greatlearning
Learning for Life

Capstone Project Final Report

On

Automatic Ticket Assignment System

Submitted in fulfillment of

Post Graduate Program
In
Artificial Intelligence and Machine Learning

Submitted By:

Aparna Bhattacharjee
Roumi Ghosh
Jayant Kalaghatagi
Hariharan Balraj

Index

1.Introduction

Categorizing support tickets is the one of the most important features of almost any helpdesk ticketing system available in the market today. When an issue or support ticket comes to any help desk, at first it needs to be processed and assigned with a tag or category so that it can be routed to the correct team member for resolution. This involves reading and analysis of the ticket, so that agents should know which category to choose. However, manual classification systems are often complicated and cluttered with too many categories to choose from. After spending endless hours in going through so many numbers of tickets, agents often end up assigning tickets to wrong category for the sake of completing the work fast and avoid spending time searching for the appropriate one.

Ticket classification with machine learning avoids this problem because machine learning tools only assign a previously defined tag and ticket categorization models assign these tags automatically, without scrolling through a long list of tags. Instead of humans interpreting the content and categorizing it accordingly, automatic ticket classification uses Natural Language Processing, which helps the machines to process, understand, and potentially generate human language in a fast and cost-effective way.

2.Real World Problem and Solution

Let's take a real-world example of Sap support helpdesk wherein almost 30–40% of incident tickets are not routed to the right team and the tickets keep roaming round and round. By the time it reaches the right team, the issue might become widespread or reach the top management inviting a lot of trouble. This problem could be solved easily if there is an automatic mechanism to route the incident ticket to the right category and right team. This helps in saving effort as well as cost and time.

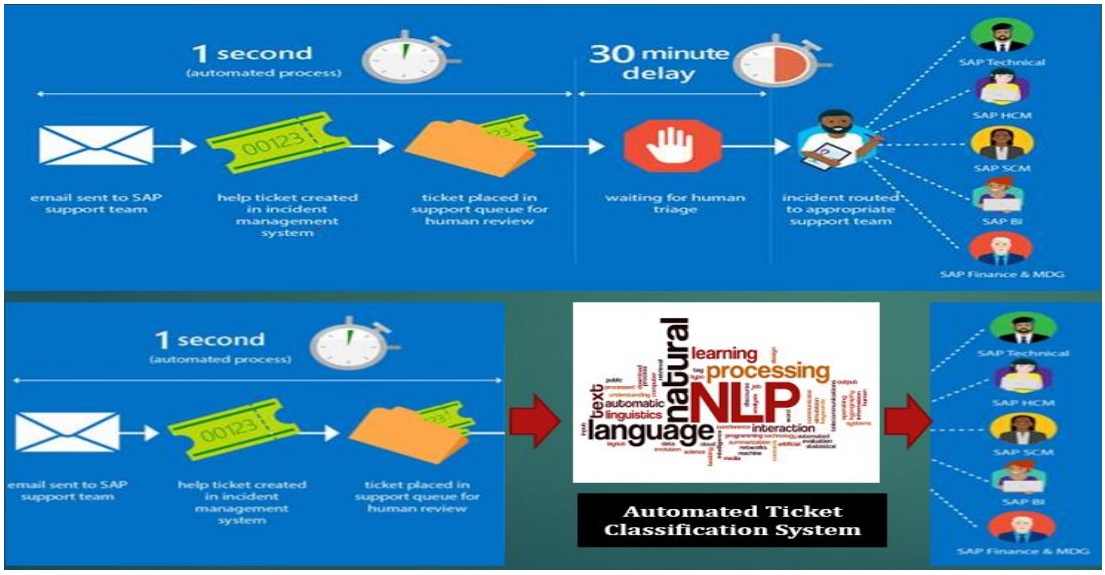


Image 2.1: Help desk ticket assignment system with human interaction and ML approach

