AIML Capstone Project: Automatic Ticket Assignment(NLP)

Nov2019 B-batch, Group 5

28 NOVEMBER 2020

Contents

Chapter 1

Introduction

Credits

This Capstone project report provides a comprehensive detail on the project thus worked upon by "November2019 B-batch, Group 5" on the topic of Natural Language processing. The project is based on the issue of ticket assignment in customer support.

The group that has worked on the project consists of 6 people:

1. Anugrah George James
2. Shivang Pathak
3. Milind Mukesh
4. Naga Aruna Sreshta Appadwedula
5. Mona Datta
6. Shikhar Vashishtha

The group was mentored by Saandeep Sreerambatla and we thank him for his insights and guidance for providing direction to this project.

Finally, we would like to thank the team of Great Learning for providing us a wonderful opportunity to learn and grow.

1.1 Summary of problem statement, data and findings

1.1.1 Problem Statement

One of the key activities of any IT function is to "Keep the lights on" to ensure there is no impact to the Business operations. IT leverages Incident Management process to achieve the above Objective. An incident is something that is 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 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. In most of the organizations, incidents are created by various Business and IT Users, End Users/ Vendors if they have access to ticketing systems, and from the integrated monitoring systems and tools. Assigning the incidents to the appropriate person or unit in the support team has critical importance to provide improved user satisfaction while ensuring better allocation of support resources. 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.

1.1.2 Data and Findings

The data provided for the project consisted of support tickets consolidated in and excel file, which consisted of 4 columns viz "Short Description, Description, Caller and Assignment Group". The records in the data were in multiple languages other than English, (Ex: German and French) and also held ascii values.

Apart from this there were null values within the data for some records, that require to be cleaned up.

1.1.3 Implications

As per the data we found, first we have done data pre-preprocessing, that would include removal of unnecessary columns, replacing null values, perform some data visualization to understand how the data is distributed and its attributes, data translation, etc. and then working on NLP methodologies. Furthermore, we have worked with traditional ML models and Deep Learning techniques to find out which is the best model that can be used for this data, and further tuning of the model would be required to increase its efficiency and accuracy. The results of this model will help us assigning the support tickets in better fashion, thus improving cost and time savings. This will also result in better customer feedback and timely resolution of the tickets.

1.2 Overview of the final process

1.2.1 Problem Methodology

To solve the problem, we will divide the project in 3 stages that are:

1. Pre-Processing, Data Visualization and EDA
2. Model Building
3. Test the model, Fine Tuning and repeat.

1.2.2 Data Pre-Processing, Data Features and EDA

As explained previously, that the data consists of lot of inconsistencies like missing values, unnecessary data columns, multiple languages, etc. We would need to remove these inconsistencies by performing data pre-processing. For this we have done the below:

1. Explore the given Data files. Here we first loaded the data using panda libraries into a dataframe and saw how the data looked as a tabular format.

2. Understanding the structure of data. Using normal python methods, we found out that the data consisted around 8500 records distributed in 4 columns that are "Short Description, Description, Caller and Assignment Group". We then tried to understand what data types did the data had along with statistical analysis. Thus, we came to following conclusion:

   (a) We observe that all columns don't have 8500 non null values, means there are null vales in data that we have to handle. We can observe that the data is highly imbalanced & skewed.

   (b) Total records: 8500 & Total Attributes: 4

   (c) Since our goal is automatic ticket assignment, It doesn't depend on the caller. Also, a caller can raise tickets for any issue he or she is facing. Therefore, we can ignore the caller attribute/feature for further detail analysis.

   (d) There are 8 records with null value in short description and 1 record with null value in description.

   (e) One user/caller has raised 810 tickets.

   (f) There are 74 unique groups and the group GRP_0 has been assigned 3976 tickets. 'GRP_0' has maximum instances around ∼40%

   (g) Top description is just the word 'the' which also we have to take care of.

   (h) Short description & description count doesn't match with the total no of callers or assigned groups.

   (i) Password Reset is one of the most occurring ticket topics.

3. Finding inconsistencies in the data. We found out that the caller column was the one which had no use howsoever. Thus, we removed that column to prevent any inconsistencies.

4. Null Value treatment We found out that the data consisted of some null values which needed replacement. Following was the process:

   (a) Drop the missing values: The number of complete cases i.e. observation with no missing data must be sufficient for the selected analysis technique then we can see this. However, in our case, we already have quite skewed data and a long tail. So, ignoring null rows can be fatal in totally missing out some groups.

   (b) Imputation: If the missing values in a column or feature are numerical, the values can be imputed by the mean or median of the complete cases of the variable. But in our case, we have character strings.

   (c) Conjoining/ custom replacement: The cases where we can find another custom solution to give a logical value to null. In our case fortunately the we observed that wherever we have NaN in any description we have proper input in short column and vice versa. Hence, we can make a readable combined description and make NaN replaceable and combine short and description as one column.

5. Visualizing different patterns. For this, we used the wordcloud library to visualize different words that are occurring most frequently within the data. A wordcloud is an image composed of words used in a particular text or subject, in which the size of each word indicates its frequency or importance. Below is an example of a wordcloud from a specific ticket group.

6. Visualizing different text features. For doing this, we first encoded the data or precisely, the text within the data. Text encoding transforms words into numbers and texts into number vectors. And in the given data set we also found that there are a lot of entries/records that are in multiple languages. Thus, we need them to translate as well to one language, that is easily understandable. Here we chose English. Also, there are some special characters in some records, so we translated them as well. For

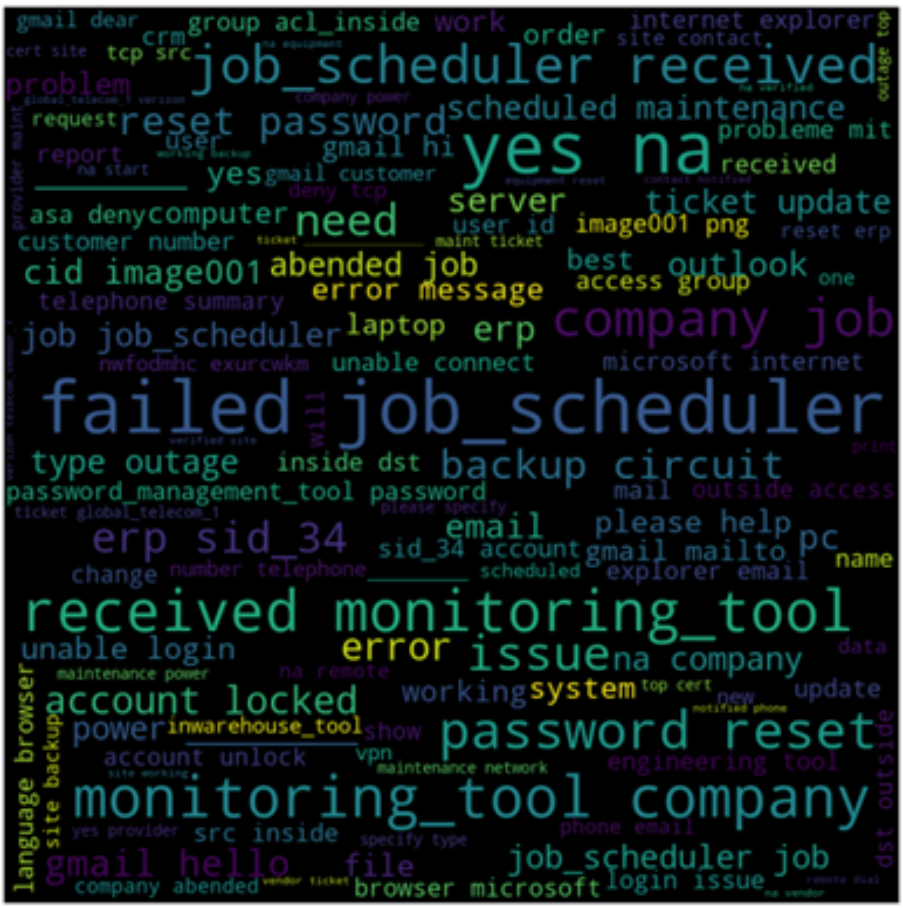

Figure 1.1: World Cloud Example

this, we first tried to detect the different languages present in the data set by using langdetect library. This library helped us to detect the various languages present in our data and gave us a comprehensive view on it. After detecting the given languages, it was time to use the "googletrans" library by Google, to translate all these languages into English.

