AIML ONLINE CAPSTONE - AUTOMATIC IT TICKET ASSIGNMENT

By,

Deepkumar Aravindkumar Patel

Harika.M

Madhumitha.R

Rachit Vaid

Table of Contents

[1. Introduction 3](#_Toc75555197)

[2. Business Case 4](#_Toc75555198)

[3. Data Collection 4](#_Toc75555199)

[4. Challenges to be Expected 4](#_Toc75555200)

[5. Exploratory Data Analysis 5](#_Toc75555201)

[6. Data Pre-Processing 6](#_Toc75555202)

[7. Creation of train and test sets 12](#_Toc75555203)

[8. Model Building 13](#_Toc75555204)

[8.1 Traditional Machine Learning 13](#_Toc75555205)

[8.1.1 Multinomial Logistic Regression 13](#_Toc75555206)

[8.1.2 Support Vector Machine 14](#_Toc75555207)

[8.1.3 Stochastic Gradient Descent 14](#_Toc75555208)

[8.1.4 Multinomial Naïve Bayes 14](#_Toc75555209)

[8.1.5 K Nearest Neighbor 15](#_Toc75555210)

[8.1.6 Random Forest 15](#_Toc75555211)

[8.1.7 Decision Tree 15](#_Toc75555212)

[8.1.8 XGBoost 16](#_Toc75555213)

[8.2 Deep Learning 16](#_Toc75555214)

[8.2.1 Neural Network 16](#_Toc75555215)

[8.2.2 Bi-Directional LSTM 17](#_Toc75555216)

[8.3 Summary 18](#_Toc75555217)

[9. Fine-Tuning 19](#_Toc75555218)

[9.1 Data Centric Approach 19](#_Toc75555219)

[9.2 Model Centric Approach 20](#_Toc75555220)

[10. Comparison to Benchmark 21](#_Toc75555221)

[11. User Interface 21](#_Toc75555222)

[12. Implications 21](#_Toc75555223)

[13. Limitations 22](#_Toc75555224)

[14. Conclusion 22](#_Toc75555225)

# 1. Introduction

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 the Incident Management process to achieve the above Objective. An incident is something that 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.

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.

# 2. Business Case

In the support process, incoming incidents are analysed and assessed by the organization's support teams to fulfil the request. In many organizations, better allocation and effective usage of the valuable support resources will directly result in substantial cost savings.

Currently the incidents are created by various stakeholders (Business Users, IT Users and Monitoring Tools) within the IT Service Management Tool and are assigned to Service Desk teams (L1 / L2 teams). This team will review the incidents for right ticket categorization, priorities and then carry out initial diagnosis to see if they can resolve. Around ~54% of the incidents are resolved by L1 / L2 teams. In case L1 / L2 is unable to resolve, they will then escalate / assign the tickets to Functional teams from Applications and Infrastructure (L3 teams). Some portions of incidents are directly assigned to L3 teams by either Monitoring tools or Callers / Requestors. L3 teams will carry out detailed diagnosis and resolve the incidents. Around ~56% of incidents are resolved by Functional / L3 teams. In case if vendor support is needed, they will reach out for their support towards incident closure.

L1 / L2 needs to spend time reviewing Standard Operating Procedures (SOPs) before assigning to Functional teams (Minimum ~25-30% of incidents needs to be reviewed for SOPs before ticket assignment). 15 min is being spent for SOP review for each incident. Minimum of ~1 FTE effort needed only for incident assignment to L3 teams.

# 3. Data Collection

The Organizational IT Service Management Tool is the primary source for project data related to the incidents. Details from all the resolved incidents are collected to form the data for this project. The incidents have features like Short Description, Description, Caller, Assignment Group, etc.

# 4. Challenges to be Expected

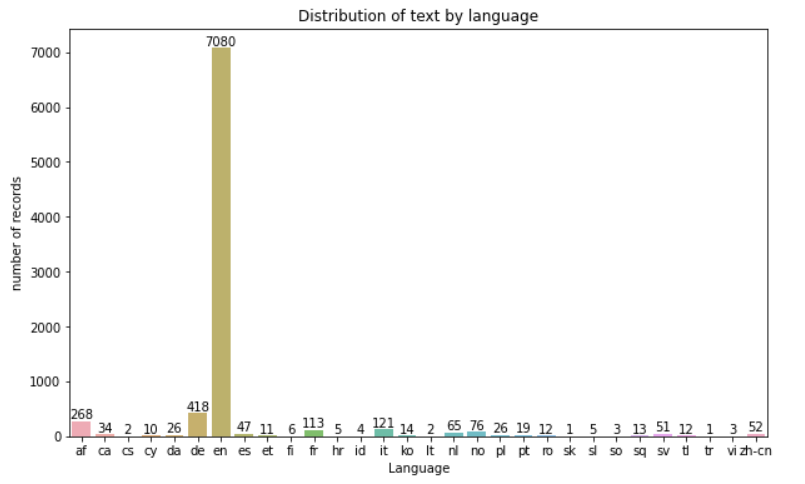
There is no guarantee that the data we receive from the IT Service Management Tool would be clean. There might be several issues with the data and we must be ready to deal with them. Below are some of the issues that can be expected:

* Null or missing values in the tickets. The support teams sometimes fail to fill all the necessary details in the ticket while closing it. We can either remove such tickets from the dataset or perform some imputation to deal with such problems.
* Presence of special characters in the data might be problematic. While building models, we need to ensure that the data fed to the model is clean. Not providing proper data to the model might lead to incorrect results. We can use Regex to remove all the non-word characters which are of no significance to the model.
* Ticket description written in a foreign language. If the IT team is supporting a global application, there are high chances that the description or other details could be written in a different language. In such cases, we either need to manually translate the sentences to a Base language (English in our case) or write additional code to perform the translation based on the volume of data.
* Class imbalance in Multi Class Classification problems might lead to incorrect results. In such cases, we need to use techniques like up-sampling and down-sampling to handle the class imbalance.

# 5. Exploratory Data Analysis

After loading the data from the excel sheet, we did some basic EDA to get an idea on the look and feel of the dataset. The below are some observations:

1. The dataset has 8500 rows/tickets and 4 columns/properties.
2. Each ticket has a short description, a description, caller and the assignment group. All the values are of type ‘object’.
3. There are 74 unique assignment groups in the dataset.
4. About 46.8% of the total tickets are assigned to Group\_0. The rest is distributed among the remaining assignment groups. This shows that there is high Class Imbalance in the target.
5. There are 9 rows in the dataset which contain NaN. All the NaN values have been converted to type string. The rows which previously had NaN values now contain the text ‘NaN’.
6. On careful observation, few tickets in the dataset had descriptions written in a different language. On translating the words manually, we found them to be in German. This indicates that the dataset has tickets written in multiple languages.
7. EDA indicated the presence of foreign languages in the dataset. To find out the languages, we have used the detect function from the ‘langdetect’ library. There are a total of 30 languages used in the dataset. A simple bar chart visualization showed that most of the tickets were written in English followed by tickets in German.