3.Problem Statement for Capstone

We have received a data set of a XYZ incident management system where the incidents are raised by the users in form of tickets which can be further assigned to appropriate group for resolution based on the ticket description, short description etc.
Since manual assignment of incidents is time consuming and the amount of human effort and error are more because of misaddressing. It increases the response and resolution times which results in the poor customer services/ satisfaction deterioration.
So, with the help of strong AI and ML techniques we have to classify incidents in the correct functional groups, which would help organizations reduce resolving time of the issue, and also can focus on more productive tasks. This cut short the manual entry or classification of a huge database altogether which is a tedious and error friendly job.

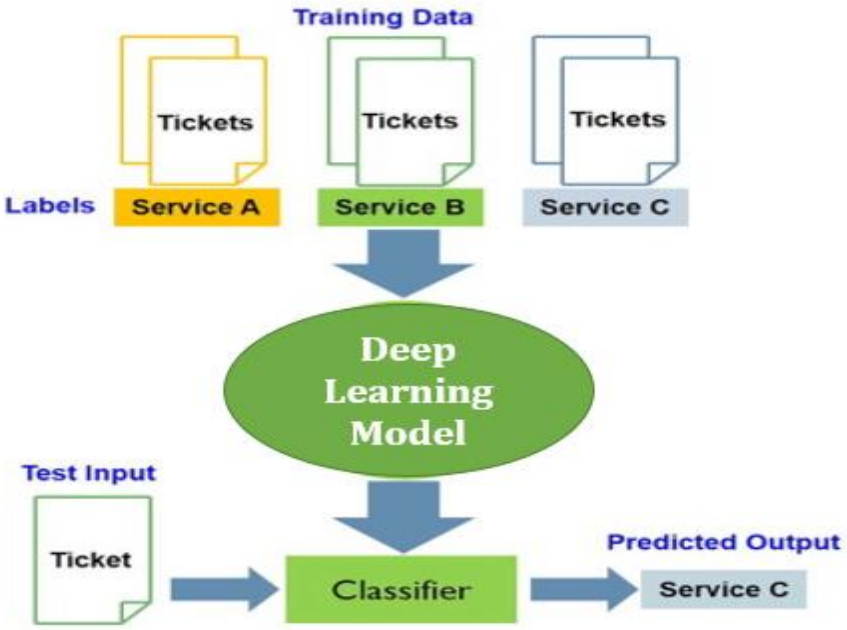


Image 3: Flowchart for Automatic Ticket Assignments using AIML techniques

4.Architecture and Processes

We have incorporated all the steps we will perform to build the best model and predict output in the below architecture diagram.