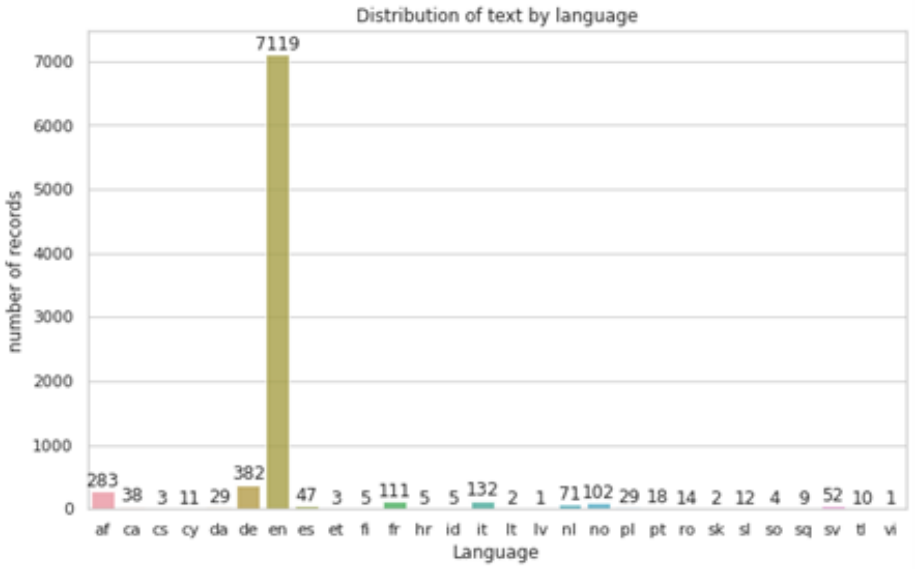

Figure 1.2: Distribution of Detected Languages

7. Text Pre-Processing. Text Pre-processing includes several parts of working with data. These are:

   (a) Lower casing: Converting the words into lower case format. "(NLU -> nlu)". Words having the same meaning like nlp and NLP if they are not converted into lowercase then these both will constitute as non-identical words in the vector space model.

   (b) Stop words removal: These are the most often used that do not have any significance while determining the two different documents like "(a, an, the, etc.)" so they are to be removed.

   (c) Contextual conversational words removal: In our case, words like 'received from', 'to', 'regards', 'subject', 'email address', which are

identified as words used in an standard email conversations to be removed.

(d) Punctuation: The text has several punctuations. Punctuations are often unnecessary as it doesn't add value or meaning to the NLP model.

(e) Other steps: Other cleaning steps can be performed based on the data. Listed a few of them below, Remove URLs, Remove HTML tags, remove numbers, remove hashtags etc. are also used here.

8. Creating word vocabulary from the corpus of report text data. For this we worked the following ways:

(a) **Deriving N-grams**: N-grams of texts are extensively used in text mining and natural language processing tasks. They are basically a set of co-occurring words within a given window. N-grams is a contiguous sequence of N items from a given sample of text or speech, in the fields of computational linguistics and probability. The items can be phonemes, syllables, letters, words or base pairs according to the application. N-grams are used to describe the number of words used as observation points, e.g., unigram means singly-worded, bigram means 2-worded phrase, and trigram means 3-worded phrase. We'll be using scikit-learn's CountVectorizer function to derive n-grams We will write a generic method to derive the n-grams.

(b) **Lemmatization**: Lemmatization is the process of process of reducing a word to its root form by grouping together the different inflected forms of a word so they can be analyzed as a single item. It helps to reduce variations of the same word, thereby reducing the corpus of words to be included in the model. So, it returns the base or dictionary form of a word, which is known as the lemma. It is important when clean the data of all words of a given root. Lemmatizing considers the context of the word and shortens the word into its root

form based on the dictionary definition.

9. Creating tokens as required. Tokenization is a process that splits an input sequence into so-called tokens where the tokens can be a word, sentence, paragraph etc. We Tokenized the given data by using NLTK library and from TensorFlow's tokenizer.

   The last work to be done to organize our corpus data was by topic modelling. Topic modeling provides methods for automatically organizing, understanding, searching, and summarizing large electronic archives.

   Here for topic modeling, we are using Genism.

   Gensim("Generate Similar") is a popular open source natural language processing(NLP) library used for unsupervised topic modeling. It uses top academic models and modern statistical machine learning to perform various complex tasks such as:

   (a) Building document or word vectors

   (b) Corpora

   (c) Performing topic identification

   (d) Performing document comparison (retrieving semantically similar documents)

   (e) Analyzing plain-text documents for semantic structure

   Lastly, we used LDA(Latent Dirichlet Allocation). The aim of LDA is to find topics a document belongs to, based on the words in it. Points 7-9 are required to prepare the data for working with NLP algorithms. These are the basic and fundamental activities that need to be performed while working with NLP. Finally we have introduced pyLDavis for visualizing this data. pyLDAvis is designed to help users interpret the topics in a topic model that has been fit to a corpus of text data. The package extracts information from a fitted LDA topic model to inform an interactive web-based visualization.

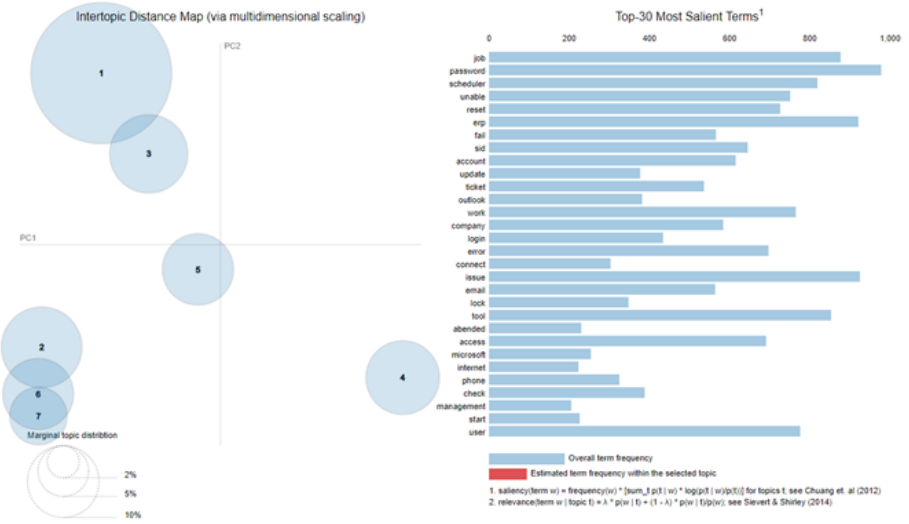

Figure 1.3: pyLDAvis LDA Topic Model

1.2.3 Model Building

The model building part of the project will involve building an NLP model where we:

1. Build a model architecture which can classify.
2. Try different model architectures by researching state of the art for similar tasks.
3. Train the model.
4. To deal with large training time, save the weights so that you can use them when training the model for the second time without starting from scratch.

