'poly'}
```

Figure 41: Grid Search - SVM

RESULT:
The best param found is C-0.1, degree-2, gamma-0.01, kernel-poly. The best accuracy is 43

7.3 Decision Tree



Decision Tree

```
[64]
#setting parameters for grid search
max_dep_range = [4,5,8,10, 12,15,18,20,25,30, 40,50,60,70,80]
min_lf = np.arange(4,20,2)
tree_param = [{'criterion': ['entropy', 'gini'], 'max_depth': max_dep_range}, {'min_samples_leaf': min_lf}]
GD = GridSearchCV(dt_model, tree_param)
GD.fit(X_train,y_train)
print("Best Hyper Parameters for DT:\n",GD.best_params_)
```

```
Best Hyper Parameters for DT:
 {'criterion': 'entropy', 'max_depth': 15}
```

```
[65]
#building model with best parameters
DT_model = DecisionTreeClassifier(criterion = 'entropy', max_depth= 15)
DT_model.fit(X_train,y_train)
test_pred_DT = DT_model.predict(X_test)
print('DT train accuracy after Grid Search:', DT_model.score(X_train , y_train))
print('DT test accuracy after Grid Search:', DT_model.score(X_test , y_test))
```

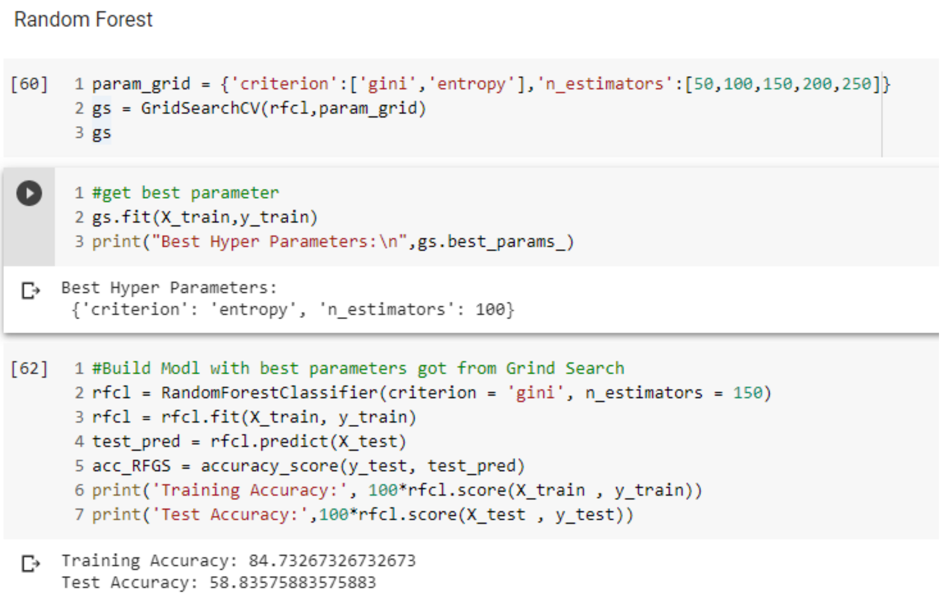
```
DT train accuracy after Grid Search: 0.6112871287128713
DT test accuracy after Grid Search: 0.5518265518265518
```

Figure 42: Decision Tree – Grid Search

Result:
The training accuracy is 61% and testing accuracy is 55% after running the model with Grid search's best parameters. The train accuracy has come down while the test accuracy is slightly more than the initial model. The model seems to be able to generalize better but the test accuracy is still not good enough

7.4 Random Forest

Using Grid Search for Random Forest model, we see that the best hyperparameters for the model is Entropy and 100 estimators. Applying these parameters and running the model gives us test accuracy of 58% but the accuracy is not a great improvement from the initial model accuracy.



Random Forest

```
[60] param_grid = {'criterion':['gini','entropy'],'n_estimators':[50,100,150,200,250]}
     gs = GridSearchCV(rfcl,param_grid)
     gs
```