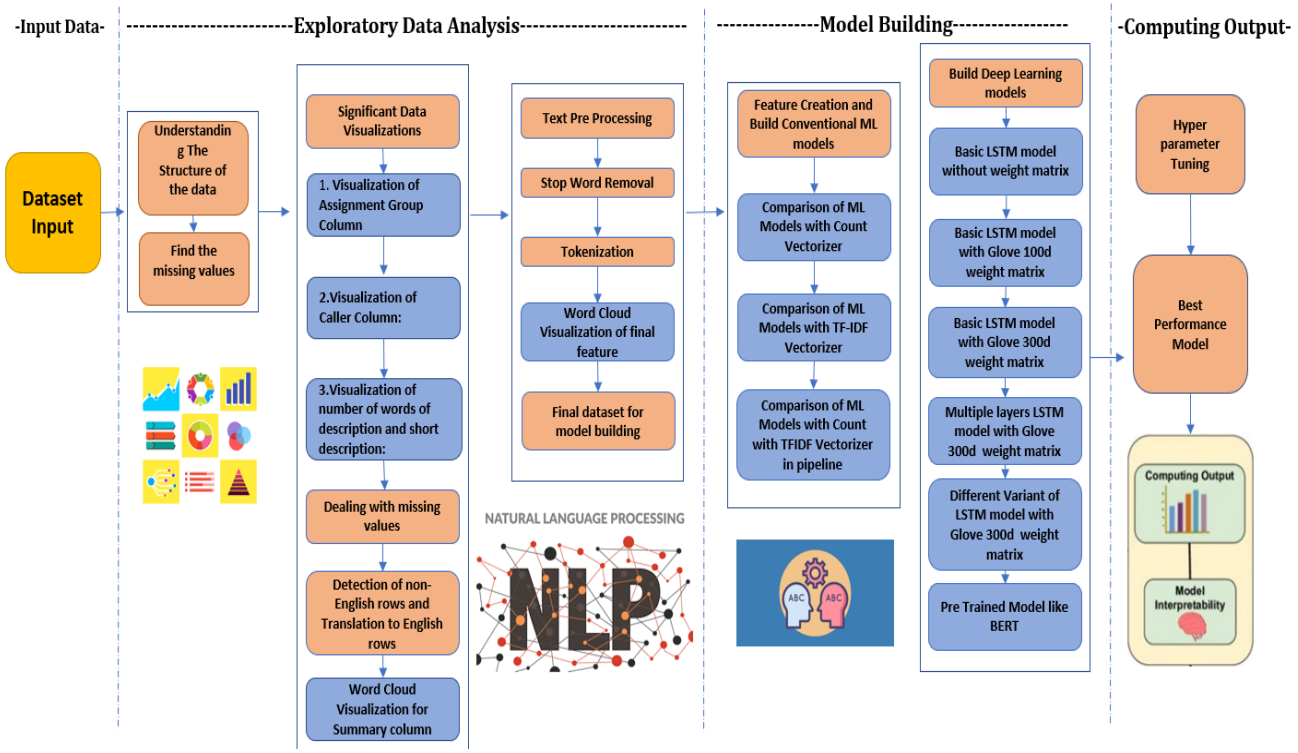


Image 4: Architecture and Process Flow

5.Objectives

The objective of the project is to build a classifier that can classify the tickets to right group by analyzing the text. We will concentrate on the below objectives also for building a good classifier.

- EDA and visualizations are to be done for understanding the importance of each column and to have proper conclusion on the data set.
- The data has to be checked for inconsistencies if any then tackle those in an efficient way.
- Different natural language processing, preprocessing techniques are tried using maximum of our features.
- Different feature extraction techniques and comparison of models.
- How to use different classification models have to be explored.
- How to use transfer learning to use pre-built models have to be explored.
- Learn to set the optimizers, loss functions, epochs, learning rate, batch size, checkpointing, early stopping etc.
- Different research papers of given domain to obtain the knowledge of advanced models for the given problem has to be explored.

6.Implementation

In this section, we present the implementation part, describing the dataset, tools and machine learning methods that are used to produce the final output.

6.1. Data Set details:

We use primary-data to evaluate the performance of the algorithms. The dataset is extracted from the ticketing system of XYZ software developing company.
Details about the data are given in below link:
https://drive.google.com/open?id=1OZNJm81JXucV3HmZroMq6qCT2m7ez7IJ
The data set is provided in .xlsx format. **It has 4 columns and 8500 rows**. **The columns are short description, Description, Caller and Assignment group.**