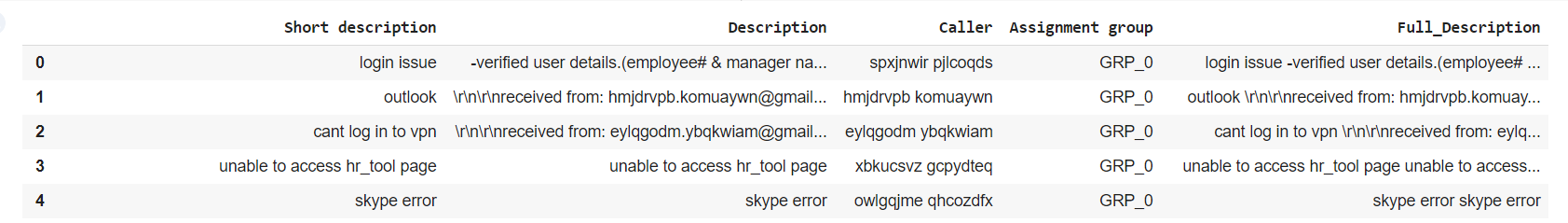
**Figure 1: Distribution of languages in the text**

# 6. Data Pre-Processing

The dataset provided to us has 3 features and 1 target. But we cannot directly feed it to the model as input as the data isn’t clean yet. It contains special characters, stop words and lots of irrelevant text ‘sent’, ‘email’, ‘from’, etc. To get high model accuracy, we need to ensure that the model input is clean.

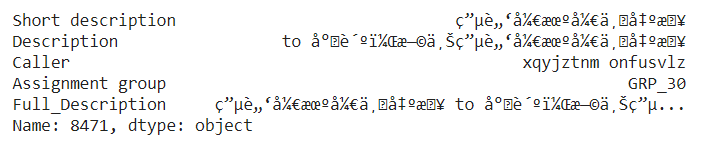
The following are some methods implemented to clean out the text in the dataset:

1. We merged the ‘Short description’ and ‘Description’ columns to create a new feature called ‘Full\_Description’.

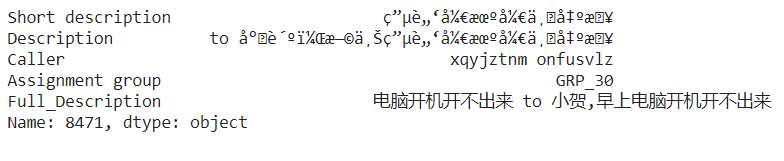


**Figure 2: Creation of new feature**

1. When dealing with text written in multiple languages, there’s a high chance of encountering Mojibakes. Mojibake often occurs when a character coding is incorrectly tagged in a document, or when a document is moved to a system with a different default coding than its preceding location. Using the ‘ftfy’ library, we have identified the mojibakes in the dataset and fixed them.

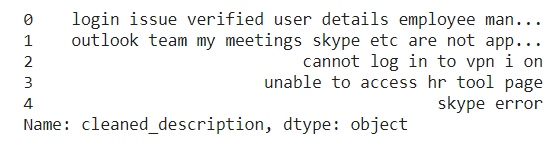


**Figure 3: Original text**



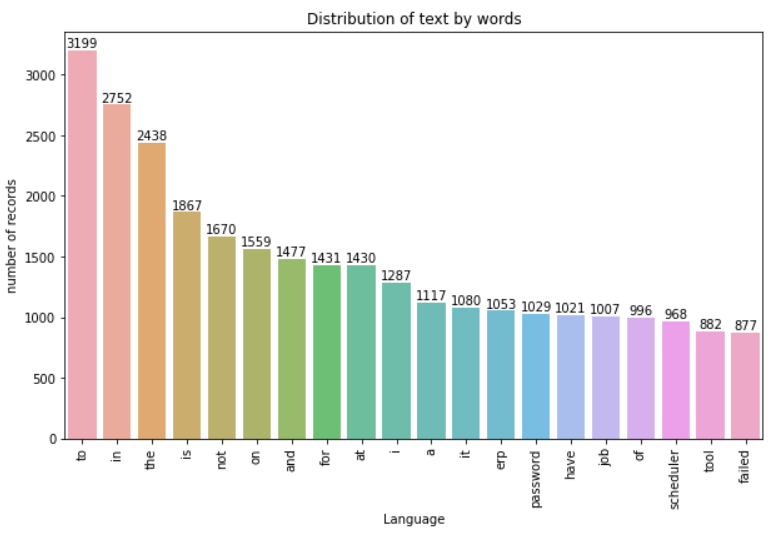
**Figure 4: Text after removing mojibakes**

1. With the help of regex, we removed all the irrelevant content from the new feature ‘Full\_Description’. Irrelevant text includes all the special characters, whitespaces, URLs, email addresses, salutations, etc.



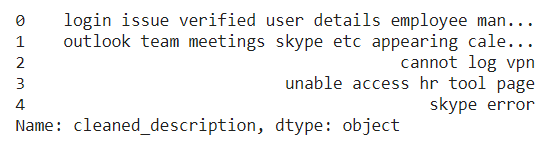
**Figure 5: Clean text**

1. Calculated word count to find out the most frequent words. Most of the words were stop words that do not add any meaning to the content.



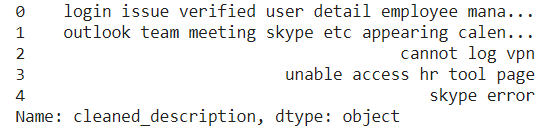
**Figure 6: Bar chart for most frequent words**

1. With the help of ‘stopwords’ corpus from nltk, we remove all the stop words and create ‘cleaned\_description’.



**Figure 7: Text after removing stopwords**

1. We performed Lemmatization on the ‘cleaned\_description’ to find the root form or the lemma of the words.



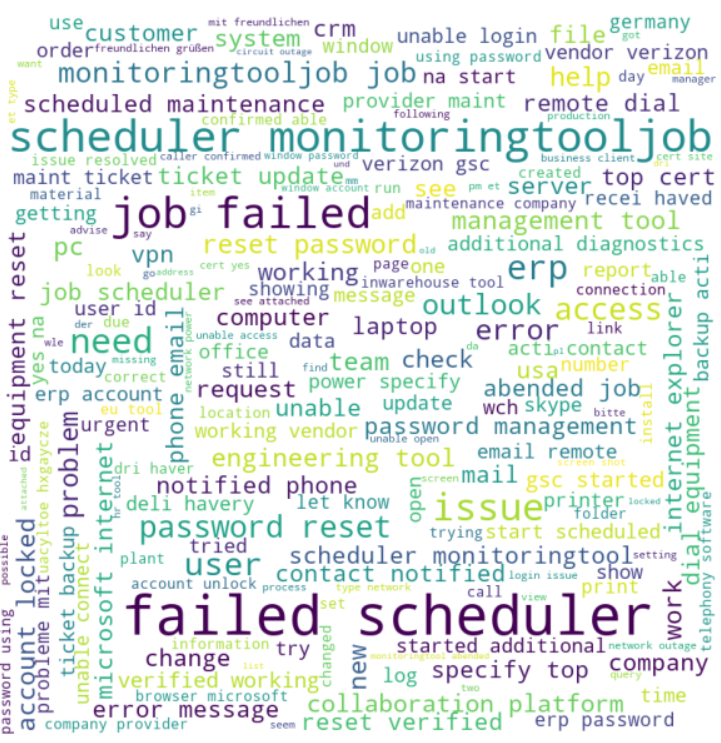
**Figure 8: Lemmatised text**

1. After lemmatizing the text, we dropped those rows whose description contained only one word.
2. A lot of visualizations were done like bar chart, pie chart and pareto chart to check the distribution of the tickets among various groups. Some of the observations are listed below:

* Top 5 groups with highest tickets are - 0, 8, 24, 12 and 9.
* Bottom 5 groups with lowest tickets are - 70, 64, 67, 73 and 35.
* There are 40 groups with just 30 or less tickets assigned, amongst which 6 groups happen to be assigned with just 1 ticket, 4 groups with just 2 tickets each and 5 groups with 3 tickets each.
* There are 15 assignment groups that have more than 100 tickets which accounts to only 20.3% of the overall dataset.