```
#get best parameter
gs.fit(X_train,y_train)
print("Best Hyper Parameters:\n",gs.best_params_)
```

```
Best Hyper Parameters:
 {'criterion': 'entropy', 'n_estimators': 100}
```

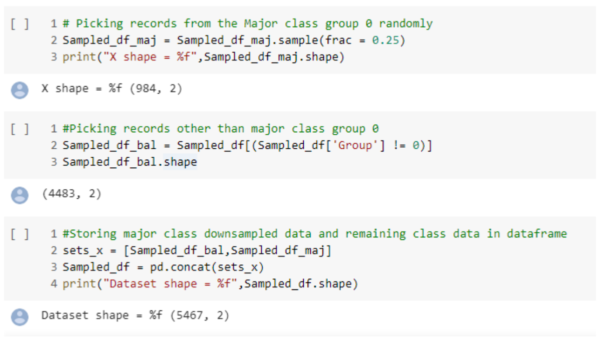
```
[62] #Build Modl with best parameters got from Grind Search
     rfcl = RandomForestClassifier(criterion = 'gini', n_estimators = 150)
     rfcl = rfcl.fit(X_train, y_train)
     test_pred = rfcl.predict(X_test)
     acc_RFGS = accuracy_score(y_test, test_pred)
     print('Training Accuracy:', 100*rfcl.score(X_train , y_train))
     print('Test Accuracy:',100*rfcl.score(X_test , y_test))
```

```
Training Accuracy: 84.73267326732673
Test Accuracy: 58.83575883575883
```

Figure 43: Random Forest – Grid Search

8 Deep Learning Networks

In the basic models built so far the maximum accuracy we could reach was 58% with hyper tuning Random Forest model. We can now check to see if we can get better result with some deep neural network models. Random OverSampling
In this project, we have tried oversampling technique as one of the way to address class imbalance. We have upsampled the minority class and down sampled the majority class to get effective representation of each class.



```
[ ] # Picking records from the Major class group 0 randomly
    Sampled_df_maj = Sampled_df_maj.sample(frac = 0.25)
    print("X shape = %f",Sampled_df_maj.shape)
```

```
X shape = %f (984, 2)
```

```
[ ] #Picking records other than major class group 0
    Sampled_df_bal = Sampled_df[(Sampled_df['Group'] != 0)]
    Sampled_df_bal.shape
```

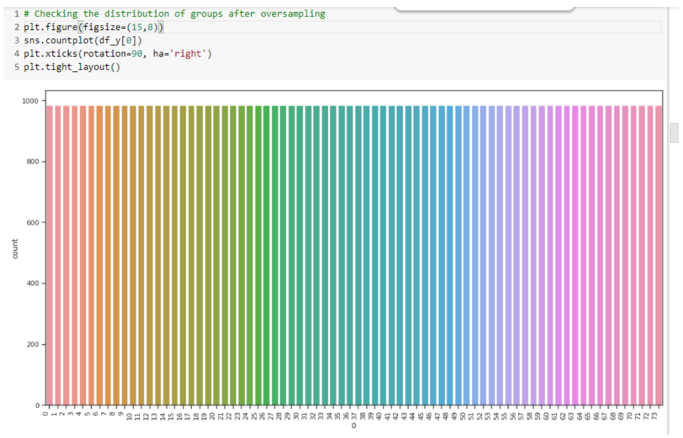
```
(4483, 2)
```

```
[ ] #Storing major class downsampled data and remaining class data in dataframe
    sets_x = [Sampled_df_bal,Sampled_df_maj]
    Sampled_df = pd.concat(sets_x)
    print("Dataset shape = %f",Sampled_df.shape)
```

```
Dataset shape = %f (5467, 2)
```

Figure 44: Sampling techniques implemented

After oversampling the minority class and undersampling the majority class, the size of features (X) is 72816. The distribution of groups after sampling is shown below.