	A	B	C	D	E
1	Short description	Description	Caller	Assignment group	
2	login issue	-verified user details.(employee# & manager name)-checked t	spxjnwir pjlcoqds	GRP_0	
3	outlook	received from: hmjdrvpb.komuaywn@gmail.comhello team,n	hmjdrvpb komuaywn	GRP_0	
4	cant log in to vpn	received from: eylqgodm.ybqkwiam@gmail.comhii cannot log	eylqgodm ybqkwiam	GRP_0	
5	unable to access hr_tool page	unable to access hr_tool page	xbkucsvz gcpydteq	GRP_0	
6	skype error	skype error	owlgqjme qhcozdfx	GRP_0	
7	unable to log in to engineering tool and skype	unable to log in to engineering tool and skype	eflahbxn ltdgrvkz	GRP_0	
8	event: critical:HostName_221.company.com the value of mo	event: critical:HostName_221.company.com the value of mo	jyoqwxhz clhxsoqy	GRP_1	
9	ticket_no1550391- employment status - new non-employee	ticket_no1550391- employment status - new non-employee	eqzibjhw ymebpoih	GRP_0	
10	unable to disable add ins on outlook	unable to disable add ins on outlook	mdbegvct dbvichlg	GRP_0	
11	ticket update on inplant_874773	ticket update on inplant_874773	fumkcsji sarmtlhy	GRP_0	
12	engineering tool says not connected and unable to submit re	engineering tool says not connected and unable to submit re	badgknqs xwelumfz	GRP_0	
13	hr_tool site not loading page correctly	hr_tool site not loading page correctly	dcqsolkx kmsijcuz	GRP_0	
14	unable to login to hr_tool to sgxqsuojr xwbesorf cards	unable to login to hr_tool to sgxqsuojr xwbesorf cards	oblekmrw qltgvspb	GRP_0	
15	user wants to reset the password	user wants to reset the password	iftldbmu fujslwby	GRP_0	
16	unable to open payslips	unable to open payslips	epwyvjsz najukwho	GRP_0	
17	ticket update on inplant_874743	ticket update on inplant_874743	fumkcsji sarmtlhy	GRP_0	
18	unable to login to company vpn	received from: xyz@company.comhi,i am unable to login to c	chobktqj qdamxfuc	GRP_0	
19	when undocking pc , screen will not come back	when undocking pc , screen will not come back	sigfdwcj reofwzlm	GRP_3	
20	erp SID_34 account locked	erp SID_34 account locked	nqdyowsm yqerwtna	GRP_0	

Image 6.1.1: Glimpse of the dataset in excel file

6.2. Tools

Below libraries, we have used for this problem:

1. Pandas library for data frame operations.
2. For Visualizations: matplotlib, seaborn, plotly and Word Cloud
3. For Language Detection and Translation: Polyglot and Lang detect.
4. For Text Preprocessing: nltk
5. ML conventional model: sklearn
6. Deep learning: TensorFlow and keras
7. BERT: Transformers, BERT Uncased
8. ULM FIT: fastai
9. FastText: fasttext
10. Deployment: Flask API

6.3. Exploratory Data Analysis:

EDA is generally done to understand the dynamics of the dataset given by exploring the values in it. This is always the first step of any problem on the basis of which we chalk out what plan has to be carried out further for good results and outcomes.
As a part of EDA, we have completed the below tasks:

6.3.1. Understanding the structure of data

After importing the excel sheet as the data frame, we see that the data initially looks like the image below:

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

Image6.3.1.1.Imported data from Excel

After using shape, describe and info functions we have the given observations:

```
[ ]  #Data type of each attribute
     df.info()
     #There are total 9 null values
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8500 entries, 0 to 8499
Data columns (total 4 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   Short description  8492 non-null   object
 1   Description        8499 non-null   object
 2   Caller             8500 non-null   object
 3   Assignment group   8500 non-null   object
dtypes: object(4)
memory usage: 265.8+ KB
```

```
[ ]  #5 point summary
     df.describe(include="all")
```

	Short description	Description	Caller	Assignment group
count	8492	8499	8500	8500
unique	7481	7817	2950	74
top	password reset	the	bpctwhsn kzqsbmtp	GRP_0
freq	38	56	810	3976

Image6.3.1.2.Describe and info function

1. It has a total of **8500 rows and 4 columns**. Here we have also inferred that the **Short Description and Description and caller are independent columns and Assignment Group is the dependent variable** which we need to predict.
2. It is a **supervised classification problem** having the target variable already defined. Now, here we have to predict the group column value (target variable) with respect to other independent variables.
3. **In Short description and description column, there are many tickets where only short description is not sufficient to take it as main feature**. Also, there are many tickets for which both the short description and description are same. Different variations are there.
4. There are many rows where **description/short description is written in other languages which are not English**.
5. There are some repetitive rows having same information in all the columns.