1. WordClouds were created for the whole dataset and a few individual groups. The findings are as follows:

* Group 0 seems to have issues related to password reset, access, login, etc.
* Groups that have words similar to Group\_0 are 2, 7,12,14,17, etc.
* Group 8 has issues related to monitoring tool, scheduled maintenance, outage, job failure, etc.
* Groups that have similar words as that of Group\_8 are 9, 6, 5, 10, 4, 47, 0, 45, 12, 1, 13, 14, 29, 18, etc.
* Group\_12 tickets mostly revolve around server, asa deny, dst outside, outside access.
* Group\_24 consists of German language which needs translation.



**Figure 9: WordCloud created for cleaned description**

1. Analysis was done on the caller column and WordClouds were created. The findings are as follows:

* Top 10 frequent callers were identified from the database.



**Figure 10: Top 10 frequent callers**

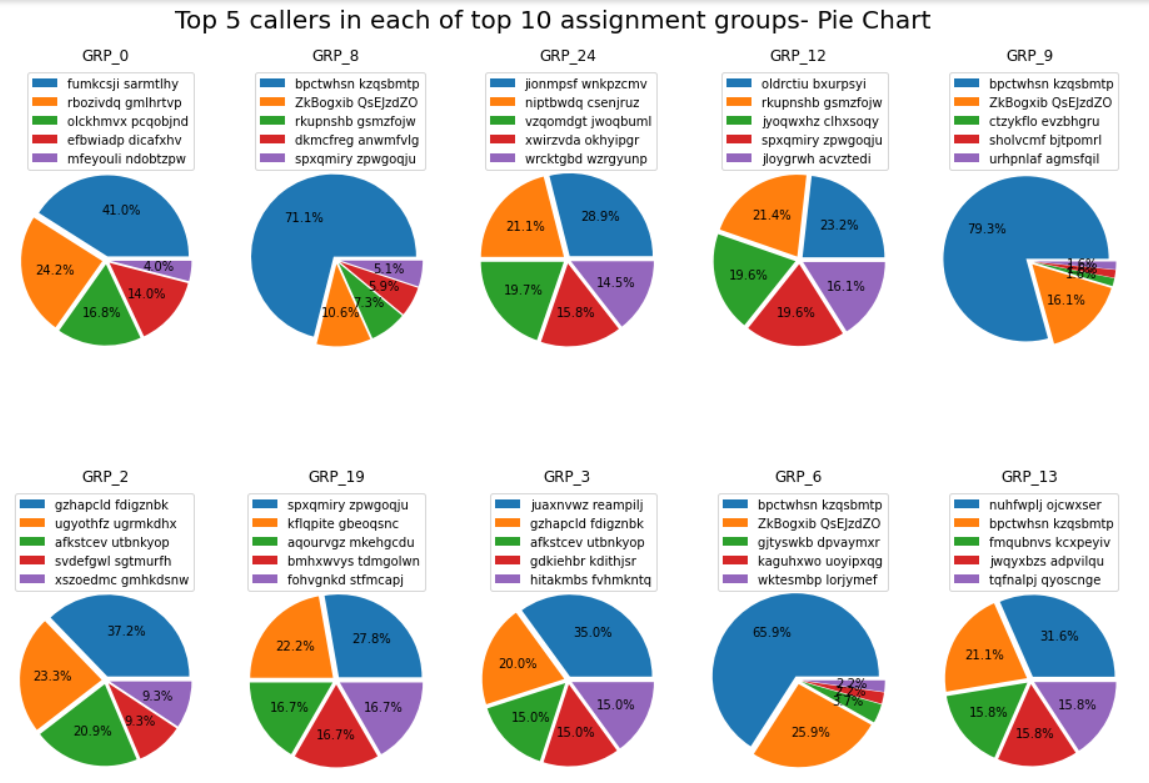
* The top callers ‘pctwhsn kzqsbmtp’ & ‘ZkBogxib QsEJzdZO’ mostly addressed issues related to job scheduler and monitoring tool.
* Caller ‘fumkcsji sarmtlhy’ addressed issues related to ticket updates and specific ticket related issues.
* Tickets created by caller ‘rbozivdq gmlhrtvp’ are mostly related to basic front desk resolvable issues such as loud noise, general enquiry, loud noise, etc.



**Figure 11: WordCloud created for caller**

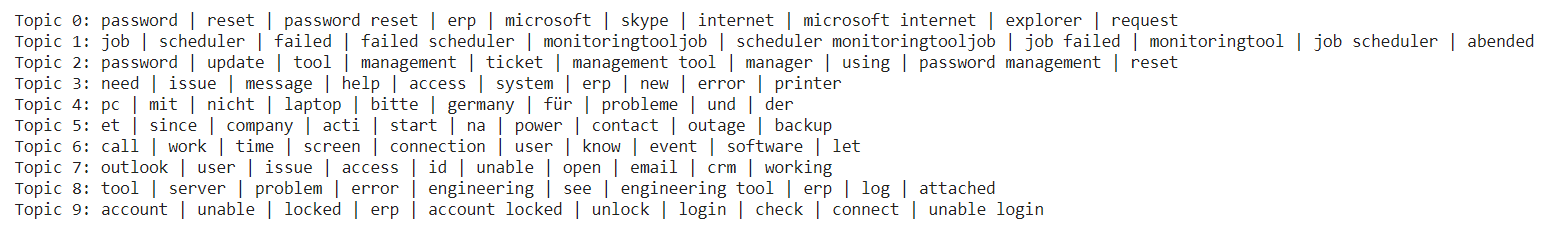
1. Performed some analysis on the relation between caller and assignment group and found the below insights:

* There are overall 2950 callers in the dataset.
* 15 callers are involved in raising tickets for multiple assignment groups, overall contributing to 281 tickets in the dataset.
* The top 5 caller ticket distribution in every group is widely distributed, almost having a balanced distribution. This indicated that most of the group resolves issues widely over various fields impacting business.