```
# Checking the distribution of groups after oversampling
plt.figure(figsize=(15,8))
sns.countplot(df_y[0])
plt.xticks(rotation=90, ha='right')
plt.tight_layout()
```

Figure 45: Group distribution after sampling

8.1 LSTM

Long short-term memory (LSTM) is an artificial recurrent neural network (RNN) architecture. Recurrent Neural Networks (RNN) has an internal state that store information of past inputs for a certain time by which the model learns the context of the information. Where the inputs of the model are sequential in nature, the model can return the output as sequence with the contextual information.

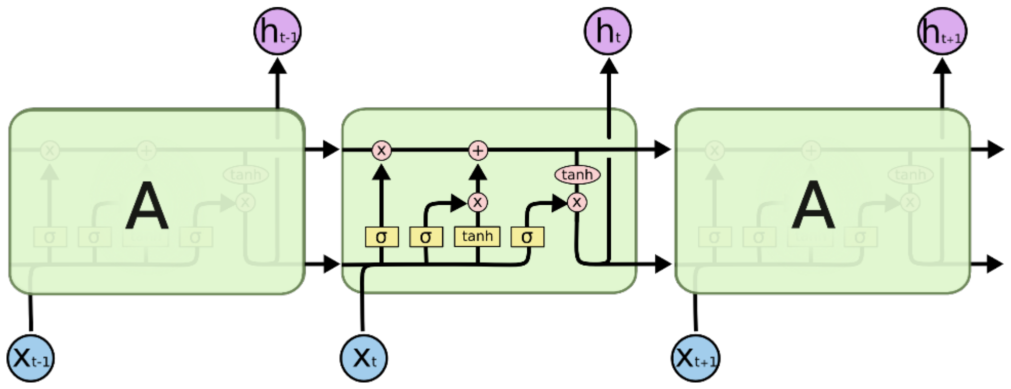
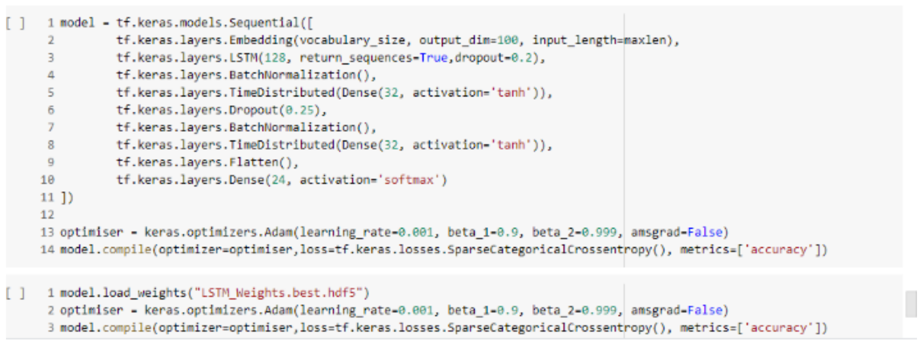


Figure 46: LSTM cell Archtechture(https://colah.github.io/posts/)

LSTM networks have some internal contextual state cells that act as long-term or short-term memory cells. The output of the LSTM network is modulated by the state of these cells. This is a very important property when we need the prediction of the neural network to depend on the historical context of inputs, rather than only on the very last input. In our project, we have not been able to reach the expected accuracy with conventional ML models as SVM, NB, Ensembles, we decided to use LSTM to solve this Automated Ticket Classification problem. We use the bidirectional LSTM which presents each training sequence forwards and backwards to two separate recurrent nets, both of which are connected to the same output layer. This means that for every point in a sequence, the model has complete information.



```
[ ]
model = tf.keras.models.Sequential([
        tf.keras.layers.Embedding(vocabulary_size, output_dim=100, input_length=maxlen),
        tf.keras.layers.LSTM(128, return_sequences=True,dropout=0.2),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.TimeDistributed(Dense(32, activation='tanh')),
        tf.keras.layers.Dropout(0.25),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.TimeDistributed(Dense(32, activation='tanh')),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(24, activation='softmax')
])