We've used various algorithms to build our model, to find the best suited model. This was done first without resampling the data at an initial level. The algorithms used were:

1. K Nearest Neighbor
2. Support Vector Machines

3. Decision Trees

4. Random Forest Classifier

5. Logistic Regression

6. ADA Boost Classifier

7. XG boost Classifier

At last, all of these models were compared to check which of them were performing better. It was found that there is over-fitting and data imbalance in the dataset.

1.2.4 Model Improvement - Test the model, Fine Tuning and Repeat

Model creation is not the only part that needs to be done for providing the final model for the project. We would need to also fine tune the model and test it comparing it to other Deep Learning methodologies to arrive at the final conclusion. This would include the following steps:

1. Test the model and report as per evaluation metrics.

2. Try different models.

3. Try different evaluation metrics.

4. Set different hyper parameters, by trying different. optimizers, loss functions, epochs, learning rate, batch size, check pointing, early stopping etc. for these models to fine-tune them.

5. Report evaluation metrics for these models along with your observation on how changing different hyper parameters leads to change in the final evaluation metric.

For this we worked with various model improvement techniques and worked with Deep Learning techniques with and without sampling. The Model improvement was further divided in 3 parts:

1. Utilizing deep learning techniques - without sampling
   (a) Recurring Neural Networks - LSTM
   (b) Recurring Neural Networks - GRU
   (c) Bidirectional LSTM
2. Utilization of Resampled training data set. Here we tested the above non deep learning and deep learning models with resampled data and checked their performance.
3. Utilization of grouping and fastext. Here we grouped the data based on ticket count and then used fastext for faster and better performing text classification.

Chapter 2

Modeling, Evaluation and Comparison

2.1 Step By Step Walk Through The Solution

- Traditional machine learning algorithms meant for classification problem solving will be tried against the vectorized features generated out of TF-IDF.
- Comparison of the model accuracy for selecting best performing model.
- Analyse & check for possible improvements, if required.

Machine Learning involves the extensive use of mathematical models to predict the outputs for classification problems. So, different models are compiled on the data and evaluated based on performance metrics. The traditional models considered are:

1. Multinomial Naive Bayes
2. K Nearest neighbour (KNN)
3. Support Vector Machine

4. Decision Tree

5. Random Forest

6. Logistic Regression

7. Ada Boost Classifier

8. Bagging Classifier

9. XG Boost Classifier

After the data has been cleaned and pre-processed, training and testing data sets were created by splitting the cleaned data set in the ratio 80:20 and label encoding is done. In machine learning, we usually deal with datasets which contains multiple labels in one or more than one column. These labels can be in the form of words or numbers. To make the data understandable or in human readable form, the training data is often labeled in words. **Label Encoding** refers to converting the labels into numeric form to convert it into the machine-readable form.

Now, the data is ready to be fitted on different traditional machine learning models mentioned above. The f1 score has been proven as a good performance metric for text classification problems. So, the models, though have been evaluated on different performance metrics, were concluded upon based on f1 score and was the highest for Linear SVM model.

⇒ Observation: Here in this comparison of different traditional ML models, we can observe a substantial difference in the accuracy of training and test sets. Major reason is possibly the imbalanced data distribution of the dataset used and the inability of the model to learn and adapt during training.

We need to check if these issues can be handled by resampled data and using deep learning techniques.

2.1.1 Model Improvement 1: Utilizing Deep Learning Techniques

Deep learning is a subset of machine learning where artificial neural networks, algorithms inspired by the human brain, learn from large amounts of data. Similarly, to how we learn from experience, the deep learning algorithm would perform a task repeatedly, each time tweaking it a little to improve the outcome. We refer to deep learning because the neural networks have various (deep) layers that enable learning.

Deep Learning Models considered are:

- Recurrent Neural Network (RNN)-LSTM Model:
- Recurrent Neural Network (RNN)-GRU Model
- Bidirectional LSTM Model

All the necessary libraries were imported and embedding matrix was created.

- Instead of learning word embeddings jointly with the problem we want to solve, we prefer loading embedding vectors from a pre-computed embedding space known to be highly structured and to exhibit useful properties that captures generic aspects of language structure.
- Such word embeddings are generally computed using word occurrence statistics (observations about what words co-occur in sentences or documents), using a variety of techniques, some involving neural networks, others not. In our case, we used GloVe.
- GloVe- developed by Stanford researchers in 2014. It stands for "Global Vectors for Word Representation", and it is an embedding technique based on factorizing a matrix of word co-occurrence statistics. The GloVe model is trained on the non-zero entries of a global word-word co-occurrence matrix, which tabulates how frequently words co-occur with one another

in each given corpus. Populating this matrix requires a single pass through the entire corpus to collect the statistics.

- GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.

- We used glove.6B.200d.txt which is Glove embeddings with 6 Billion words with each word being a 200-dimensional vector.

1. **Recurrent Neural Network (RNN) - LSTM Model - without Sampling**

   Defining the model:

   (a) We've used Keras' Sequential () to instantiate a model. It takes a group of sequential layers and stacks them together into a single model. Into the Sequential () constructor, we pass a list that contains the layers we want to use in our model.

   (b) We have made several Dense layers and a single Dropout layer in this model. We have made the input shape equal to the Maximum Sequence Length.

   (c) We defined Embedding Layer on the first layer as the input of that layer.

   (d) There are 200 neurons in Convolution layers, and It has ReLU as activation function and 128 neurons for LSTM layer.

   (e) There are 100 neurons in dense layer. This is typically up to testing - putting in more neurons per layer will help extract more features, but these can also sometimes work against the goal. It has ReLU as activation function.

   (f) Finally, we have a Dense layer with size as the output layer. It has the SoftMax activation function.

(g) At last, we measure the loss with categorical cross entropy function, the efficient ADAM optimization algorithm is used to find the weights and the accuracy metric is calculated and reported each epoch.

(h) We had to add some call backs to the model for early stopping and reduce on plateau if we see no improvement in loss.

2. **Recurrent Neural Network (RNN) - GRU Model - without Sampling**

Model can be defined as:

(a) We've used Keras' Sequential () to instantiate a model. It takes a group of sequential layers and stacks them together into a single model. Into the Sequential () constructor, we pass a list that contains the layers we want to use in our model.

(b) We have made several Dense layers and a single Dropout layer in this model. We've made the input shape equal to the Maximum Sequence Length

(c) We defined Embedding Layer on the first layer as the input of that layer.

(d) We added a GRU function layer with 128 neurons.

(e) There are 100 neurons in dense layer. This is typically up to testing - putting in more neurons per layer will help extract more features, but these can also sometimes work against the goal. It has ReLU as activation function.

(f) Finally, we have a Dense layer with size as the output layer. It has the SoftMax activation function.

(g) At last, we measure the loss with categorical cross entropy function. The efficient ADAM optimization algorithm is used to find the weights and the accuracy metric is calculated and reported each epoch.

3. **LSTM model (Bidirectional) - without Sampling**

   Model can be defined as:

   (a) We created two copies of the hidden layer, one fit in the input sequences as-is and one on a reversed copy of the input sequence. By default, the output values from these LSTMs will be concatenated.

   (b) We defined Embedding Layer on the first layer as the input of that layer.

