

|  |
| --- |
| Final Report  2021 |
| Mentor: Dr. Sandeep Raghuvanshi |
| January 20  Team Members:  Vasundhara Madan, Vimal Pandey,  Pradeep Kumar VP, Raghav Agarwal Apurba Banerjee |

Automatic Ticket Assignment System

Table of Contents

[Introduction 4](#_Toc62081178)

[Problem Statement 5](#_Toc62081179)

[Objective 6](#_Toc62081180)

[EDA (Exploratory Data Analytics) 6](#_Toc62081181)

[Data Findings 6](#_Toc62081182)

[Feature Engineering 7](#_Toc62081183)

[Visual Analysis 8](#_Toc62081184)

[Text Preprocessing 12](#_Toc62081185)

[Modelling 13](#_Toc62081186)

[ML Based Classification models with TF-IDF 13](#_Toc62081187)

[ML Based Classifier with Glove Embeddings 13](#_Toc62081188)

[AI Based Classification Models 14](#_Toc62081189)

[AI Based Models: Architectural Details 14](#_Toc62081190)

[How to Improve Model Performance 15](#_Toc62081191)

[Model Selection and Hyper Tuning 15](#_Toc62081192)

[ML based Classifiers 15](#_Toc62081193)

[Benchmark 16](#_Toc62081194)

[Approach 1: TF-IDF with Bi-grams 16](#_Toc62081195)

[Approach 2: TF-IDF with Tri-grams 16](#_Toc62081196)

[Conclusion 16](#_Toc62081197)

[AI or Deep Learning Based Classifier 17](#_Toc62081198)

[Benchmark 17](#_Toc62081199)

[Hyper-tuning Techniques 18](#_Toc62081200)

[1. Unidirectional LSTM with embedding size 100 18](#_Toc62081201)

[2. Unidirectional LSTM with embedding size 200 19](#_Toc62081202)

[3. Unidirectional LSTM with embedding size 300 21](#_Toc62081203)

[4. Bidirectional LSTM with embedding size 100 22](#_Toc62081204)

[5. Bidirectional LSTM with embedding size 200 24](#_Toc62081205)

[6. Bidirectional LSTM with embedding size 300 25](#_Toc62081206)

[7. Simple LSTM Model 26](#_Toc62081207)

[Final AI Based Model Selected 27](#_Toc62081208)

[Predictions Using the Best Model 29](#_Toc62081209)

[Top 10 groups 29](#_Toc62081210)

[Top 5 groups 31](#_Toc62081211)

[Deployment 33](#_Toc62081212)

[Comparison to Benchmark 35](#_Toc62081213)

[Implications 36](#_Toc62081214)

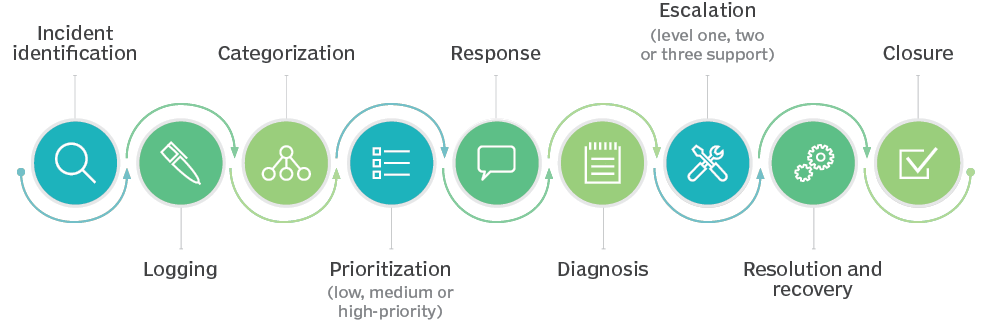
[Conclusion and Future Enhancements 36](#_Toc62081215)

[Closing Reflections 37](#_Toc62081216)

# 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 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.

****

# Problem Statement

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

However, successful closure is not the only thing which matters in customer satisfaction. SLA management of incident must be managed effectively. Generally, manually assigning the incident has some challenges stated below:

* More resource usage and expenses.
* Human errors - Incidents get assigned to incorrect groups
* Delay in assigning the incidents
* More resolution times
* If an incident takes more time in analysis, other productive tasks get affected for the Service Desk

Currently the incidents 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). 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.

During the process of incident assignments by L1 / L2 teams to functional groups, there were multiple instances of incidents getting assigned to wrong functional groups. Around ~25% of Incidents are wrongly assigned to functional teams. Additional effort needed for Functional teams to re-assign to right functional groups. During this process, some of the incidents are in queue and not addressed timely resulting in poor customer service.

# Objective

Guided by powerful AI techniques**,** build a classifier that 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.

# EDA (Exploratory Data Analytics)

EDA is the Data Analysis Process where several techniques are used to better understand the dataset being used. It helps clean up a dataset. It gives you a better understanding of the variables and the relationships between them.

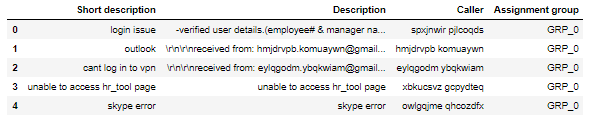
EDA was performed on the given data set. Following were the findings:

## Data Findings

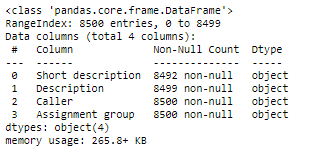
After analyzing the given data set, following observations were drawn:

* The dataset contains 8500 records with 4 attributes i.e. Short Description, Description, Caller and Assignment Group, which is the target column.
* Each column is of data type object.
* There are 7481 unique Short Description types for the given dataset. “Password Reset” is most frequently occurring Short Description.
* There are 74 different Assignment groups. Almost 47% of the tickets are assigned to GRP\_0 and 53% of the tickets are assigned to the remaining 73 assignment groups.
* Imbalanced class data
* There are 9 records with missing values- 8 records with missing/NaN Short description and 1 record with missing/NaN Description.
* There are 83 records with duplicate values.
* The data belongs to 29 different languages. Almost 84% of the records belong to English language.