optimiser = keras.optimizers.Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999, amsgrad=False)
model.compile(optimizer=optimiser,loss=tf.keras.losses.SparseCategoricalCrossentropy(), metrics=['accuracy'])
```

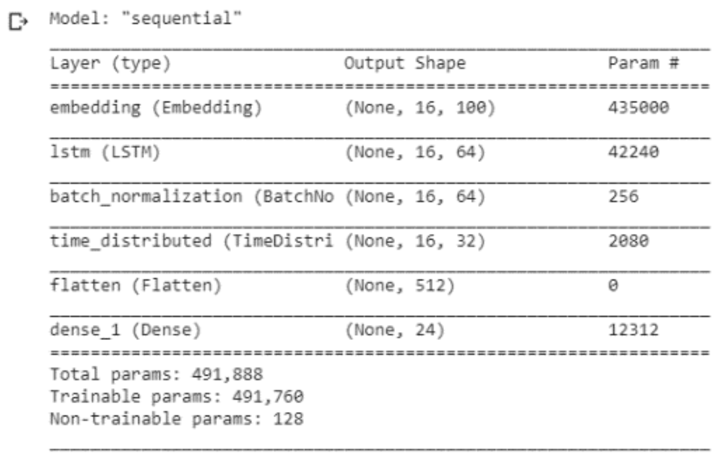
```
[ ]
model.load_weights("LSTM_Weights.best.hdf5")
optimiser = keras.optimizers.Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999, amsgrad=False)
model.compile(optimizer=optimiser,loss=tf.keras.losses.SparseCategoricalCrossentropy(), metrics=['accuracy'])
```

Figure 47: LSTM Model

There were total 74 categories in the data but the data was imbalanced across different categories with some categories having as high as 3900 records while around 49 categories contained less than 50 records. Thus a threshold for Automation criteria was setup. All categories with records less than 50 records were merged into single category and this category was not considered for automation or model training. The tickets in these categories should be considered for manual triaging. 24 categories with ticket count more than or equal to 50 were used for LSTM model training. The architecture of our LSTM model is shown as below.
The architecture of our LSTM model is shown as below.
Model was trained with following setting of optimizers.

- SGD
- RMSP
- Adam
- Nadam
- Adagrad

And different learning_rates [0.001, 0.01,0.1]. The model was trained (with validation split of 0.2) for varying values of batch [100,200,250,500]. The model weights for each execution was saved and was reloaded for next execution for respective optimizer/learning rates. The total parameters in this model is 491,888 with 491,760 trainable parameters. Out of the optimizers tried, Adam with



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 16, 100)           435000
_________________________________________________________________
lstm (LSTM)                  (None, 16, 64)            42240
_________________________________________________________________
batch_normalization (BatchNo (None, 16, 64)            256
_________________________________________________________________
time_distributed (TimeDistri (None, 16, 32)            2080
_________________________________________________________________
flatten (Flatten)            (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 24)                12312
=================================================================
Total params: 491,888
Trainable params: 491,760
Non-trainable params: 128
_________________________________________________________________
```

Figure 48: LSTM Model Summary

learning rate 0.001 and batch of 200 have been able to attain a Training Accuracy of 90% and Test Accuracy of 88We executed LSTM model with two options. Option 1: all 74 categories and option2: 24 classes, combining the minority classes into one category. LSTM with 74 category gives 90% training accuracy and 89% test accuracy. LSTM with 24 category gives training accuracy of 90.6%, testing accuracy of 88.1%, validation accuracy of 87.8%, and AUC 94%.

8.2 DNN

Deep Neural Networks architectures are designed to learn through multiple connection of layers where each single layer only receives connection from previous and provides connections only to the next layer in hidden part.