**Figure 12: Distribution of top 5 callers in top 10 assignment groups**

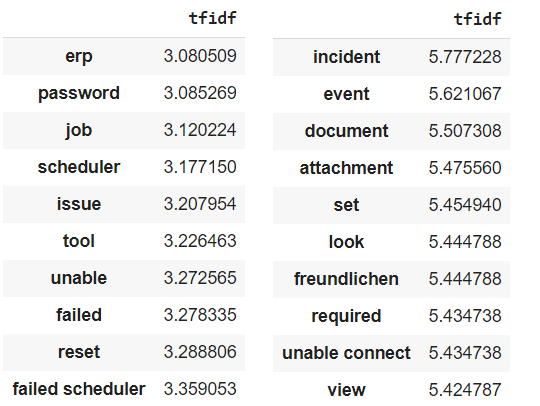
1. Unigram, bigram and trigram models are used to get the most frequent words or phrases from the cleaned description.
2. Used LDA modelling (Latent Dirichlet Allocation) to discover topics that are hidden in the cleaned description. The below are few topics and the relevant words in each topic:



**Figure 13: Topics created using LDA model**

1. Performed TF\_IDF i.e., the term frequency – inverse document frequency to find out the relevancy of the words in the corpus. Few observations are listed below:

* Words that have low significance in the dataset are ‘Erp, job, passwords, jobscheduler, failed’ etc.
* Words that have high significance in the dataset are ‘outside accessgroup, dst outside, src inside, src, accessgroup’, etc.



**Figure 14: Tf-Idf values**

# 7. Creation of train and test sets

To create the train and test sets, we first had to create the X and Y variable. X contains the features and Y contains the targets. The creation of X and Y variables depend on the type of model used. The input given to a model depends on the underlying algorithm and other factors. Hence, we followed the below steps to create the train and test sets:

1. We took the tf-idf vectors created previously in the Data Pre-Processing step and converted it to an array. The values were stored in a variable called X.
2. Performed Label Encoding on the Assignment Group column which is the original target. On performing label encoding, we got a list of numbers for the targets. These label-encoded values were put in a new column in the DataFrame and also in a new variable called Y.
3. Using the X and Y variables and the train\_test\_split function, we got our train and test sets.
4. For the traditional Machine Learning models, we used the train set created above for training the model.
5. For the deep learning model, we had to do further processing on the data. Word embedding was created for the ‘cleaned\_description’ column using techniques like bag of words, word2vec, GloVe embeddings, etc. One-hot encoding was performed on the Y variable.

# 8. Model Building

The targets present in the dataset are categorical values indicating that it is a classification problem. Preliminary analysis done on the data indicated that we had multiple classes in the target. Thus, we need to implement algorithms that support multiclass classification.

There are several models/algorithms available that could help us with the given problem. We could use the simple models which involve traditional Machine Learning algorithms. And we could also implement deep learning models which are highly used in the field of Artificial Intelligence. Our approach here is to divide the Modelling into two phases, one using Machine Learning and other using Deep Learning models.

## 8.1 Traditional Machine Learning

There are several algorithms in Supervised Machine Learning that support multi class classification problems. Below are some algorithms that we have implemented in this project:

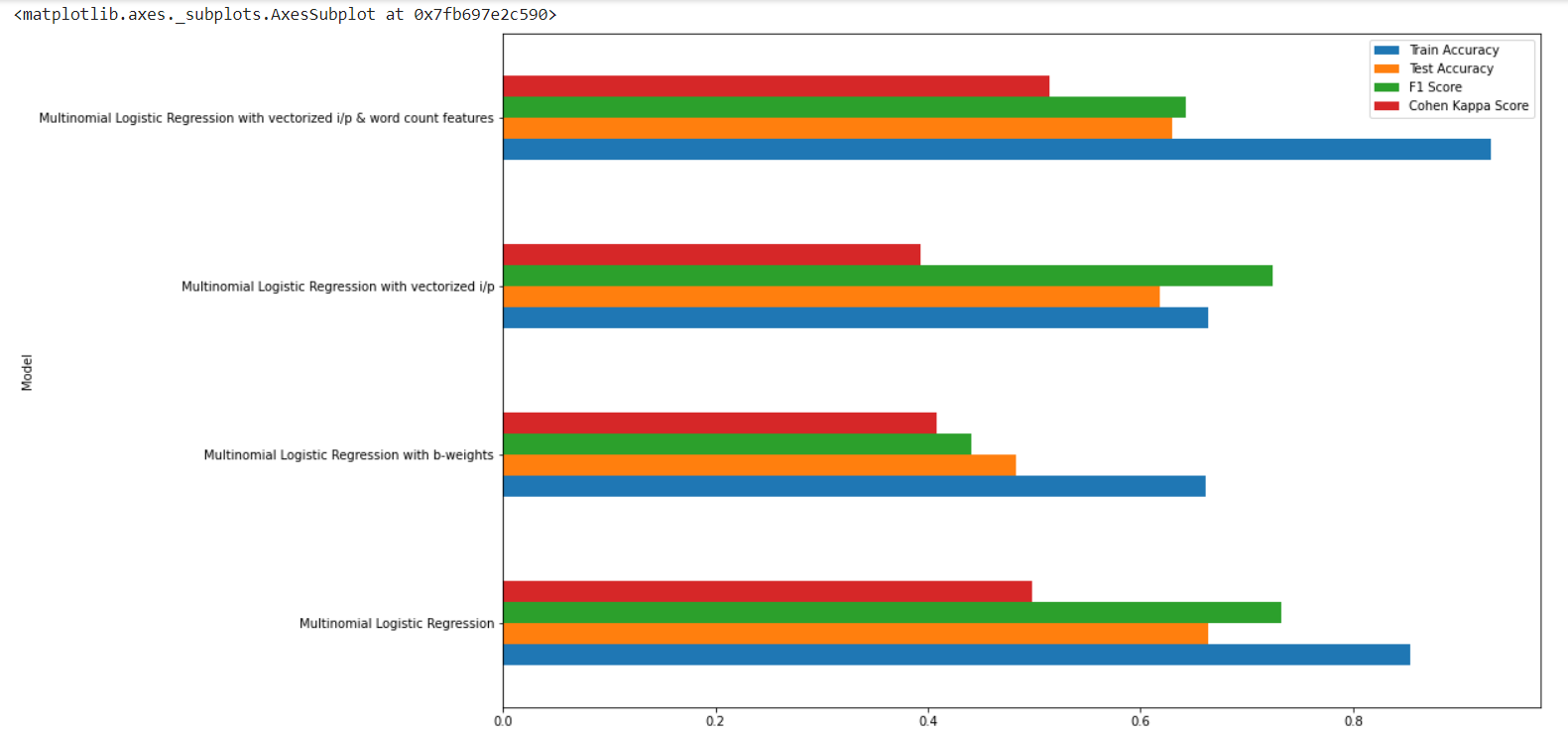
### 8.1.1 Multinomial Logistic Regression

A logistic regression model was created using the parameters multi\_class='multinomial' and solver='lbfgs'. The tokenized description and label encodings were provided as the model input. Training score was 85.43% and testing score was 66.34% indicating overfitting. F1-score was high for groups that were more frequent compared to less frequent groups.