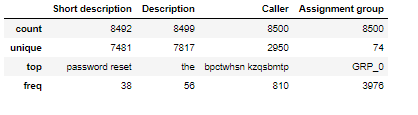
Glimpse of the dataset



Column attribute information



Key Data findings



## Feature Engineering

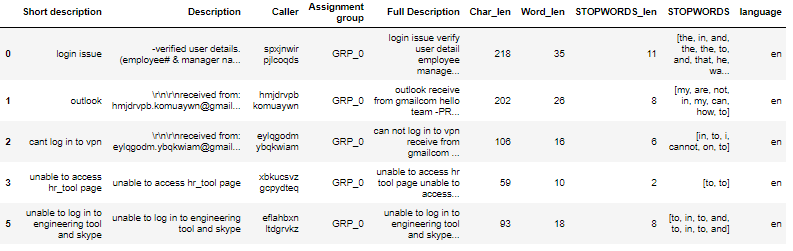
It is the process of creating features for an AI-ML based models from raw text data.

To further analyze other aspects of the data presented, we have employed different methods to analyze text and extract features that can be used to build a classification model to draw note-worthy conclusions.

We created custom columns to study and better understand the data. Following columns were created:

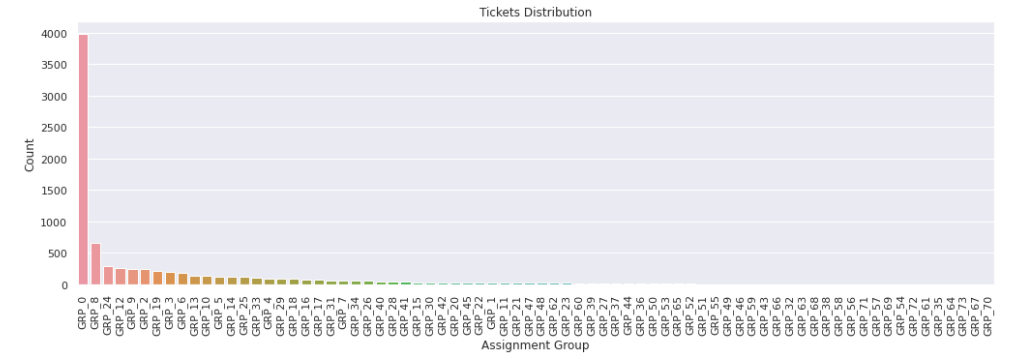
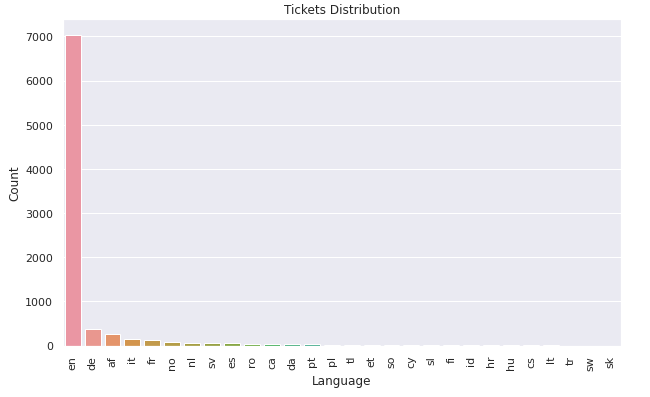
* Created a custom column “Full Description” by concatenating “Short description” and “Description” column.
* Added few custom columns like Char\_len and Word\_len to study the no. of characters and no. of words in Full Description column for every record. The min length of Full Description is 3 whereas the max length for Full Description is 13104.
* Added a ‘language’ column to study to detect the language type for every record. The data belongs to 29 different languages. Almost 84% of the records belong to English language.
* Added a column ‘STOPWORDS’ to study the stop words in Full Description column of every record.

Final data frame:

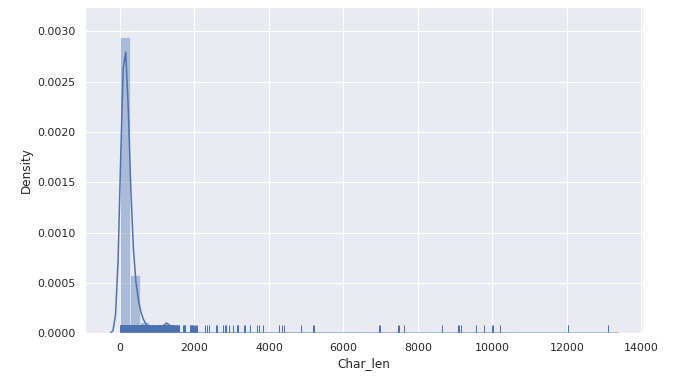


## Visual Analysis

Distribution of Assignment group across the dataset

Distribution of Language across the dataset

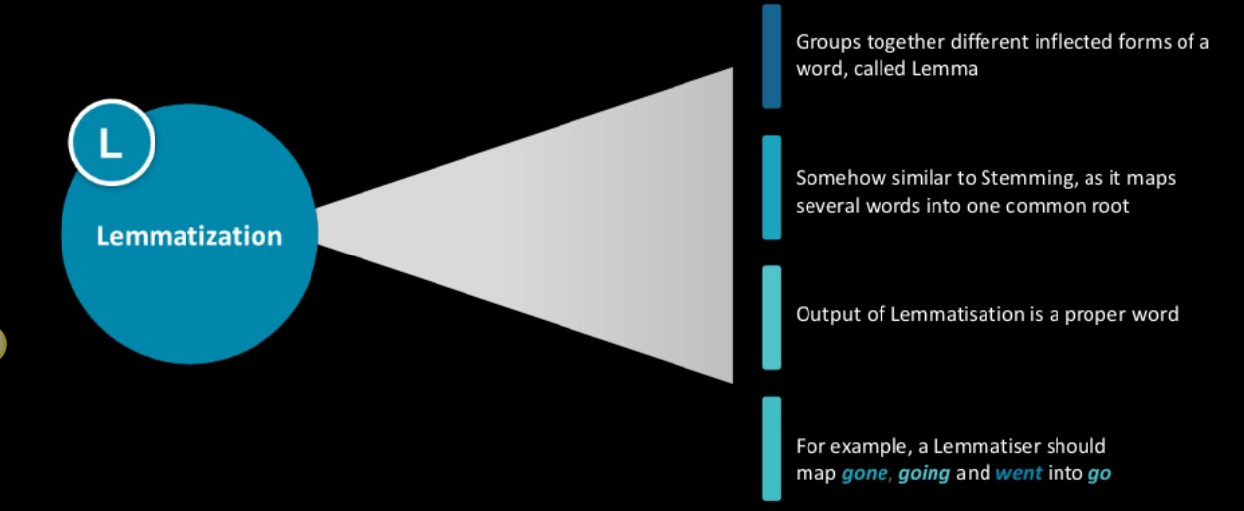
Distribution of Char\_len across the dataset