   (c) We added a LSTM layer with 128 neurons.

   (d) There are 100 neurons in dense layer. This is typically up to testing - putting in more- neurons per layer will help extract more features, but these can also sometimes work against the goal. It has ReLU as activation function

   (e) Finally, we have a Dense layer with size as the output layer. It has the SoftMax activation function.

   (f) At last, we measure the loss with categorical cross entropy function. The efficient ADAM optimization algorithm is used to find the weights and the accuracy metric is calculated and reported each epoch.

Performance comparison of the RNN with LSTM, RNN with GRU model and Bidirectional LSTM model:

The performance comparison of the models has been done based on the classification metrics such as training and testing accuracy, f1-score and recall.

Observations and Insights:

- The difference between training accuracy and testing accuracy is not high. There is scope for much improvement in deep learning models and the testing accuracies of these models look promising.

Table 2.1: Comparison of the Deep Learning Models without Resampling

Performance comparison of the Deep Learning Techniques				
Classifier	Training Accuracy	Testing Accuracy	f1-Score	Recall
RNN-LSTM Model - without Sampling	0.496439	0.464688	0.307287	0.464688
RNN-GRU Model - without Sampling	0.643917	0.513353	0.431207	0.513353
Bi-Directional LSTM Model - without Sampling	0.659050	0.525223	0.448620	0.525223

- The model seems to be over-fitting even with high need of tuning the model, but we still observe the over-fitting already
- The low accuracy is suspected to be due to imbalanced dataset used for training and testing.
- We need to work upon the resampling the data to make the model work better.
- If we see in the above results, we see that RNN-LSTM Model has more room to be tuned without getting over-fitted.
- We need to explore ways to improve & fine tune the model performance without over-fitting.

Future steps planned for deriving better results

1. Data Imbalance Rationalization: Data set will be resampled based on multiple approaches:
    - Creating separate single target group for not well represented groups

(may be with 20 or less assigned tickets) and then classify against highly represented group.

- The reclassify the group (cluster of meager groups) into the original groups.
- May be total drop off from some groups that are very sparsely represented (may be with less than 5 observations).

2. Caution to be taken for avoiding data leak between training and testing runs.
3. Hyper-parameter tuning for both Traditional ML models.
4. Playing with the learning rate to derive optimal speed and avoid minima jumps.
5. Increasing epochs for model to avoid under-fitting
6. Will also try to find any new feature that is more defining to the assignment groups.
7. Increasing number of layers in Deep Learning Model.
8. Increasing number of neurons in hidden layers.

2.1.2 Model Improvement 2: Utilization of Resampled Training Dataset

Resampling is the method that consists of drawing repeated samples from the original data samples. Resampling is a methodology of economically using a data sample to improve the accuracy and quantify the uncertainty of a population parameter.

1. Grouping of Dataset based on ticket count
    - The groups with tickets less than 100 are grouped into a new Group - "GRP_Rare".

- A new dataframe (df_sample2) is created to hold these group data separately.
- The dataframe "df_sample1" holds all the groups and a new column to be added where the Group Name is given as "GRP_Rare", if the corresponding record belongs to a group with ticket count <= 100.

2. Resampling of Training datasets: The training datasets were resampled before fitting into the models.

3. Traditional Models with Resampling.

Comparison of Traditional ML Models with Resampling

After fitting the traditional ML models after the resampling was done, the results obtained were as follows:

Table 2.2: Comparison of Traditional ML Models with Resampling

Performance comparison of Traditional ML Models with Resampling				
Classifier	Training Accuracy	Testing Accuracy	f1-Score	Recall
ADA Boost Classifier Model - with Resampling	0.499750	0.328979	0.346974	0.328979
Naive Bayes Model - with Resampling	0.923192	0.328385	0.359826	0.328385
Decision Tree Model - with Resampling	0.960207	0.412114	0.443202	0.412114
Bagging Classifier Model - with Resampling	0.959864	0.418646	0.451072	0.418646
KNN Model - with Resampling	0.955855	0.410333	0.452627	0.410333
XG Boost Classifier Model - with Resampling	0.954104	0.451306	0.478148	0.451306
Logistic Regression Model - with Resampling	0.954002	0.472684	0.503911	0.472684
Linear SVM Model - with Resampling	0.958790	0.481591	0.511182	0.481591
Random Forest Classifier Model - with Resampling	0.960207	0.551069	0.528577	0.551069

4. Deep Learning Models with Resampling.

Comparison of Deep Learning Models with Resampling

Resampling has been done on the data before fitting onto the deep learning models and the results are as follows:

Table 2.3: Comparison of Deep Learning Models with Resampling

Performance comparison of Deep Learning Models with Resampling				
Classifier	Training Accuracy	Testing Accuracy	f1-Score	Recall
RNN-LSTM Model - with Resampling	0.959707	0.476841	0.487695	0.476841
RNN-GRU Model - with Resampling	0.959901	0.505344	0.514237	0.505344
Bi-Directional LSTM Model - with Resampling	0.959846	0.525534	0.524968	0.525534

Conclusion on Model Improvement 2:

- The performance of the models is not increasing as per expectations.
- We need to utilize better techniques for text classifications given such datasets.
- We need to use grouping in which the datasets are divided into groups as per ranges of ticket counts.
- We need faster and better performing text classification modelling technique and we have decided to use FastText which is developed by Facebook.

2.2 Model Evaluation

Based on the conclusion of Model Improvement 2 Process Results, we decided to proceed with Grouping and FastText method for improvement of model performance.

2.2.1 Model Improvement 3: Utilizing Grouping and FastText

Grouping: The partitioning of the values (also called grouping) of a categorical attribute aims at constructing a new synthetic attribute which keeps the information of the initial attribute and reduces the number of its values.

FastText: A brief introduction FastText is a library for learning of word embeddings and text classification created by Facebook's AI Research lab(FAIR) in 2015. The model allows one to create an unsupervised learning or supervised learning algorithm for obtaining vector representations for words. Facebook makes available pre-trained models for 294 languages. FastText is an open-source, free, lightweight library that allows users to learn text representations and text classifiers. It works on standard, generic hardware. Models can later be reduced in size to even fit on mobile devices.

In a nutshell,

- A sentence/document vector is obtained by averaging the word/n-gram embeddings.
- For the classification task, multinomial logistic regression is used, where the sentence/document vector corresponds to the features.
- When applying FastText on problems with many classes, you can use the hierarchical SoftMax to speed-up the computation.
- Starts with word representations that are averaged into text representation and feed them to a linear classifier (multinomial logistic regression).
- Text representation as a hidden state that can be shared among features and classes.
- SoftMax layer to obtain a probability distribution over pre-defined classes.
- Uses a bag of n-grams to maintain efficiency without losing accuracy. No explicit use of word order.

- Uses hashing trick to maintain fast and memory efficient mapping of the n-grams.
- It supports multiprocessing during training.

FastText vs Deep Learning for text classification: For different text classification tasks FastText shows results that are on par with deep learning models in terms of accuracy, though an order of magnitude faster in performance.

Grouping of Dataset based on ticket count

- The groups with tickets less than 40 are grouped into a new Group - GRP_LE40 and all the other ticket categories are considered individually for final evaluation & testing. GRP_0 is also considered individually since it has highest no of the ticket counts and quite higher than the rest.
- But during the model building process, we grouped the rest of the ticket categories into groups based on ticket counts.
- New data frames are created to hold these group data separately.
- The data frame "df_sample1" holds all the groups.
- We are using supervised fasttext here for our modeling.