We built a second logistic regression with the parameter class\_weight='balanced' in addition to the above. The model was provided with same input as the first and obtained train and test accuracies of 66.2% and 48.3% respectively. All the less frequent groups had an increase in F1 score.

The third model were provided with same hyper-parameters as the first. But the input data was vectorized. The train and test scores were 66.4% and 61.8% respectively and there was no improvement in the F1 score for groups.

The fourth model too had the same hyper-parameters as the first. This time the model input was both vectorized and normalized. The training accuracy increased to 93.01% but the test accuracy remained at 63.01%. Shows high overfitting. The F1 scores for the groups were better compared to first model.



**Figure 15: Performances of different Logistic Regression models**

### 8.1.2 Support Vector Machine

The first model was an SVC with the hyper-parameters kernel='rbf', probability = True. Train and test scores 78.97% and 59.4% respectively. The F1 score was zero for most groups/classes.

The second model was an SVC similar to above but with an addition of balance weights i.e., class\_weight='balanced'. Train and test scores were 63.3% and 41.2% respectively. There was an improvement in F1 scores for the groups compared to above model.

### 8.1.3 Stochastic Gradient Descent

The first model in SGD was an SGDClassifier with the hyper-parameters loss='modified\_huber', penalty='l2', alpha=1e-3, random\_state=42, max\_iter=100, tol=None. Train and test accuracies were 89.2% and 66.98% respectively. F1 scores were zero for some groups, most probably the less significant groups.

The second model in SGD had same parameters as above and an addition of class\_weight='balanced'. Train and test accuracies reduced to 80.66% and 54.89% respectively. But there was an improvement in the F1 scores for individual groups.

### 8.1.4 Multinomial Naïve Bayes

A simple MultinomialNB model was created without specifying any hyper-parameters. The train and test accuracies were 70.26% and 60.46% respectively. F1 score was zero for less significant groups.

### 8.1.5 K Nearest Neighbor

A simple KNeighborsClassifier model was created without specifying any hyper-parameters. The train and test accuracies were 61.18% and 57.32% respectively. F1 score was zero for less significant groups.

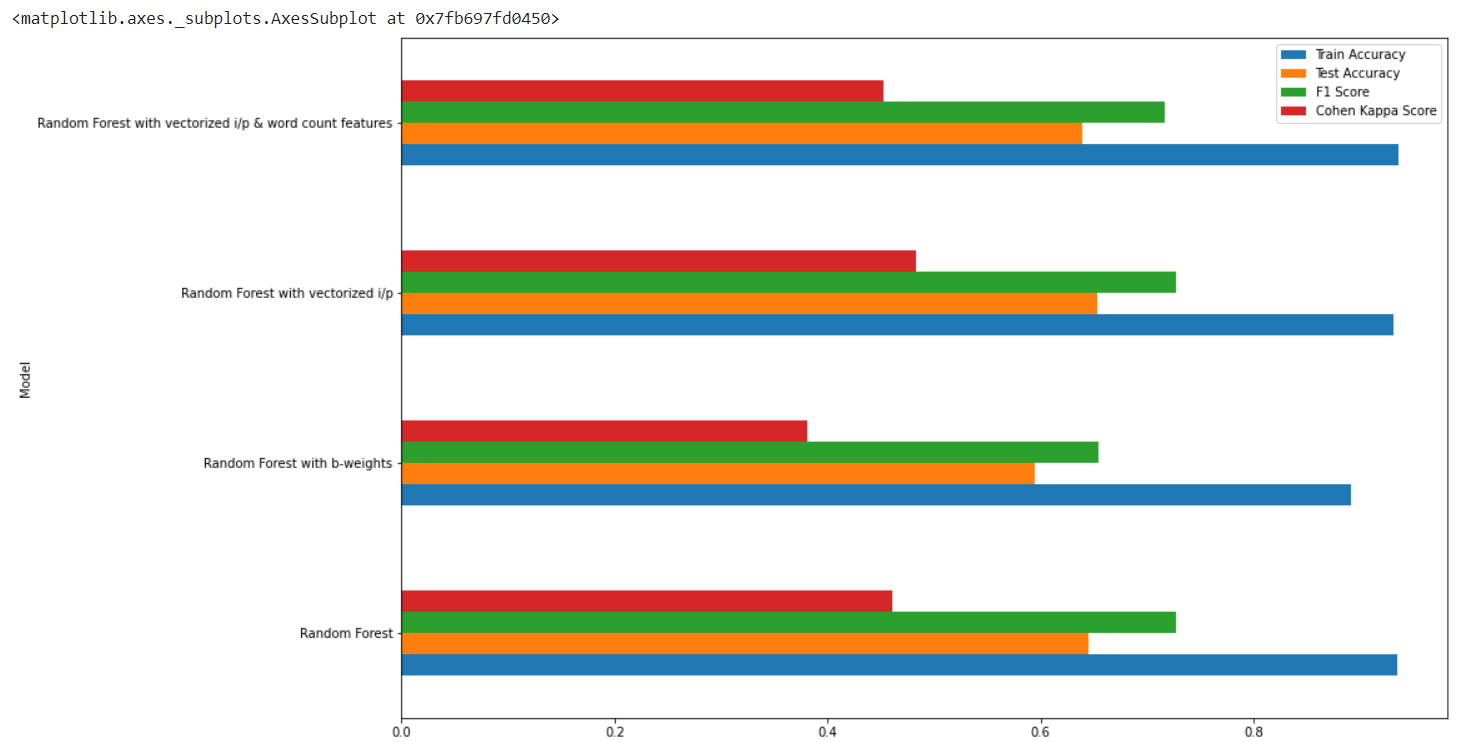
### 8.1.6 Random Forest

A RandomForestClassifier model was created with n\_estimators = 100. The train and test accuracies were 93.49% and 64.49% respectively. F1 score was zero for less significant groups. Significant difference in training and test accuracies indicate overfitting of the model.

A second model was created with n\_estimators=100 and class\_weight = 'balanced'. The train and test accuracies decreased to 89.17% and 59.39%. But there was some improvement in the F1 scores in less significant groups.

A third model was created having hyper-parameters same as the first model. But the input data was vectorized. There was a very slight reduction in the train and test accuracies which were 93.11% and 65.32% respectively. The F1 scores improved for more significant groups but remained zero for the less significant groups.

The fourth model had the same hyper-parameters and vectorized and normalized inputs. The train and test accuracies were similar to the third model. The same was observed for F1 score.