Word Cloud to analyze important and most frequently occurring words 

# Text Preprocessing

Following text/data pre-processing steps were employed to clean the data before it is presented to models:

* Drop Missing Values: 9 records were found with missing or NaN values- 8 records with missing/NaN Short description and 1 record with missing/NaN Description. These records were dropped.
* Drop Duplicates: 83 records with duplicate values were found that were dropped from the dataset.
* Removed HTML Tags
* Removed Accented Text
* Removed Punctuations
* Remove Special Characters
* Performed Lemmatization



Note: Stemming and Lemmatization both generate the root form of the inflected words. The difference is that stem might not be an actual word whereas, lemma is an actual language word. Stemming follows an algorithm with steps to perform on the words which makes it faster. Whereas, lemmatization, slower than stemming. We chose Lemmatization over Stemming.

# Modelling

## ML Based Classification models with TF-IDF

We have used ML based classification models as well as AI based classification models to assign tickets to functional groups.

* Full Description column i.e. Short description + Description has been vectorized to TF-IDF using TfidfVectorizer using unigrams
* Select top 5000 features using Chi-squared test
* Different ML based classifications models were used, and following accuracies were found

|  |  |  |
| --- | --- | --- |
| Model Type | Training Accuracy | Test Accuracy |
| Naïve Bayes Classifier | 57.05 | 57.53 |
| SVM Classifier | 66.48 | 64.26 |
| Decision Tree Classifier | 63.07 | 60.95 |
| Random Forest Classifier | 95.06 | 64.1 |

Note: Random Forest Classifier model is overfitting

## ML Based Classifier with Glove Embeddings

* Vectorized Full Description column using Glove embedding with embedding size as 300
* Label encoding of the target variable i.e. Assignment group
* Different ML based classification models were used, and following accuracies were found

|  |  |  |
| --- | --- | --- |
| Model Type | Training Accuracy | Test Accuracy |
| SVM Classifier | 89.25 | 66.59 |
| Decision Tree Classifier | 62.27 | 55.87 |
| Random Forest Classifier | 95.16 | 61.43 |

Note: Using glove embeddings, all the models are overfitting

## AI Based Classification Models

* Vectorised Full Description column using Glove embedding with embedding size as 300
* Label encoding of the target variable i.e. Assignment group
* Different AI based classification models were used and following accuracies were found

|  |  |  |
| --- | --- | --- |
| Model Type | Training Accuracy | Test Accuracy |
| Simple Dense NN | 53.46 | 49.21 |
| Convolutional NN | 63.57 | 58.32 |
| RNN based LSTM | 60.79 | 55.94 |

Observations:

* All the AI based model are overfitting.
* Hyper tuning of the models in required to further improve the performance all the while ensuring that the model doesn’t overfit

## AI Based Models: Architectural Details

Simple Dense NN

* Input Dense Layer wit 512 neurons/nodes and ‘he\_normal’ initializer
* 3 hidden layers with 512 neurons in each layer, relu activation
* Every hidden layer is followed by Batch Normalization layer and Dropout layer with dropout ratio of 0.3
* Output layer with activation function softmax

Convolutional NN

* Input layer with max length as 200
* Embedding layer with embedding size as 300
* 5 layers of (Conv1D+MaxPool1D) with kernel size in each Conv1D layer as 2,3,4,5,6 and nodes=128, activation: relu
* 2 layers of (Conv1D+dropout+Batch Norm+MaxPool1D) with 128 nodes in each Conv1D, kernel size 5, activation: relu, dropout ratio 0.5,
* Next flatten followed by dense layer of 1024 and 512 resp.
* Output layer with 74 nodes and softmax activation

RNN Based LSTM

* Embedding layer with max length: 200, dense embedding size 300
* Bidirectional LSTM Layer with 256 units
* 3 units of (Dense + Activation + Dropout) layers
* Every dense layer has 512 nodes
* Activation function used is ReLu
* Dropout layer with a dropout ratio of 0.5 is used
* Output dense layer with 74 units along with softmax activation is used

# How to Improve Model Performance

To improve model performance, we will try out following:

* Try glove embeddings with size 50, 100, 200 and 300
* Try to learn embeddings instead of using glove embeddings
* For LSTM Model, try unidirectional as well as bidirectional LSTM
* For LSTM Model, try different no. of LSTM layers. Try with different no. of LSTM units like 64, 128, 256 etc. in each layer of the LSTM architecture
* Try different optimizers like Adam, RMS Prop by varying the learning rate
* Try different activation functions like ReLu, P-ReLu, Leaky ReLu etc
* Try working with data belonging to top 5 and top 10 groups to address the class imbalance issue
* Try including Caller information in the input dataset.

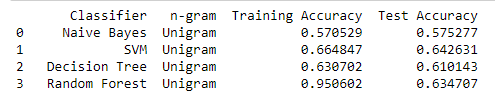
# Model Selection and Hyper Tuning

## ML based Classifiers

We tried different approaches to optimize the model performance i.e. improving the accuracy of the model all the while ensuring that it doesn’t overfit.