The grouping was done based on the below divisions:

1. Sample 1: GRP_0 vs GRP_Others1
2. Sample 2(df_sample2): GRP_Others1 - GRP_LE100, GRP_LE289GT200, GRP_LE200GT100 & GRP_8
3. Sample 3(df_sample3): In GRP_LE289GT200 - Groups with tickets counts <= 289 & > 200
4. Sample 4(df_sample4): Groups with ticket counts <= 200 & > 100(GRP_LE200GT100)
5. Sample 5(df_sample5): Groups with ticket counts <= 100(GRP_LE100:- GRP_LE40 & GRP_LE100GT40)

6. Sample 6(df_sample6): Groups with ticket counts <= 100 & > 40(GRP_LE100GT40)

The process followed on each sample is as follows:

1. Utilization of Classifications Techniques on each Sample (i.e. df_sample1, df_sample2, df_sample3, df_sample4, df_sample5, df_sample6)

2. Classification using FastText(supervised) on the unsampled dataset of each groups.

3. Evaluation of each model based on performance metrics such as Testing accuracy, f1-score and recall.

4. Testing of Sample 1 Model will help us determine whether the test text belongs to GRP_0 or not.

5. Testing of Sample 2 Model will help us determine whether the test text belongs to GRP_8 or not. It also will point us to the group it belongs.

6. Testing of Sample 3 Model will help us determine the group to which the test text belongs among the groups with tickets counts <= 289 & > 200.

7. Testing of Sample 4 Model will help us determine the group to which the test text belongs among the groups with tickets counts <= 200 & > 100.

8. Testing of Sample 5 Model will help us determine whether the test text belongs to GRP_LE40 or not. If False, then it belongs to one among the groups with tickets counts <= 100 & > 40 (GRP_LE100GT40). GRP_LE40 is considered as a new single group.

9. Testing of Sample 6 Model will help us determine the group to which the test text belongs among the groups with tickets counts <= 100 & > 40 (GRP_LE100GT40).

Autotuning of hyperparameters (in supervised fasttext):

- Finding the best hyperparameters is crucial for building efficient models. However, searching the best hyperparameters manually is difficult. Parameters are dependent and the effect of each parameter vary from one dataset to another.

- FastText's autotune feature allows us to find automatically the best hyperparameters for your dataset.

- By default, the search will take 5 minutes. You can set the timeout in seconds with the -autotune-duration argument. While autotuning, fastText displays the best f1-score found so far. If we decide to stop the tuning before the time limit, we can send one SIGINT signal. FastText will then finish the current training and retrain with the best parameters found so far.

- We used autotuning for binary classification during model building of df_sample1 & df_sample5.

The results obtained are as follows:

Table 2.4: All 6 FastText Models with Grouping

Performance of All 6 FastText Models with Grouping			
Classifier	Testing Accuracy	f1-Score	Recall
FastText Model1 - with Grouping	0.842730	0.842730	0.842730
FastText Model2 - with Grouping	0.663300	0.663300	0.663300
FastText Model3 - with Grouping	0.860000	0.860000	0.860000
FastText Model4- with Grouping	0.731278	0.731278	0.731278
FastText Model5 - with Grouping	0.800000	0.800000	0.800000
FastText Model6 - with Grouping	0.800000	0.800000	0.800000

Conclusion

- We have better results based on the grouping and FastText modeling technique which we employed for text classification.
- We could conclude from this study that text classification does not have definite method for building models. We must employ several techniques and find those set of techniques which could serve our model building purpose.

2.3 Comparison to Benchmark

We first analyzed the dataset provided to us, understood the structure of the data provided - number of columns, field, datatypes etc.

We did Exploratory Data Analysis to derive further insights from this data set and we found that Data is very much imbalanced, there are around ∼45% of the Groups with less than 20 tickets.

Few of the tickets are in foreign language like German, the data has lot of noise in it, for e.g.- few tickets related to account setup are spread across multiple assignment groups. We performed the data cleaning, google translation and pre-processing.

Here in this comparison of different traditional ML models, we can observe a substantial difference in the accuracy of training and test sets. Major reason is possibly the imbalanced data distribution of the dataset used and the inability of the model to learn and adapt during training.

We need to check if these issues can be handled by resampled data and using deep learning techniques.

2.3.1 Model Improvement 1 (Benchmark)

Model Improvement 1 - Utilizing Deep Learning Techniques(f1-scores):

- Recurrent Neural Network(RNN) - LSTM Model: 0.307287
- Recurrent Neural Network(RNN) - GRU Model: 0.431207

- Bidirectional LSTM Model: 0.448620

Observation:

- The difference between training accuracy and testing accuracy is not high. There is scope for much improvement in deep learning models and the testing accuracies of these models look promising.
- The model seems to be over-fitting for training dataset with low testing accuracy, but already we have observed this over-fitting.
- The low accuracy is suspected to be due to imbalanced dataset used for training and testing.
- We must resample the data or do some technique to make the model work better.
- If we observe the above results, we see that RNN-LSTM Model has more room to be tuned without getting over-fitted.
- We need to explore ways to improve & fine tune the model performance without over-fitting.

2.3.2 Model Improvement 2 (Benchmark)

Model Improvement 2 - Utilization of Resampled Training Dataset(f1-scores):

- ADA Boost Classifier Model - with Resampling: 0.346974
- Naive Bayes Model - with Resampling: 0.359826
- Decision Tree Model - with Resampling: 0.443202
- Bagging Classifier Model - with Resampling: 0.451072
- KNN Model - with Resampling: 0.452627
- XG Boost Classifier Model - with Resampling: 0.478148

- Logistic Regression Model - with Resampling: 0.503911
- Linear SVM Model - with Resampling - 0.511182
- Random Forest Classifier Model - with Resampling -0.528577
- RNN-GRU Model - with Resampling: 0.487695
- RNN-LSTM Model - with Resampling: 0.514237
- Bidirectional LSTM Model - with Resampling: 0.524968

Observation:

- The performance of the models is not increasing as per expectations.
- We must utilize better techniques for text classifications given such datasets.
- We must use grouping in the datasets are divided into groups as per ranges of ticket counts.
- We need faster and better performing text classification modelling technique and we have decided to use fasttext which is developed by Facebook.

So, the benchmark f1 score we declare is > 0.53. We have highly imbalanced data and resampling did not help as shown above.

2.3.3 Model Improvement 3 (Benchmark)

Model Improvement 3 - Utilization of Grouping and FastText(f1-scores):

- FastText Model1 - with Grouping: 0.842730
- FastText Model2 - with Grouping: 0.663300
- FastText Model3 - with Grouping: 0.860000
- FastText Model4 - with Grouping: 0.731278
- FastText Model5 - with Grouping: 0.800000

- FastText Model6 - with Grouping: 0.800000

With our final and third improvement with Grouping and fast text we see the f1-score have improved to 0.86 which is much higher than the original benchmark which we had set(> 0.53).

2.4 Visualization

Necessary visualizations in terms of bar charts and pie charts are shown in each step to help us understand comparisons visually. Graphs with model accuracy comparisons are shown as well.