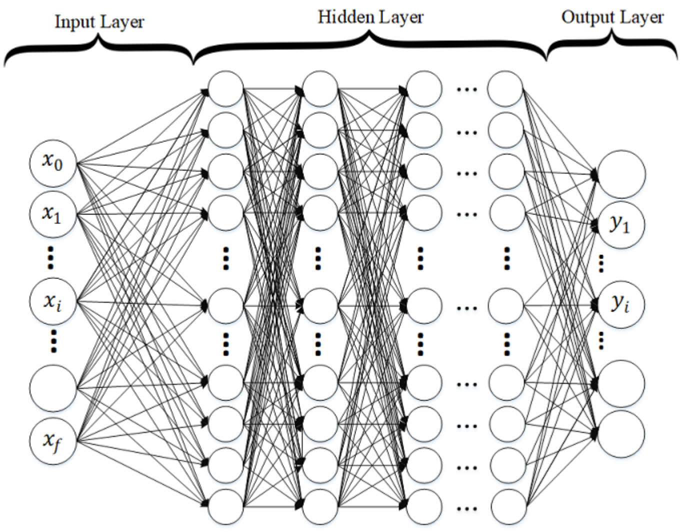
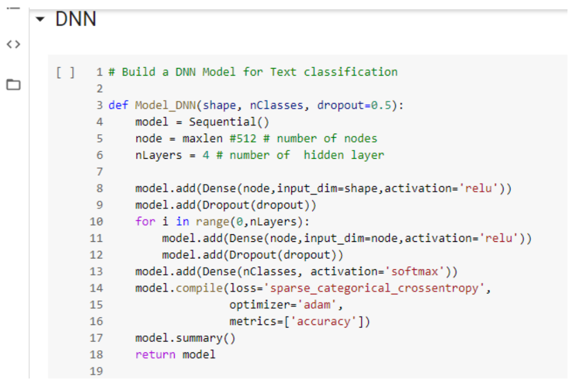


Figure 49: DNN Architecture

For Deep Neural Networks (DNN), input layer could be tf-ifd, word embedding, or etc. The output layer houses neurons equal to the number of classes for multi-class classification and only one neuron for binary classification.

Our main contribution in this project is that we have many trained different architectures of DNNs to evaluate the impact on the performance. Here, we have multi-class DNNs where each learning model is generated randomly (number of nodes in each layer as well as the number of layers are randomly assigned). Our implementation of Deep Neural Network (DNN) is basically a discriminatively trained model that uses standard back-propagation algorithm and sigmoid or ReLU as activation functions. The output layer for multi-class classification should use Softmax.



```
# Build a DNN Model for Text classification

def Model_DNN(shape, nClasses, dropout=0.5):
    model = Sequential()
    node = maxlen #512 # number of nodes
    nLayers = 4 # number of  hidden layer

    model.add(Dense(node,input_dim=shape,activation='relu'))
    model.add(Dropout(dropout))
    for i in range(0,nLayers):
        model.add(Dense(node,input_dim=node,activation='relu'))
        model.add(Dropout(dropout))
    model.add(Dense(nClasses, activation='softmax'))
    model.compile(loss='sparse_categorical_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy'])
    model.summary()
    return model

```

Figure 50: DNN Model

Besides the architectural tuning to find the optimum DNN, the epochs and batch size were in-

DNN	**DNN Architecture**	**Accuracy %**	
		Training	Test
DNN1	The model has the same number of nodes = 367 in all the 4 layers of the NN. Dropout =0.5.	85.30	84.70
DNN2	The model has the same architecture as DNN1 however the hidden nodes = 367*2.	85.66	84.74
DNN3	The model has the same number of nodes = 367 in all the 8 layers of the NN. Dropout =0.5.	31.54	30.75
DNN4	Analyzing the DNN with input same as DNN2 but with 8 layers.	71.89	70.72
DNN5	Analyzing the DNN with input same as DNN1 but with Dropout =0.2.	86.12	84.95
DNN6	Analyzing the DNN with input same as DNN1 but with without Dropout.	85.76	84.63
DNN7(NN7)	Simple NN layers- 1 input, 1 output and 1 hidden layer and no dropout.	86.35	85.43
DNN8	Activation function is replaced by sigmoid instead of relu for the hidden layers	82.55	81.27
DNN9	Analyzing the DNN with input same as DNN1 but with reduction in the hidden nodes by half the input nodes (367).	86.19	85.03

creased however there was marginal improvement in the model accuracy. An epoch size of 20 and batch size between 100-150 delivered the same results as when the epoch was increased between 100-300. The DNN 5 and DNN 9 architecture gives maximum accuracy. The AUC for DNN5 and DNN9 is computed to be 92%.