### Benchmark

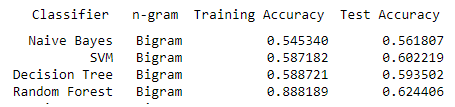
We started out with 4 ML based classifiers i.e. Naïve Bayes Classifier, SVM, Decision Tree, Random Forest Classifier using TF-IDF with unigram as vectorizer. Following accuracies were observed.



Random Forest Classifier Model clearly overfits

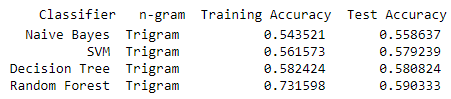
### Approach 1: TF-IDF with Bi-grams

Here we analyzed how the performance of the above models changed when we used bi-gram TF-IDF. Following accuracies were observed:



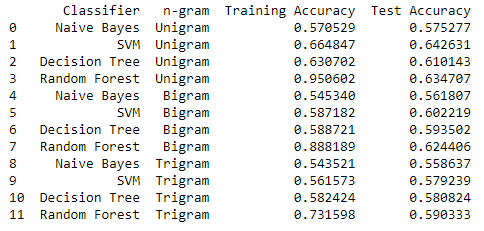
### Approach 2: TF-IDF with Tri-grams

Here we analyzed how the performance of the above models changed when we used tri-gram TF-IDF. Following accuracies were observed:



### Conclusion

Comparing the ML based classifier accuracies w.r.t TF-IDF using uni-gram, bi-gram and tri-gram, it can be seen that SVM model performs best with TF-IDF using uni-gram.



Final Model Selected: SVM with TF-IDF (unigram) vectorizer

Accuracy: 64.26%

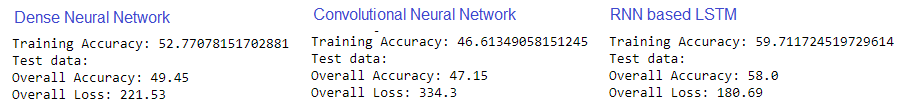
## AI or Deep Learning Based Classifier

We started out with a dense neural network, a convolutional network and an RNN based LSTM Model. We tried to train these models on 2 types of datasets- Data with Caller info and Data without Caller info.

### Benchmark

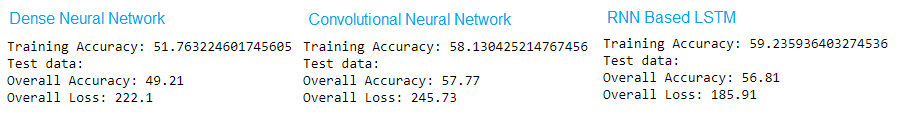
Data with Caller info

Following model accuracies were found:



Data without Caller info

Following model accuracies were found:



It was found that including the caller information in the dataset did not improve the accuracy. So, for further analysis, the dataset w/o the caller information was used.

Also, looking at the accuracies of the above AI based models, RNN based LSTM model seems to be performing best for both dataset types. However, the model seems to be overfitting in case of dataset w/o caller information. This would require hyper tuning to further improve the performance.

Dataset: Without Caller information

Model Selected: RNN Based LSTM

Benchmark Accuracy: 56.81%

## Hyper-tuning Techniques

LSTM Model

### Unidirectional LSTM with embedding size 100

Architectural Details:

-Embedding layer of dense embedding size:100

-2 layers of unidirectional LSTM with 128/256 nodes in each layer

-Dense layer with 128 neurons, activation: relu

-Dropout layer with dropout ratio 0.5

-Dense layer with 128 neurons, activation: relu

-Dropout layer with dropout ratio 0.5

-Dense layer with 64 neurons, activation: relu

-Dropout layer with dropout ratio 0.2

-Output layer with 74 neurons and softmax activation function

LSTM Units in LSTM Layer:

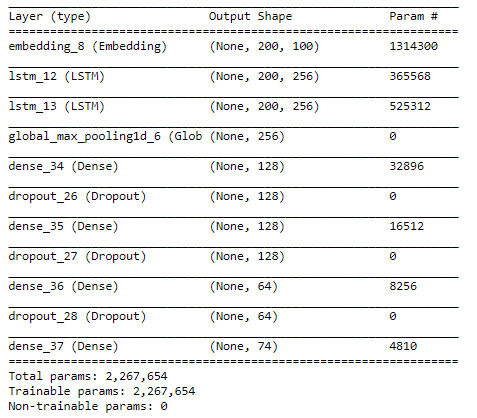
-128, 256

Optimizers used:

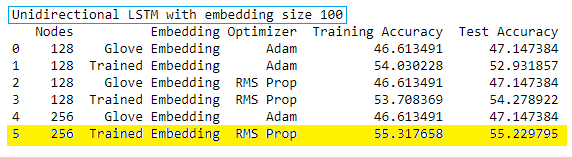
-Adam and RMS Prop

Embedding used:

-Glove Embeddings and Trained Embeddings



After fitting and compiling the variants of unidirectional LSTM Model with embedding size 100, following accuracies were found:



### Unidirectional LSTM with embedding size 200

Architectural Details:

-Embedding layer of dense embedding size:200

-2 layers of unidirectional LSTM with 128/256 nodes in each layer

-Dense layer with 128 neurons, activation: relu

-Dropout layer with dropout ratio 0.5

-Dense layer with 128 neurons, activation: relu

-Dropout layer with dropout ratio 0.5

-Dense layer with 64 neurons, activation: relu

-Dropout layer with dropout ratio 0.2

-Output layer with 74 neurons and softmax activation function

LSTM Units in LSTM Layer:

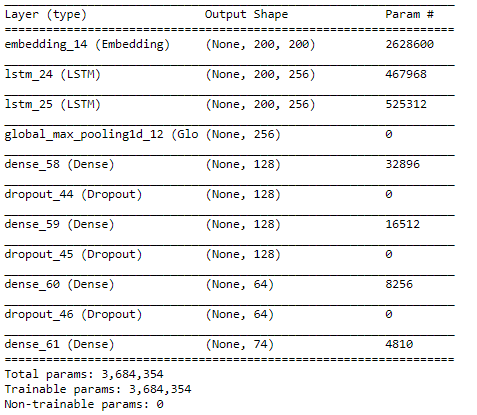
-128, 256

Optimizers used:

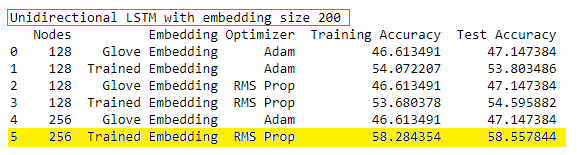
-Adam and RMS Prop

Embedding used:

-Glove Embeddings and Trained Embeddings



After fitting and compiling the variants of unidirectional LSTM Model with embedding size 200, following accuracies were found:



### Unidirectional LSTM with embedding size 300

Architectural Details:

-Embedding layer of dense embedding size:300

-2 layers of unidirectional LSTM with 128/256 nodes in each layer

-Dense layer with 128 neurons, activation: relu

-Dropout layer with dropout ratio 0.5

-Dense layer with 128 neurons, activation: relu

-Dropout layer with dropout ratio 0.5

-Dense layer with 64 neurons, activation: relu

-Dropout layer with dropout ratio 0.2

-Output layer with 74 neurons and softmax activation function

LSTM Units in LSTM Layer:

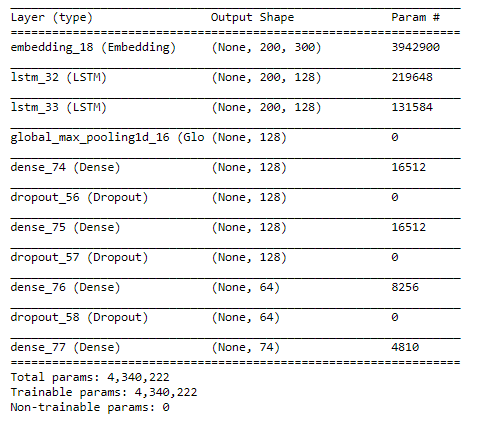
-128, 256

Optimizers used:

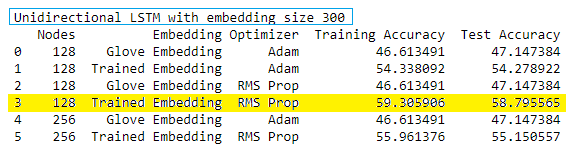
-Adam and RMS Prop

Embedding used:

-Glove Embeddings and Trained Embeddings



After fitting and compiling the variants of unidirectional LSTM Model with embedding size 300, following accuracies were found:



### Bidirectional LSTM with embedding size 100

Architectural Details:

-Embedding layer of dense embedding size:100

-2 layers of bidirectional LSTM with 128/256 nodes in each layer

-Dense layer with 128 neurons, activation: P-Relu

-Dropout layer with dropout ratio 0.5

-Dense layer with 64 neurons, activation: P-Relu

-Dropout layer with dropout ratio 0.2

-Output layer with 74 neurons and softmax activation function

LSTM Units in LSTM Layer:

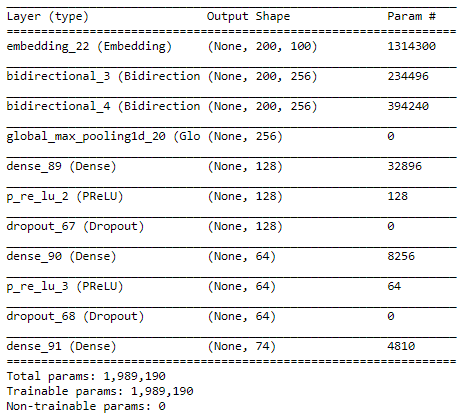
-128, 256

Optimizers used:

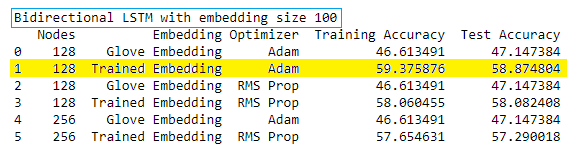
-Adam and RMS Prop

Embedding used:

-Glove Embeddings and Trained Embeddings



After fitting and compiling the variants of bidirectional LSTM Model with embedding size 100, following accuracies were found:



### Bidirectional LSTM with embedding size 200

Architectural Details:

-Embedding layer of dense embedding size:200

-2 layers of bidirectional LSTM with 128/256 nodes in each layer

-Dense layer with 128 neurons, activation: P-Relu

-Dropout layer with dropout ratio 0.5

-Dense layer with 64 neurons, activation: P-Relu

-Dropout layer with dropout ratio 0.2

-Output layer with 74 neurons and softmax activation function

LSTM Units in LSTM Layer:

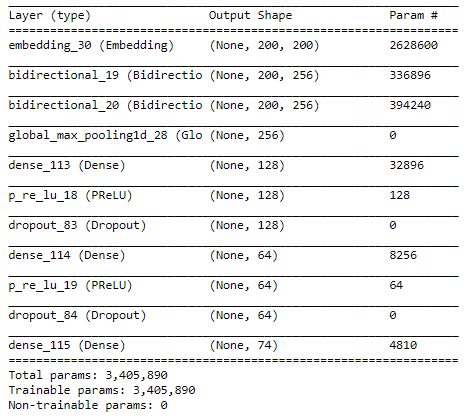
-128, 256

Optimizers used:

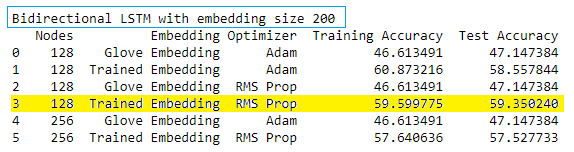
-Adam and RMS Prop

Embedding used:

-Glove Embeddings and Trained Embeddings



After fitting and compiling the variants of bidirectional LSTM Model with embedding size 200, following accuracies were found:



### Bidirectional LSTM with embedding size 300

Architectural Details:

-Embedding layer of dense embedding size:300

-2 layers of bidirectional LSTM with 128/256 nodes in each layer

-Dense layer with 128 neurons, activation: P-Relu

-Dropout layer with dropout ratio 0.5

-Dense layer with 64 neurons, activation: P-Relu

-Dropout layer with dropout ratio 0.2

-Output layer with 74 neurons and softmax activation function

LSTM Units in LSTM Layer:

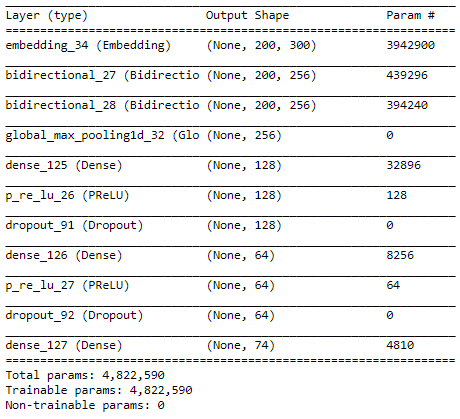
-128, 256

Optimizers used:

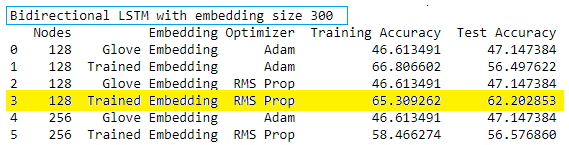
-Adam and RMS Prop

Embedding used:

-Glove Embeddings and Trained Embeddings



After fitting and compiling the variants of bidirectional LSTM Model with embedding size 300, following accuracies were found:



### Simple LSTM Model

Let’s try out a very simple RNN based LSTM Model instead of using a complex architecture and see how the model performs.

Architecture Details:

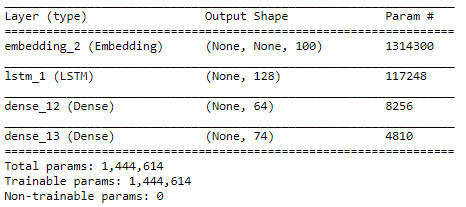
* Embedding layer with trained embeddings of size 100
* Unidirectional LSTM Layer with 128 nodes
* Dropout ratio: 0.3 and Recurrent Dropout:0.2
* Dense layer with 64 units and relu activation function
* Output layer with 74 units and softmax activation function

Optimizers used:

* Adam with learning rate 0.001

Embedding used:

* Trained Embeddings size 100



Observations:

1. Trained Embedding seems to give better model accuracy as compared to Glove Embeddings
2. 128 LSTM units in LSTM layer give better accuracy
3. In each case, we observed that RMS Prop optimizer gave a better accuracy as compared to Adam optimizer

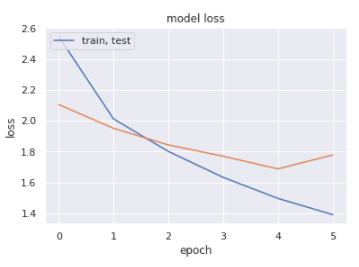
## Final AI Based Model Selected

* Bidirectional LSTM with trained embeddings of size 300 with RMS Prop Optimizer
* Accuracy: 62.2% which is an improvement compared to the LSTM model we started out with i.e. 56.81%.
* Saved the best performing model to ‘best\_model.h5’ file and the corresponding weights to ‘best\_weights.h5’ for future reference.
* After performing hyper-tuning the LSTM model, we were able to improve the accuracy by around 5.39%

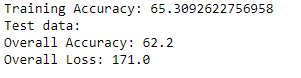
Train Vs. Validation Accuracy



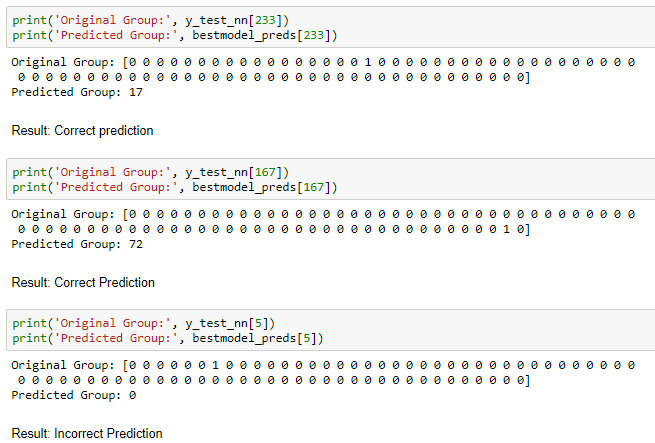
Train Vs. Validation Loss



After fitting and compiling the above model, following accuracy was observed:



## Predictions Using the Best Model

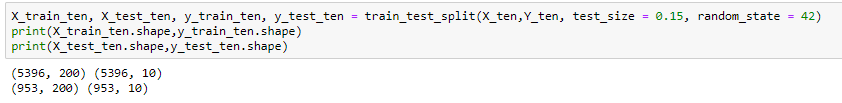


## Top 10 groups

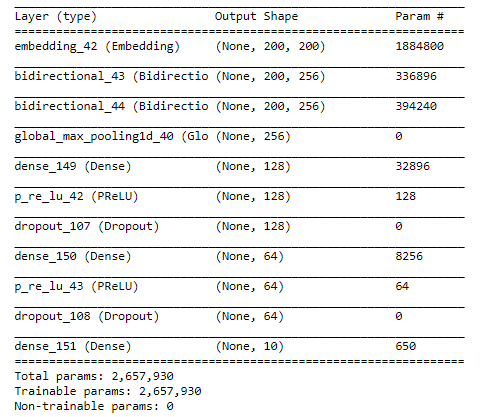
Since the dataset provided is highly imbalanced. Almost 47% of the tickets are assigned to GRP\_0 and the remaining 53% tickets were distributed or assigned to remaining 73 functional groups. This imbalanced class data causes the model to perform with only 60% accuracy. Thus, we decided to have subset of the data to include records that only belong to top 10 groups.