**Figure 16: Performances of different Random Forest models**

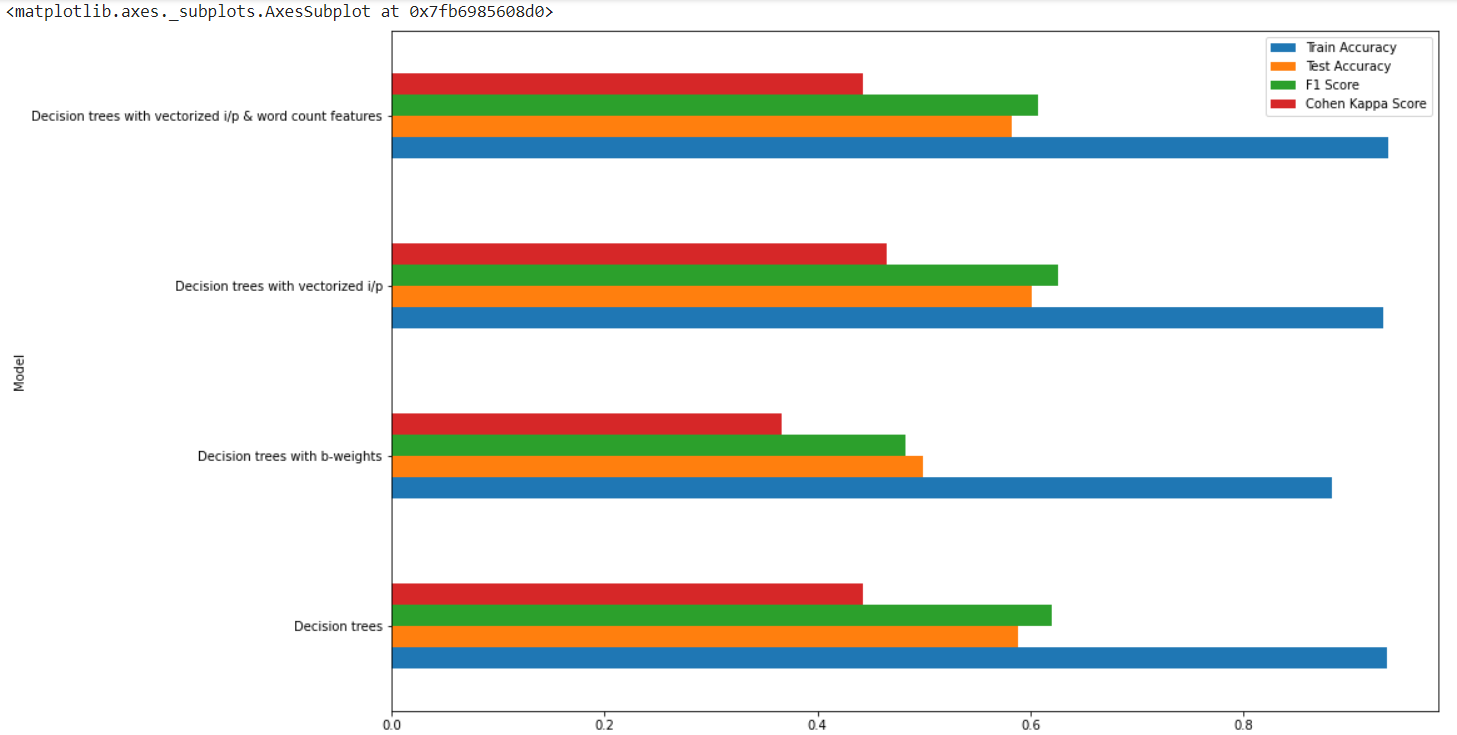
### 8.1.7 Decision Tree

A simple DecisionTreeClassifier was created without providing any hyper-parameters. The train and test accuracies were 93.49% and 58.86% respectively indicating overfitting. F1 score was zero for less significant groups.

A second model was created with class\_weight = 'balanced'. The train and test accuracies decreased to 88.27% and 49.85%. But there was some improvement in the F1 scores in less significant groups.

A third model was created which was similar to the first model. But the input data was vectorized. There train and test accuracies were 93.56% and 58.26% respectively which is very similar to the first model. The F1 scores were zero for more groups compared to the first model.

The fourth model had vectorized and normalized inputs. The train and test accuracies were similar to the third model. The same was observed for F1 score.



**Figure 17: Performances of different Decision Tree models**

### 8.1.8 XGBoost

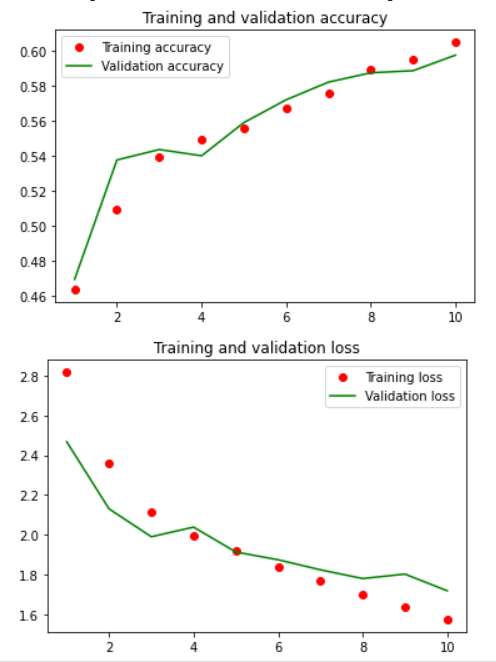
An XGBClassifier was created with hyper-parameters as max\_depth=7, n\_estimators=200, colsample\_bytree=0.8, subsample=0.8, nthread=10 and learning\_rate=0.1. The train and test accuracies were 88.71% and 63.95% respectively. F1 score was zero for some less frequent groups.

## 8.2 Deep Learning

Considering the nature of data and the complexity, we have implemented the below two deep learning algorithms:

### 8.2.1 Neural Network

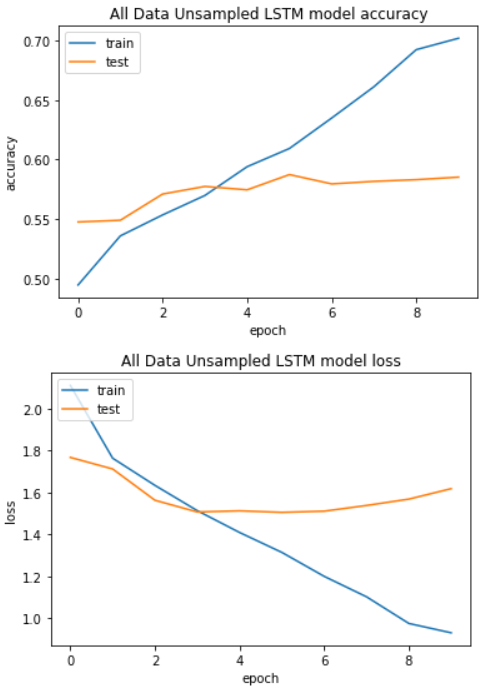
For the neural network, we created a sequential model containing an input layer, 3 hidden dense layers, 3 dropout layers and 1 output layer. We used ‘ReLu’ and ‘Softmax’ as the activation functions, ‘SGD’ as the optimizer and ‘sparse\_categorical\_crossentropy’ as the loss. With vectorized data as the input, the model was run for 10 epochs. The train and test accuracies were 62.9% and 59.75% respectively. This is less compared to some traditional ML models.