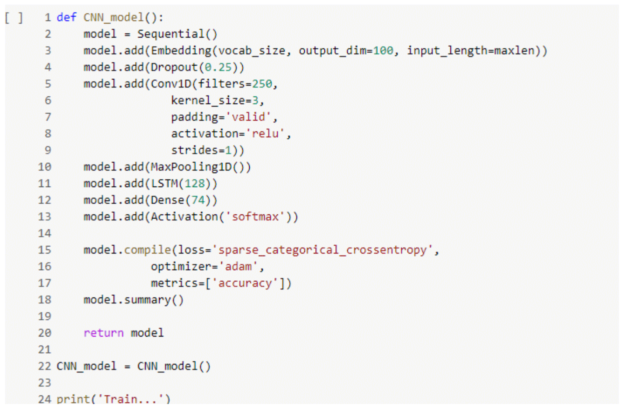
8.3 RCNN

The Recurrent Neural Network (RecurrentNN) analyzes a text word by word and stores the semantics of all the previous text in a fixed-sized hidden layer. The advantage of RecurrentNN is the ability to better capture the contextual information. This could be beneficial to capture semantics of long texts. However, the RNN is a biased model, where later words are more dominant than earlier words. Thus, it could reduce the effectiveness when it is used to capture the semantics of a whole document, because key components could appear anywhere in a document rather than at the end.

To tackle the bias problem, the Convolutional Neural Network (CNN), an unbiased model is introduced to NLP tasks, which can fairly determine discriminative phrases in a text with a max-pooling layer. Thus, the CNN may better capture the semantic of texts compared to recursive or recurrent neural networks. The time complexity of the CNN is also O(n). However, previous studies on CNNs tends to use simple convolutional kernels such as a fixed window.

When using such kernels, it is difficult to determine the window size: small window sizes may result in the loss of some critical information, whereas large windows result in an enormous parameter space (which could be difficult to train).

To address the limitation of the above models, we suggest a combination of CNN and RNN to form a Recurrent Convolutional Neural Network (RCNN) and apply it to the task of text classification.



```python
def CNN_model():
    model = Sequential()
    model.add(Embedding(vocab_size, output_dim=100, input_length=maxlen))
    model.add(Dropout(0.25))
    model.add(Conv1D(filters=250,
                     kernel_size=3,
                     padding='valid',
                     activation='relu',
                     strides=1))
    model.add(MaxPooling1D())
    model.add(LSTM(128))
    model.add(Dense(74))
    model.add(Activation('softmax'))

    model.compile(loss='sparse_categorical_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy'])
    model.summary()

    return model

CNN_model = CNN_model()

print('Train...')
```

Figure 51: RCNN Model

First, we apply a CNN architecture of Conv1D for text classification and Maxpooling and there by feed it to the LSTM architecture which connects to a dense network to classify the output group probabilities.

Pros

- Tackles bias problems of RNN•
- Better captures the contextual information
- Combination model that uses the strengths of both RNN and CNN combined.

Cons

- Training intensive.

The test accuracy for RCNN is 87%, and AUC is 93%.

9 Model Comparison and Evaluation

We have re-run all the models after sampling techniques for Milestone2. The traditional models too have performed well after oversampling the data. Tabulating all the models with their accuracies below.