Train-Test Split for Top 10 groups

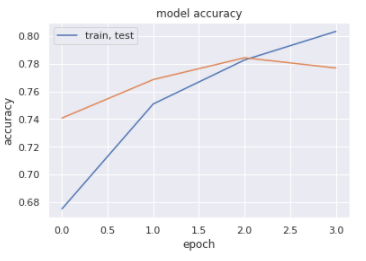
* Dataset was split in 0.85:0.15 ratio to get training and test data
* Total no. of records: 6349
* Total no. of training records: 5396
* Total no. of test records: 953



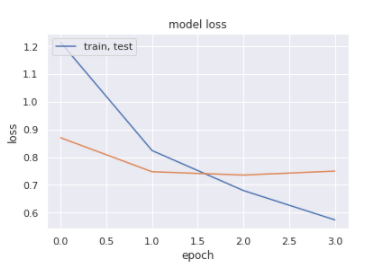
Model used: Bidirectional LSTM Model with embedding size 200



Train Vs. Validation Accuracy



Train Vs. Validation Loss



After fitting and compiling the above model, following accuracy was observed:



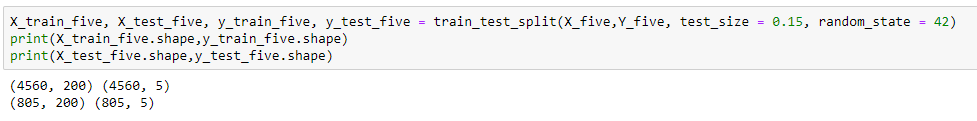
Accuracy: 74.29%

## Top 5 groups

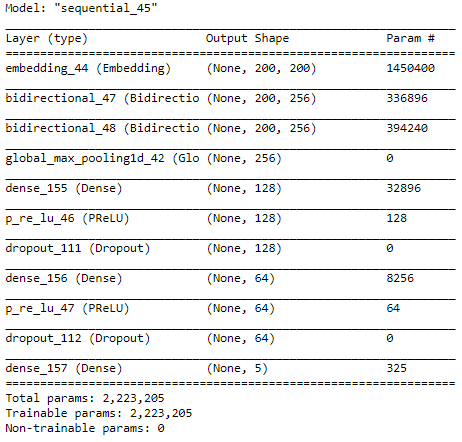
We tried to further filter the original dataset to get records belonging to top 5 functional groups and see it the accuracy was further improved or not.

Train-Test Split for Top 10 groups

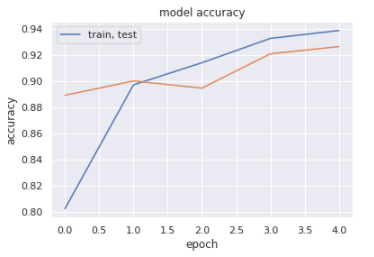
* Dataset was split in 0.85:0.15 ratio to get training and test data
* Total no. of records: 5365
* Total no. of training records: 4560
* Total no. of test records: 805



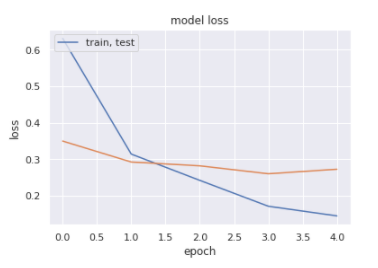
Model used: Bidirectional LSTM Model with embedding size 200



Train Vs. Validation Accuracy



Train Vs. Validation Loss



After fitting and compiling the above model, following accuracy was observed:



Accuracy: 89.57%

# Deployment

Pickling

We used python’s pickle module to serialize the best model object and created a pickle file i.e. “auto\_ticket\_assignment.pkl”.

*“The pickle module implements binary protocols for serializing and de-serializing a Python object structure. “Pickling” is the process whereby a Python object hierarchy is converted into a byte stream, and “unpickling” is the inverse operation, whereby a byte stream (from a binary file or bytes-like object) is converted back into an object hierarchy.”*

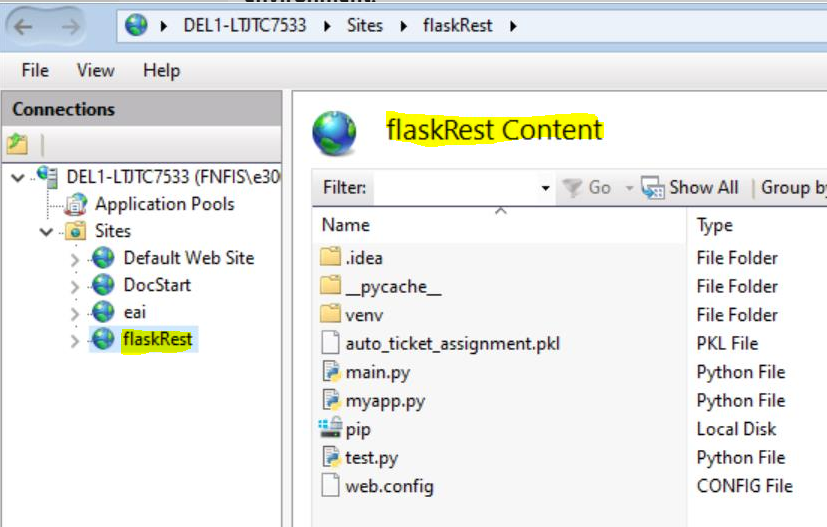
Flask- RESTful

Next, we used Flask to create a RESTful API which loads the best fit model object from the pickle file to predict the group.