2.4.1 Visualization: Initial Observation of Dataset

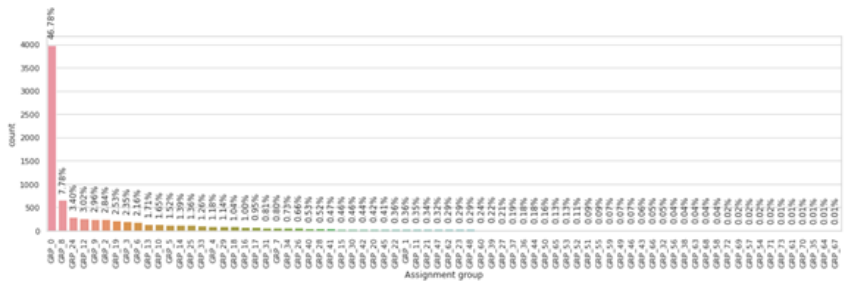

Figure 2.1: Initial Observation of Dataset

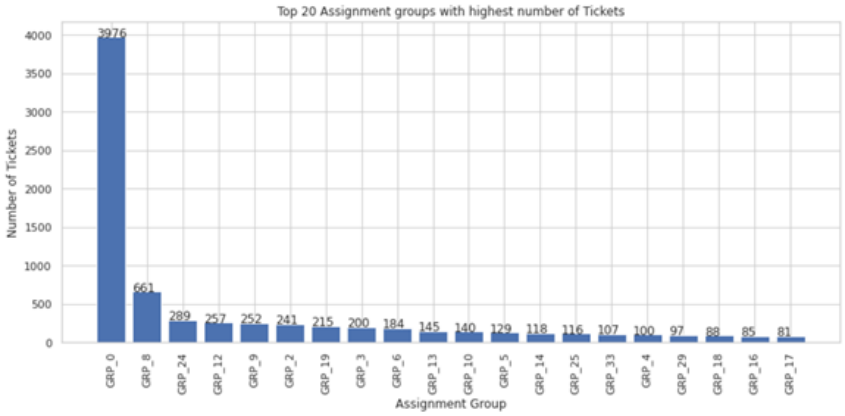

Figure 2.2: Top 20

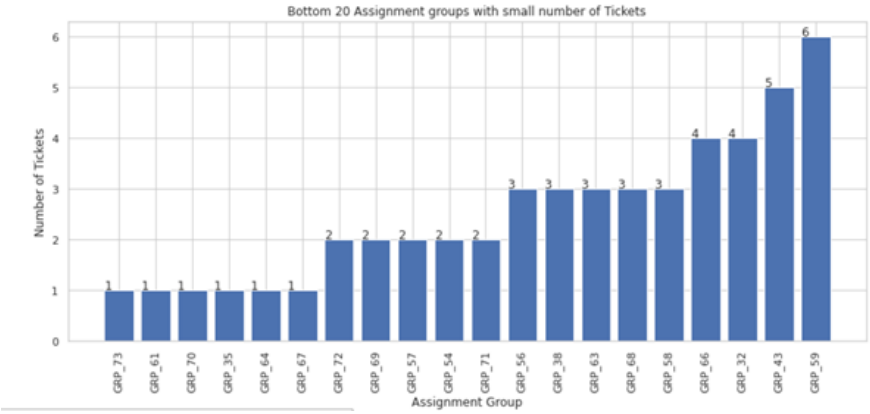

Figure 2.3: Bottom 20

2.4.2 Visualization: Pre-Processing

Figure 2.4: Creating World Cloud

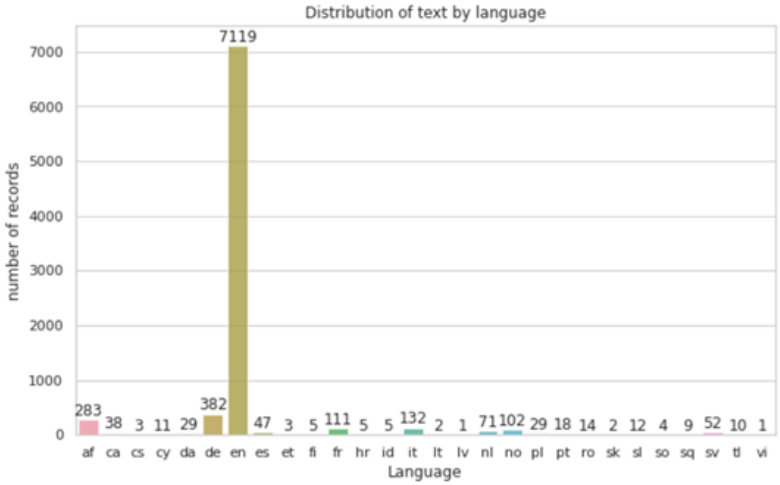

Figure 2.5: Language Distribution in the Dataset - Bar Graph

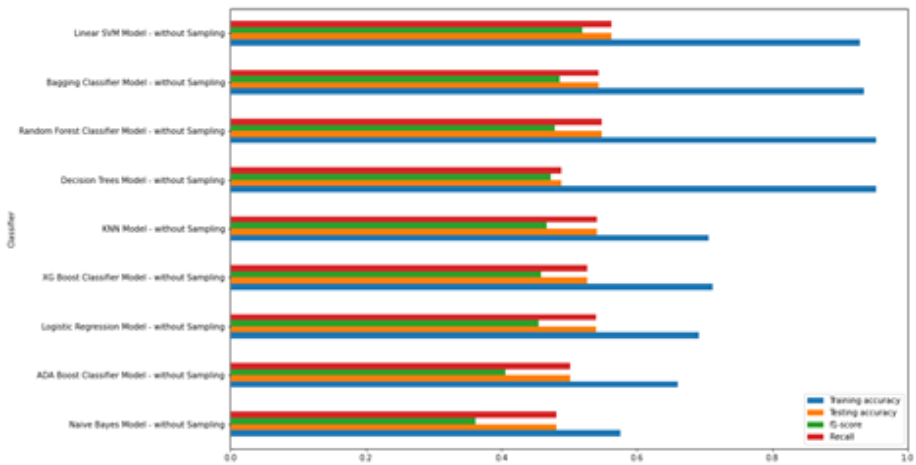

Figure 2.6: Compare Performance of Traditional ML Models - without resampling

2.4.3 Visualization: Performance Comparison without resampling

2.4.4 Visualization: Model Improvement 1

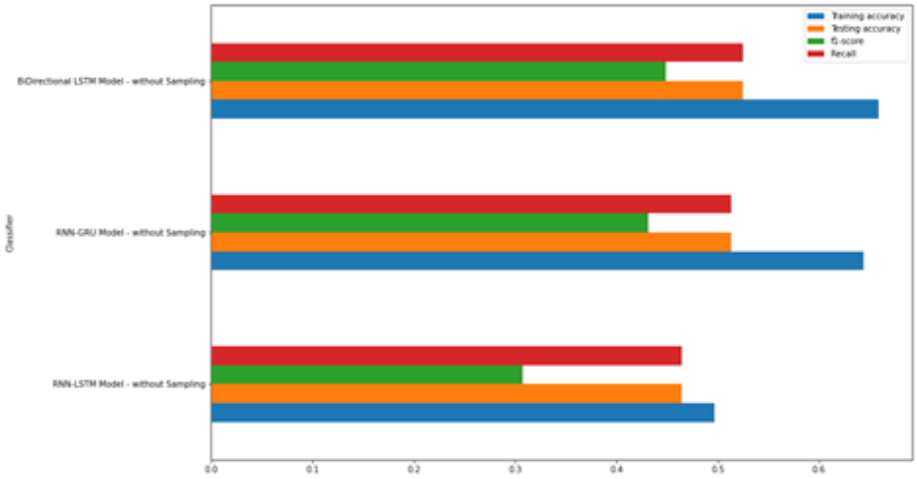

Figure 2.7: Compare Performance of Models Utilizing Deep Learning Techniques

2.4.5 Visualization: Model Improvement 2

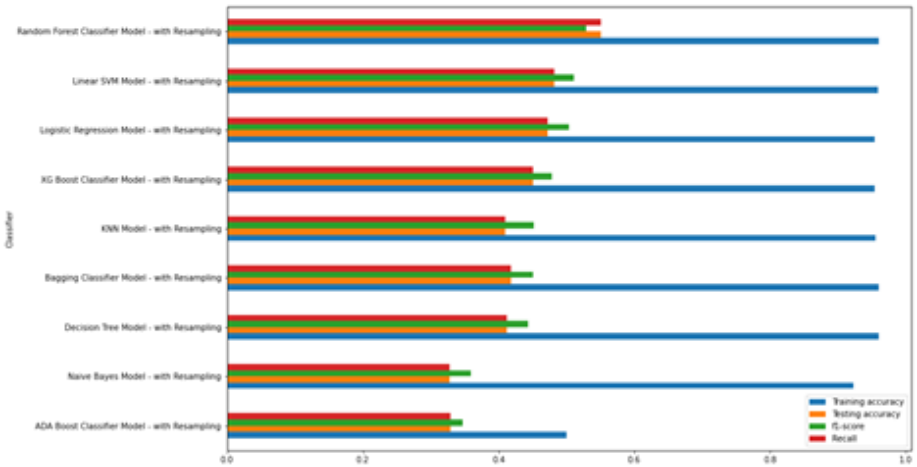

Figure 2.8: Compare Performance of Traditional ML Models Utilizing Resampled Training Dataset

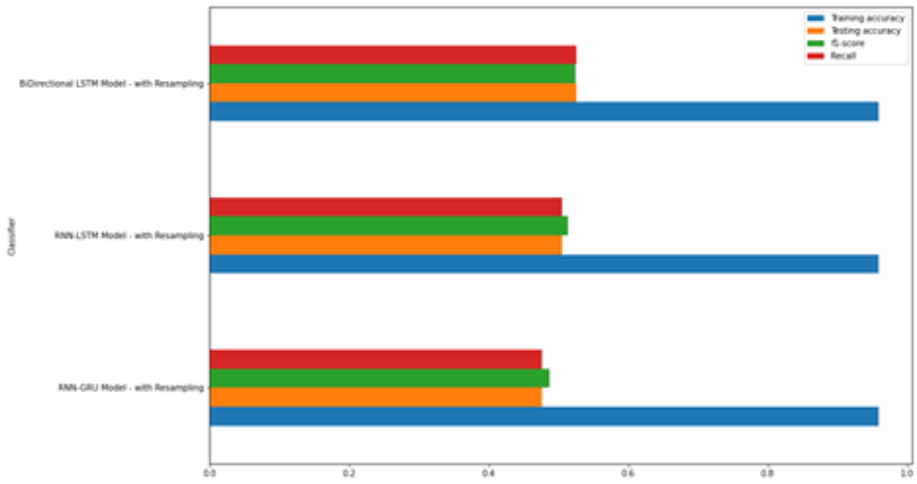

Figure 2.9: Compare Performance of Deep Learning Models Utilizing Resampled Training Dataset