Model	**Train**	**Test**	**Precision**	**Recall**	**F1 score**
Naive Bayes(Multinomial)	67.48	59.16	49.20	59.16	68.25
SVC(kernel='linear')	80.81	59.9	53.34	59.9	65.26
Decision Tree	88.47	51.91	46.52	51.91	84.1
Random Forest	88.45	58.65	50.23	58.65	67.47
Ensemble(Bagging)	-	60	-	-	-
Ensemble(Boosting)	-	60	-	-	-
Naive Bayes(Multinomial) (hyperparameter)	67.48	59.13	49.10	59.13	68.25
SVM(hyperparameter)	73.62	59.04	54.87	59.04	69.95
Decision Tree(hyperparameter)	62.45	55.44	44.00	55.44	66.55
Random Forest(hyperparameter)	88.47	59.96	51.41	59.96	67.74
DNN	85.16	83.96	91.75	83.96	85.55
DNN2	85.32	84.44	92.00	84.44	85.78
DNN3	37.95	37.61	35.42	37.61	32.63
DNN4	64.79	64.05	65.87	64.05	60.49
DNN5	86.14	85.04	92.95	85.04	86.50
DNN6	86.17	85.01	93.00	85.01	86.59
NN	86.09	84.97	93.26	84.97	86.47
DNN8	83.00	81.76	89.16	81.76	82.90
DNN9	86.05	84.82	92.99	84.82	86.30
LSTM	89.98	88.85	95.72	88.85	90.01
RCNN	88.54	87.36	95.62	87.36	88.76

Evaluation

We have evaluated the models based on the test and validation accuracy reached. For the given problem statement, we have tried six traditional models, three deep learning models and recorded the train and test accuracy. We have fine tuned each model by finding optimal parameter to increase the accuracy. We also applied sampling techniques to aid the model learning and get better accuracy. Out of all the models tried, LSTM has given the maximum accuracy of 88%.

Benchmark Comparison

For this problem statement, we had set a benchmark to achieve test and validation accuracy above 90%. After trying traditional models and then moving to Deep Neural Networks, we have tried various architectures, optimizers and learning rates to get 88% as a stabilized result with LSTM. Further training the model or tweaking lead us to model overfitting. We have managed to attain the accuracy of 88% with LSTM.

Implications and Limitations

Since the business domain of this problem is IT solutions, 88% accuracy can be considered fair. The model is robust to be deployed in production with the accuracy attained; however, the model can be further improved if:

- Non-English text is translated and included for model learning
- Considerable data available pertaining to the minority classes for better learning (We have upsampled the minority classes for this project.)

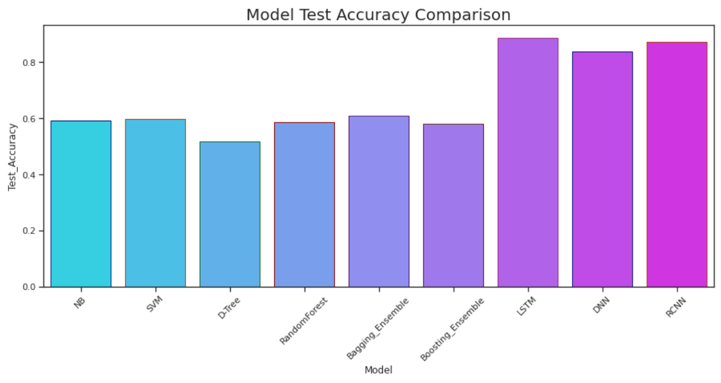
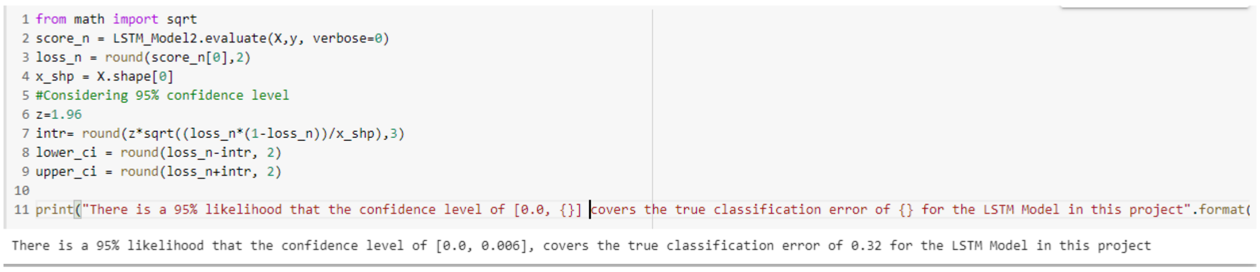