**Figure 18: Comparison b/w train and validation loss and accuracies**

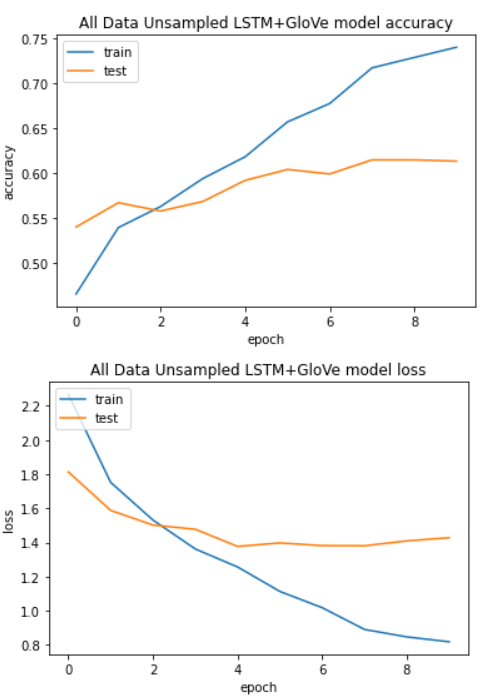
### 8.2.2 Bi-Directional LSTM

For the Bi-Directional LSTM, we performed additional transformations on the data. The description was tokenized and then Word embeddings were created using Word2Vec. The model consisted of an input layer, an embedding layer, a Bidirectional LSTM layer, a dropout layer, a dense layer and an output layer. We used ‘ReLu’ and ‘softmax’ as the activations, ‘Adam’ as the optimizer and ‘sparse\_categorical\_crossentropy’ as the loss function. The train and validation accuracies were 60.92% and 58.73% respectively.



**Figure 19: Comparison b/w train and validation loss and accuracies**

A second model was created similar to the first one. But the word embeddings were created using GloVe vectors. The train and validation accuracies were 71.73% and 61.50% respectively. There’s some improvement on using GloVe embeddings.

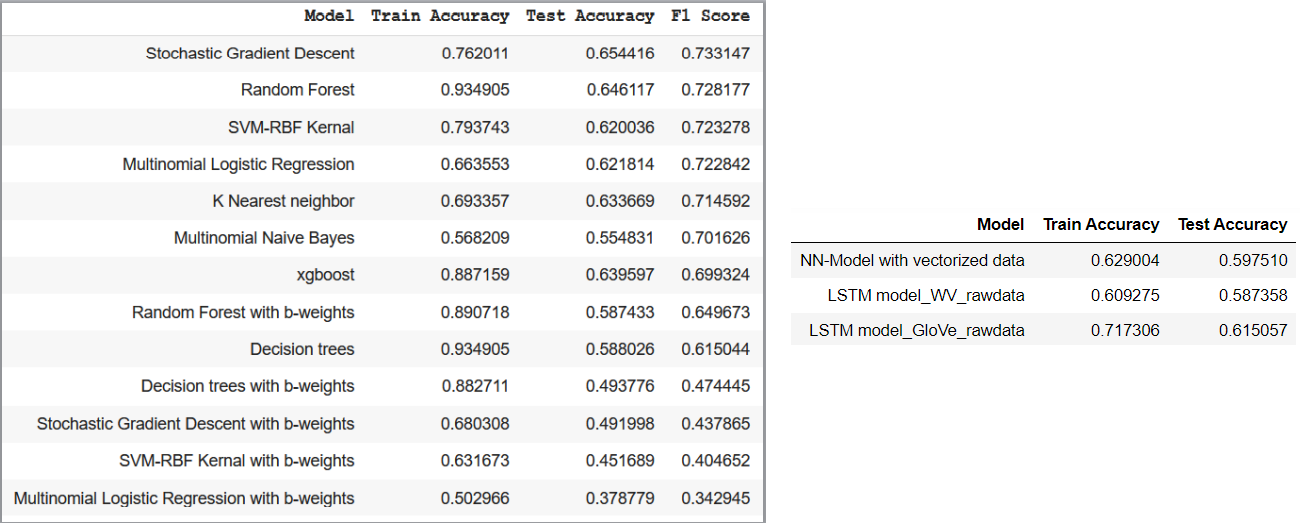


**Figure 20: Comparison b/w train and validation loss and accuracies**

## 8.3 Summary

Most of the models performed moderately in the test set. The accuracy of all the models were in the range of 37-65%. Random forest, XGBoost and Decision Tree seem to be overfitting due to high difference in training and test accuracy. K-NN, XGBoost and Random Forest seem to handle imbalance better than SGD. Meanwhile, SGD, SVM and Naïve Bayes might perform better if the class imbalance is taken care of.

Both the traditional Machine Learning models and Deep Learning models have given similar accuracies of approximately 62%. The below tables list out all the models used and their accuracies.



**Figure 21: Traditional ML vs AI models**

All the models have given pretty good accuracy in the training. But during testing, the accuracies dropped by a considerable amount. There are several reasons for this.

* High class imbalance in the dataset. This is one of the main reasons why all the models gave moderate accuracy. The model was highly biased towards groups that had a high number of tickets.
* Presence of noise in the dataset. Even though we cleaned the text and implemented several feature engineering techniques, there’s a high possibility that noise was fed to the model during training. This led to a low performance on the test set.
* Overfitting on the train set might have led to poor performance in the test set.

# 9. Fine-Tuning

As mentioned above, there are several reasons that can lead to poor model performance. The below two approaches can be used to improve the model performance.