2.4.6 Visualization: Model Improvement 3

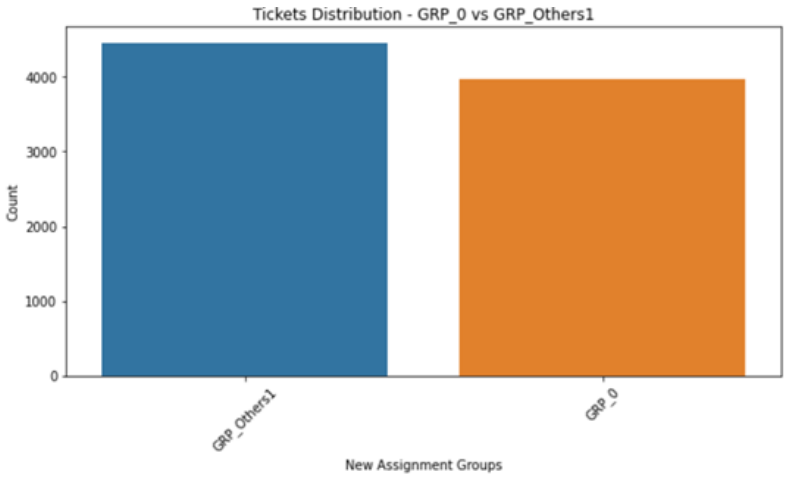

Figure 2.10: Distribution of DF_Sample1

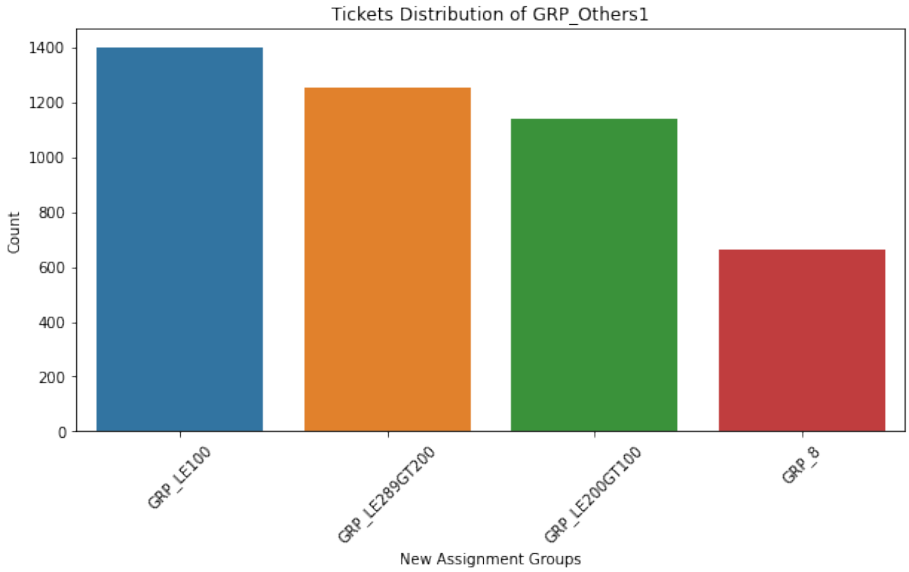

Figure 2.11: Distribution of DF_Sample2

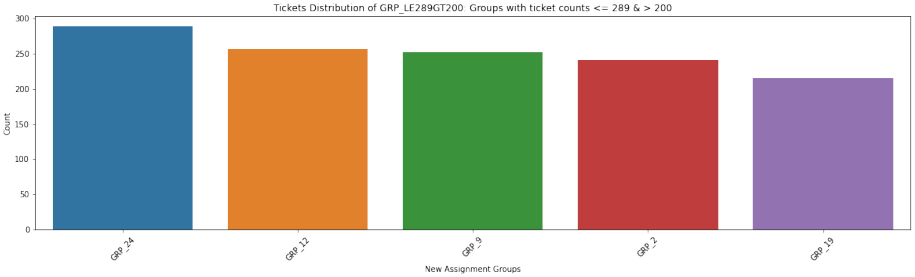

Figure 2.12: Distribution of DF_Sample3

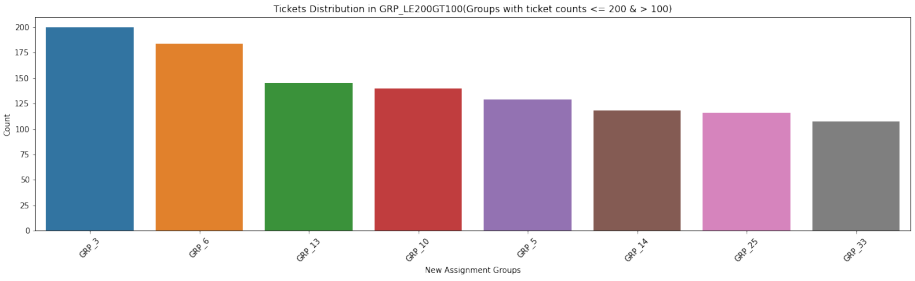

Figure 2.13: Distribution of DF_Sample4

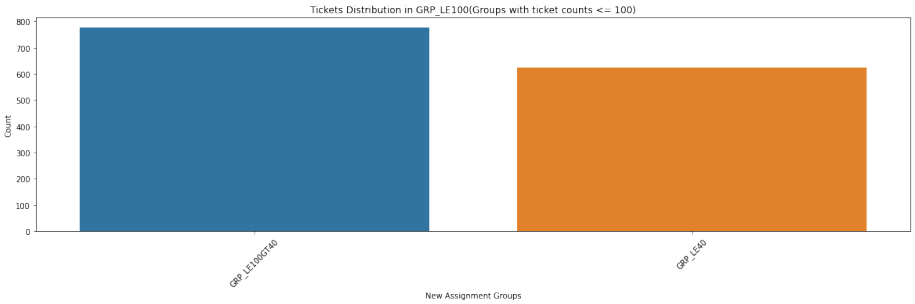

Figure 2.14: Distribution of DF_Sample5

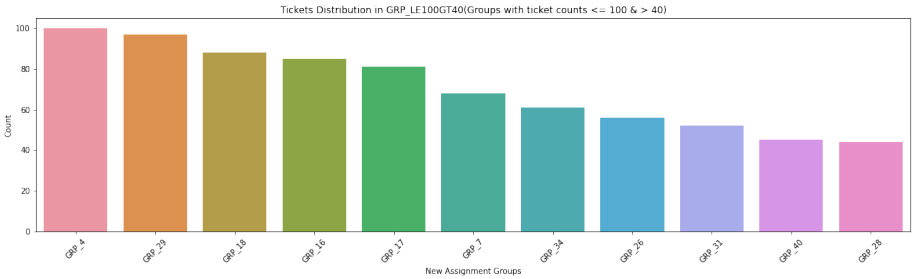

Figure 2.15: Distribution of DF_Sample6

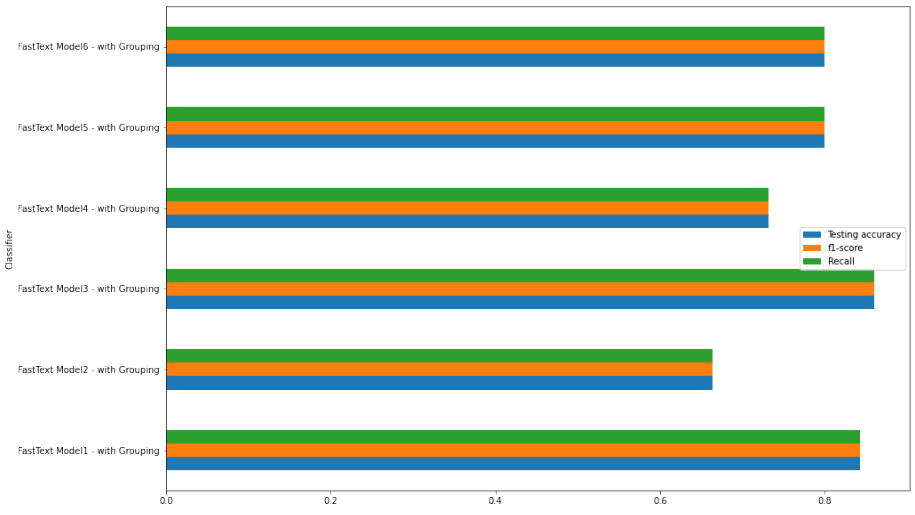

Figure 2.16: Performance of All 6 Models Utilizing of Grouping and FastText

Chapter 3

Conclusion

3.1 Implications on Business

Better allocation and effective usage of the valuable support resources will directly result in substantial cost savings by:

1. Automation of Ticket assignments results directly into reduction of Human error in Ticket handling (25% incidents).
2. $sim1$ FTE effort that used for ticket allocation can be reduced.
3. SLAs can be maintained by effective and timely allocation to and serviced by apt team.
4. Eliminate any Financial penalty associated with missed SLAs.
5. 15 mins Effort spent for SOP review can be saved.
6. Decrease in infra and tech cost for 1 FTEâ€™s allocation work.
7. Directly allocating 54% of tickets to L1 / L2 Team without any check point.
8. Customer Satisfaction can be increased by avoiding incorrect queuing of tickets and having timely resolution.

9. The classification tool, guided by powerful AI techniques, can classify incidents to right functional groups can help organizations to reduce the resolving time of the issue and can focus on more productive tasks.

Confidence Interval: accuracy = 86%

n = 8241

Confidence interval at 95% = $1.96 \times 5\sqrt{\left(\frac{accuracy \times (1-accuracy)}{n}\right)} = 0.0110$

Hence, Interval = 0.86 $\pm$0.0110 = 84.9% to 87.10%

There is a 95% likelihood that the confidence interval [84.9%, 87.1%] covers the true classification of the model on the unseen data.

Recommendations based on EDA:

1. Password Reset process can be automated. No. of Incident Ticket reduction expected by automating password reset: 1246