Figure 52: Model accuracy comparison



```python
from math import sqrt
score_n = LSTM_Model2.evaluate(X,y, verbose=0)
loss_n = round(score_n[0],2)
x_shp = X.shape[0]
#Considering 95% confidence level
z=1.96
intr= round(z*sqrt((loss_n*(1-loss_n))/x_shp),3)
lower_ci = round(loss_n-intr, 2)
upper_ci = round(loss_n+intr, 2)

print("There is a 95% likelihood that the confidence level of [0.0, {}] covers the true classification error of {} for the LSTM Model in this project".format(
```

```
There is a 95% likelihood that the confidence level of [0.0, 0.006], covers the true classification error of 0.32 for the LSTM Model in this project
```

Figure 53: Confidence Intervel for LSTM

10 Summary and Conclusion

Summary

- In this project we loaded the data from https://drive.google.com/drive/u/0/folders/1xOCdNI2R5hiodskIJbj-QySMQs6ccehL
- Analysed each feature in the given dataset
- Found null values and imputed them, found duplicated values and removed them.
- Cleaned the text in each feature for unwanted characters
- Concatenated all the features, created weighted vectors and split them into train and test
- Built basic models such as Naïve Bayes, SVC, Decision Tree, Random Forest, and Ensemble
- Tuned the model to get optimal accuracy by using Grid Search
- Applied random sampling techniques to address class imbalance
- Built DNN and LSTM model

Conclusion

- The average accuracy with basic model is around:
- With over sampling and stratified Kfold, the average accuracy is around:
- DNN accuracy is 85
- LSTM accuracy is 88
- For this dataset, LSTM model gives the best accuracy.

The confidence level of 95% is computed for the model as below.

Closing Reflection and Insights

- This project considers only the English text for model, which can be improved if non-English texts are considered after translation
- Applying NMT was considered; however, within given timeframe we were unable to incorporate this.
- The dataset was highly imbalanced and we have applied sampling techniques to address the issue. In the production scenario, the accuracy could improve if we could get more data especially for minority class.

Different Approaches for Next Time Upon brainstorming, we have discussed few other approaches that we could try next time.

- Dropping the minority classes where the data is as less as one ticket
- Selecting fixed number of samples from each class in such a way that the sample size has equal representation of each class
- Adding/ Leaving out the caller feature to the model building and see the predictions and difference
- Adding NMT and translate all the data to English before processing
- Create an API and expose parameters such as exclude caller, frequency threshold etc so that the IT Business admin can set them manually

11 Challenges, Approach and Mitigation

Challenges	Approach	Mitigation
1. Hardware		
Personal machines used for the project has 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 is the most importance piece of the puzzle with regards to Machine Learning(ML) problems. Our observations are as follows:		
2. Class Imbalance		
We can observe that the number of observations in each group is poorly distributed. There are totally 74 groups with some groups having as less as one observation.	Use sampling techniques would enable us to down sample the majority classes or/and upsample the minority classes. Group the minority classes into one group for classification	We created a group clubbing the minority classes into one group. Groups having less than 50 tickets will be categorized into one group. We also used upsampling with stratification and downsampling of majority class for Neural Network model. This increased the accuracy and precision.
3. 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.
4. 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.

Challenges	Approach	Mitigation
5. 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.

12 Code and Deliverable

- Final Report_Group 3 NLP.tex
- PDF format - Final Report_Group 3 NLP.pdf
- Filenames listed, attachments in the Great Learning (GL) portal
- NLP_Automated_Ticket_Assignment_Final.ipynb
- NLP_Automated_Ticket_Assignment_Final.html
- Snapshot of Github Repo (Capstone)-
- https://github.com/GLNLPGroup3/Capstone

13 Reference 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-pythonadvantages

17. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

18. Siwei Lai, Liheng Xu, Kang Liu, Jun Zhao, -Recurrent Convolutional Neural Networks for Text Classification

19. National Laboratory of Pattern Recognition (NLPR), Institute of Automation, Chinese Academy of Sciences, China, fswlai, lhxu, kliu, jzhaog@nlpr.ia.ac.cn

End of Report