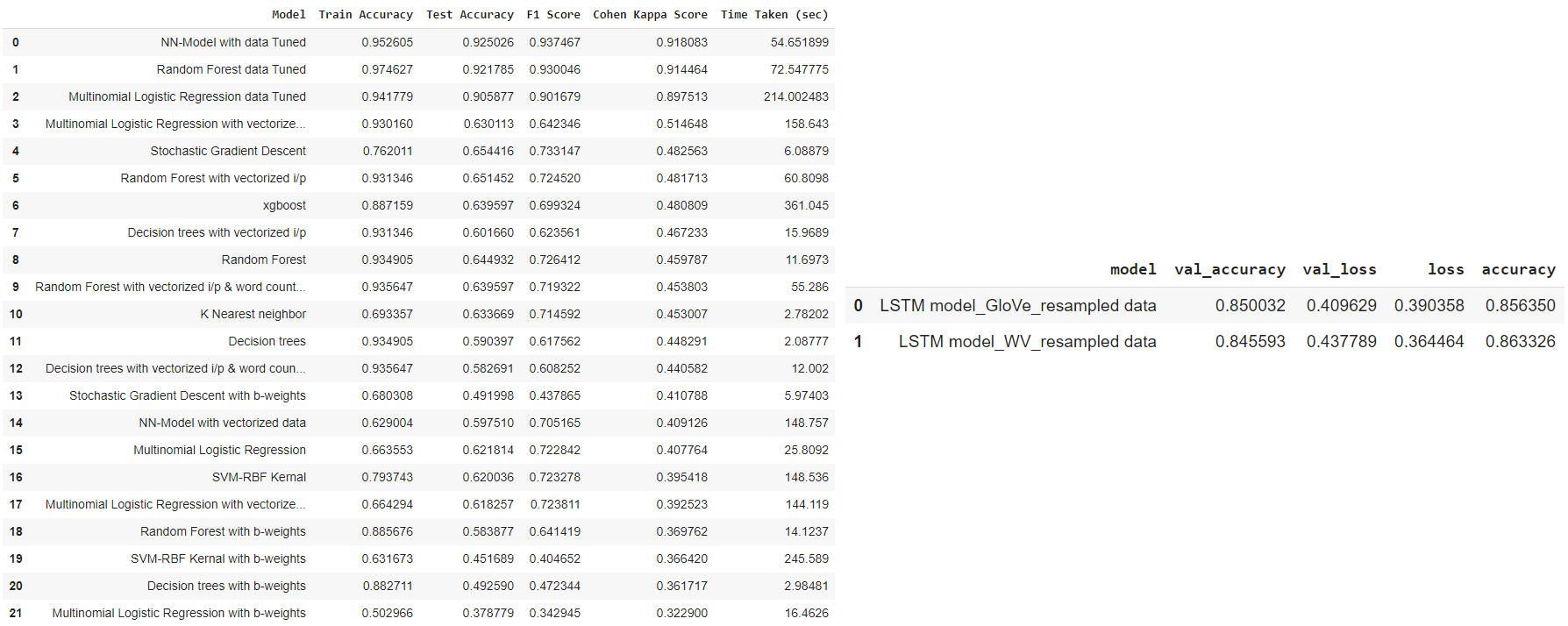
## 9.1 Data Centric Approach

The performance of any model depends on the input that we feed to it. As the saying goes, “Garbage In, Garbage Out”. This means the quality of the output depends on the quality of the input. Having good data is especially important in machine learning and deep learning, which gain greater capabilities over time by analysing large sets of data, learning from them and ultimately making adjustments that make the applications more intelligent.

Even though we performed pre-processing on the dataset, there were several shortcomings which reduced the model performance. One of the main concerns was the high class imbalance. Thus, we used the below steps to address the imbalance:

* Up-sampling – Increase the count of less frequent through SMOTE.
* Down-sampling – Decrease the count of the top frequent groups via down-sampling.
* Use a threshold like removing groups with less than 40 tickets each.
* Merging the less frequent groups into one group.
* Remove or group tickets with foreign languages.

Using the resampled data, we ran all the above-mentioned models again and observed a significant increase in the model accuracies. On comparing various metrics like test accuracy, F1-score and Cohen-Kappa-Score, we found that Neural Network performed the best. Hence, we have selected this as our final model.



**Figure 22: Traditional ML vs AI models with resampled data**

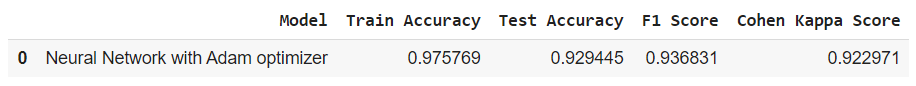
## 9.2 Model Centric Approach

After selecting Neural Network as the final algorithm, we performed hyper parameter tuning to find out the optimal hyper parameters for the model. Cross validation was done using KerasClassifier and GridSearchCV. To use the KerasClassifier wrapper, we built our model in a function which was then passed to the build\_fn argument in the KerasClassifier constructor. In the param\_grid, we passed different values of optimizers and batch size. On running GridSearchCV, we found the model gave the best results with Adam as the optimizer and batch size of 10.

The final model was constructed using the below hyper-parameters:

* Optimizer = Adam
* learning\_rate = 0.001
* batch\_size = 10
* epochs = 10

We received a train accuracy of 97.57% and test accuracy of 92.94%. The data centric approach played a major part in improving the model performance.



**Figure 23: Score for the final model**

# 10. Comparison to Benchmark

The below are few areas which we used as a benchmark to test the quality of the automation:

* Proper Data: We had to ensure that the data used to benchmark machine learning algorithms be representative of the data the model will encounter in production.
* Performance metrics: It is important not to rely solely on metrics such as Accuracy, but rather select a metric which more accurately addresses the true impact of the behaviour of a model in production. Considering the nature of the data and the problem, we considered F1 score to be an important metric in selecting the final model.
* Execution time: We considered runtime performance as an important factor in deciding the model. We were looking for models that took less time and compute power to train and predict.

# 11. User Interface

To create a web app for this project, we selected Streamlit library. Streamlit is an open-source Python library that makes it easy to create and share beautiful, custom web apps for machine learning and data science. A separate python file was created to build the UI. Some of the important functions in this file are:

1. load\_pickle\_models(): This function is used to load several pickle files which are used for the UI. The function loads the final\_model\_pkl.h5 file that contains the pickled model. It then loads the target\_enc\_dict.obj which contains the label encoder dictionary. And finally, it loads the tfidf\_vec.pkl file which contains the TfIdf Vectorizer.
2. clean\_text(): This function converts the input text into lower case and removes all the unwanted characters, words, numbers, etc. It contains several Regex patterns to clean the text.
3. text\_vectorizer(): This function takes all the input from the user and applies several transformations. It uses the clean\_text() function to clean the input text and uses the tfidf\_vec.pkl file to vectorize the cleaned text.
4. model\_prediction(): This function vectorized text returned by the text\_vectorizer() function and provides it as the input to the pickled model. It also uses the target\_enc\_dict.obj object to find out the label from the model prediction.
5. main(): The code for the look and feel of the UI is present here.

On running this python file from the command line, the UI is opened in a web browser. The user can enter the Short Description, Full Description and Caller Name in the input fields provided. The model takes this as the input and predicts the assignment group to which the ticket would be assigned.

# 12. Implications

Artificial intelligence (AI) and machine learning (ML) are turning up seemingly everywhere these days, and the IT support function is no exception. In fact, experts see AI in various forms becoming a key component of the help desk in the years to come. The solution created in this project helps companies in removing the manual overhead associated with high-volume, low-value service desk activities. This solution is similar to an IT automation of repetitive tasks that allow people to be freed up to focus on higher value-add activities.

This solution can be easily used in IT support to classify the incoming issues/tickets. The team could also automate the ticket creation process, which used in combination with our predictive model would automate the whole ticket creation process.

# 13. Limitations

Even though the model predicted assignment groups with 90% accuracy, we still have a lot of shortcomings that needs to be considered. The below are few some limitations:

* Issues/descriptions written in foreign language. Translating the tickets from multiple foreign languages is quite difficult compared to a single foreign language. Hence to improve the accuracy of our model, we only considered the tickets which were written in English. If the incoming ticket is written in a foreign language, it can be routed to a separate team that specifically does the manual translation and assigns the tickets to various groups.
* Due to high class imbalance, we had to group several less frequent groups together which was renamed as Miscellaneous. If the model predicts the group as Miscellaneous, then there would be some manual work for the IT support team. They would have to further look through the groups that were clubbed together and assign it to the right assignment group.
* Mojibakes in the input text might lead to incorrect model predictions. Hence, care must be taken while giving input to the model.

# 14. Conclusion

Overall, the project helped us in creating an automation for the IT support team. It helped us in improving our knowledge in all the areas, right from the EDA to building an UI. We dealt with several new pre-processing and feature engineering techniques. Even though the volume and the complexity of data was quite challenging, it provided us an opportunity to improve our skills in handling the data. This project also helped us to improve our knowledge in deep learning and gave us the confidence to take up more advanced algorithms in our future projects.