*“Flask-RESTful is an extension for Flask that adds support for quickly building REST APIs.”*

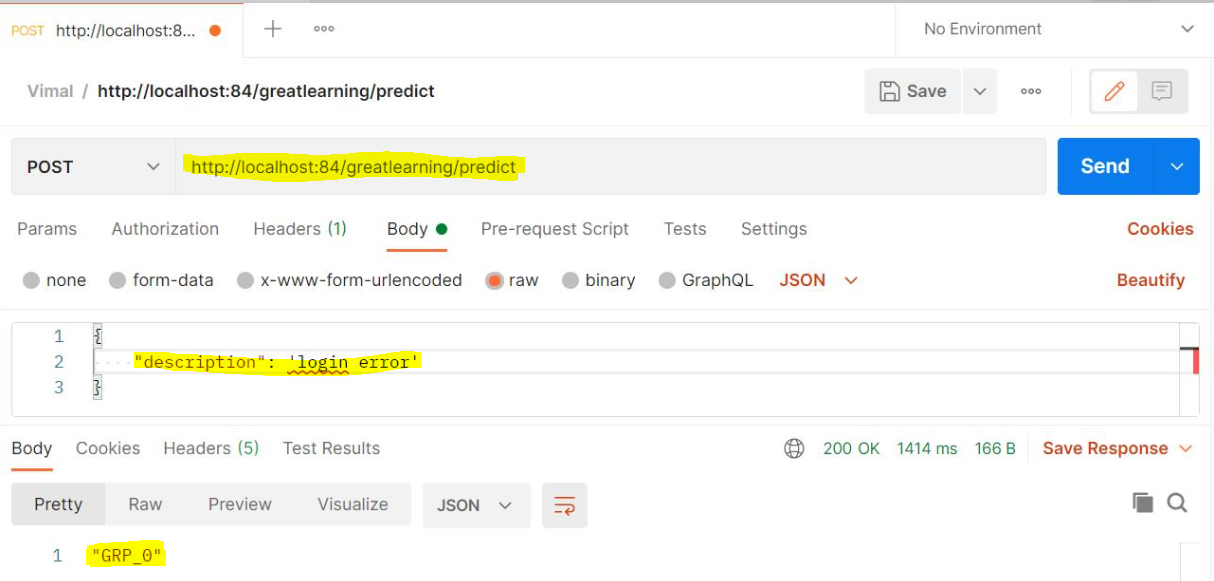
Deploying Flask API on IIS

* Installed the CGI feature
* Installed wfastcgi package on the python environment.
* Enabled wfastcgi and configured Fast CGI Settings in IIS
* Setting up the flask API on IIS.
* Setting up the handler mappings for the Flask API.



Testing Flask API through Postman

We used Postman application to test Flask API which predicts the group. The API takes ticket description as the input and gives group as the output.



# Comparison to Benchmark

We started out with RNN Based LSTM Model with an accuracy of 56.81% which was our benchmark. However, after employing hyper tuning techniques and varying the model architecture, optimizer etc. we were able to achieve an accuracy of 62.2%

We were able to improve the accuracy on the benchmark by 5.39%.

We hoped to increase the model accuracy up to 80% with the original dataset. However, since the original dataset was highly imbalanced, we could only improve the accuracy to 62.2% and 64.26% with AI-Based Classifier and ML-Based Classifier respectively.

To prove our hypothesis, we later tried to create 2 different datasets - one with top 10 groups and other with top 5 groups. We tried to fit and compile the model created for original dataset on top 10 and top 5 groups dataset and achieved an accuracy of ~75% and ~90%. This proves our assumption that the reason for low accuracy on original dataset is because of data imbalance.

# Implications

1. With manual ticket assignment in Incident Management Process, 1 FTE spends around 15 mins on SOP review for each incident before assigning it to a functional group.
2. Guided by AI-ML techniques, we have built a classifier that allows organizations to automatically assign tickets to correct group instantaneously.
3. By using this AI-ML based automated ticket assignment mechanism, organizations can achieve following:
   * FTE Cost Saving
   * Time saving
   * Better support staff allocation
   * Improve Customer Service experience by reducing the no. of queued incident tickets
   * Better SLA Management
   * Overall improved turnaround time for a request/incident

# Conclusion and Future Enhancements

1. The original dataset presented to us had a total of 74 functional groups to which tickets were assigned. However, ~47% of the tickets belonged to just one group i.e. GRP\_0. 53% tickets belonged to remaining 73 groups. This shows data presented to us was highly imbalanced.
2. For many functional groups in the dataset, there was very little training data for the models.
3. As a future enhancement, we can use text generative model to get more set of training examples which would potentially lead to better model performance.
4. The original dataset presented to us is multilingual. The embeddings used in this project are English word embeddings. We can even use non-English pre-trained word embeddings.
5. As a future enhancement, we can use some other multilingual deep learning-based models like mBert.

# Closing Reflections

While working on this problem statement, we performed many operations and learnt following:

1. Pre-Processing, Data Visualization and EDA

* Exploring the given Data files
* Understanding the structure of data
* Missing points in data
* Finding inconsistencies in the data
* Visualizing different patterns
* Visualizing different text features
* Dealing with missing values
* Text preprocessing
* Creating word vocabulary from the corpus of text data
* Creating tokens as required

1. Model Building

* Building a model architecture which can classify.
* Trying different model architectures by researching state of the art for similar tasks
* Train the model
* 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

1. Test the model and Fine Tuning

* Test the model and report as per evaluation metrics
* Try different models
* Try different evaluation metrics
* Set different hyper parameters, by trying different optimizers, loss functions, epochs, learning rate, batch size, checkpointing, early stopping etc. for these models to fine-tune them
* Report evaluation metrics for these models along with our observation on how changing different hyper parameters leads to change in the final evaluation metric.

1. We learnt how to deploy the model on IIS as Flask RESTful API

* Pickling and Unpickling
* Flask RESTful API
* Deploying Flask API on IIS
* Testing Flask API via. Postman

1. Project Closure

* Prepare final project report to present our findings and solution to the problem statement.

All the project artifacts are available at: [Vimal2308/NLP-Capstone-Project: NLP Capstone Project (github.com)](https://github.com/Vimal2308/NLP-Capstone-Project)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* THANK YOU \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*