2. We also have found that Failure of "job scheduler" and "job failure" have significant contribution with 1928 tickets out of 8500, which is 22.68%. Root cause analysis (RCA) need to be performed to further analyze and fix problem jobs. We can possibly reduce the Resource / FTE allocation also by approximately 22.68%.

3.2 Limitations

1. We observed that 37 Assignment Groups had 25 or less tickets, we had to club the 37 Assignment Groups into Others Category. Manual Assignment will be required for the tickets belonging to these 40 Assignment Groups.

2. We have used FastText classifier for final classification. In this approach we have to add label to each text. It might be a limitation for very large training data set.

3. We have used GloVe for word embeddings that can fail to provide any vector representation for words that are not in the model dictionary. To

overcome this issue, FastText is the approach which works well with rare words. So even if a word wasn't seen during training, it can be broken down into n-grams to get its embeddings

3.3 Closing Reflections

We have observed and learned following during the solution development:

1. The data leak from training to test data can really harm the classification capacity.

2. The over fitting issue was observed because of unbalanced data set.

3. We tried SMOTE, Over Sampling and Under sampling but none had resulted the desired results. Ultimately, we performed grouping of classes based on observations and did the FastText classification on the groups.

4. Out of all the text classification methods considered in this study, we found FastText as the fastest and most accurate. With FastText we were able to achieve an accuracy more than 80%.

5. FastText was seen extremely helpful in embeddings of new vocab words found during training. Word2vec and GloVe both fail to provide any vector representation for words that are not in the model dictionary. This is a huge advantage of this method.

6. Though the overall performance was satisfactory, there is still scope for improvement.

7. We could have explored more on adjusting the hyperparameters for FastText and other Deep Learnng Techniques like LSTM-Attention.

Chapter 4

Bibliography

- `https://scikit-learn.org/stable/modules/classes.html` - scikit-learn: API Reference
- `https://fasttext.cc/docs/en/supervised-tutorial.html` - FastText: Text classification

List of Figures

List of Tables