6.3.2. Checking of missing data points

It has been checked whether there are any missing values under any existing columns or not. After applying is null on the dataset we have the given results:

```
#Checking the presence of missing values
df.isnull().sum()
```

```
Short description    8
Description          1
Caller               0
Assignment group     0
dtype: int64
```

Image 6.3.2.1: Count of the missing values

	Short description	Description	Caller	Assignment group
2604	NaN	\r\n\r\nreceived from: ohdrnswl.rezuibdt@gmail...	ohdrnswl rezuibdt	GRP_34
3383	NaN	\r\n-connected to the user system using teamvi...	qftpazns fxpnytmk	GRP_0
3906	NaN	-user unable tologin to vpn.\r\n-connected to...	awpcmsey ctdiuqwe	GRP_0
3910	NaN	-user unable tologin to vpn.\r\n-connected to...	rhwsmefo tvphyura	GRP_0
3915	NaN	-user unable tologin to vpn.\r\n-connected to...	hxripljo efzounig	GRP_0
3921	NaN	-user unable tologin to vpn.\r\n-connected to...	cziadygo veiosxby	GRP_0
3924	NaN	name:wvqgbdhm fwchqjor\nlanguage:\nbrowser:mic...	wvqgbdhm fwchqjor	GRP_0
4341	NaN	\r\n\r\nreceived from: eqmuniov.ehxkcbgj@gmail...	eqmuniov ehxkcbgj	GRP_0

	Short description	Description	Caller	Assignment group
4395	i am locked out of skype	NaN	viyglzfo ajtfzpkb	GRP_0

Image 6.3.2.1: Displaying missing values

There are **9 missing** values.**1** in Description and **8** in Short description.

6.3.3. Significant Data Visualizations

Visualization of Assignment Group Column:

The unique counts for each Assignment group is calculated and all the Assignment groups are plotted in descending order. Top 50 Assignment groups are also plotted. Word Cloud has been created for Assignment Group.

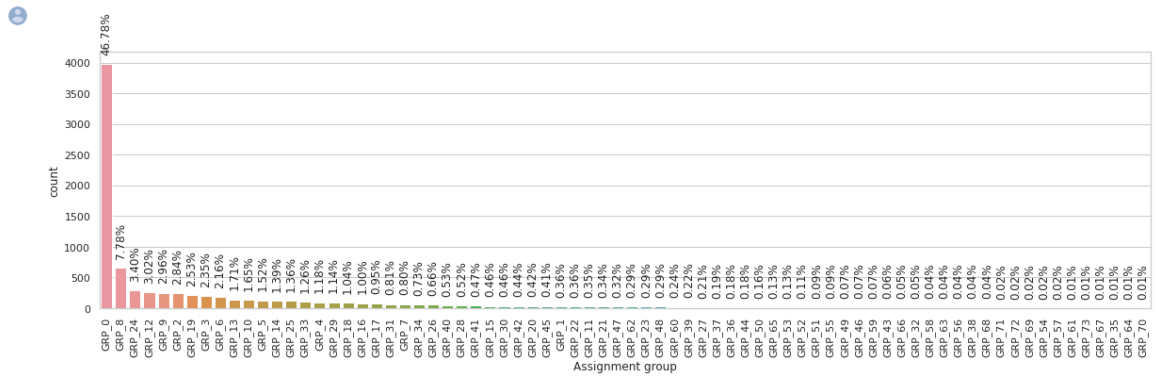


Image 6.3.3.1: Count plot for Assignment groups in descending order

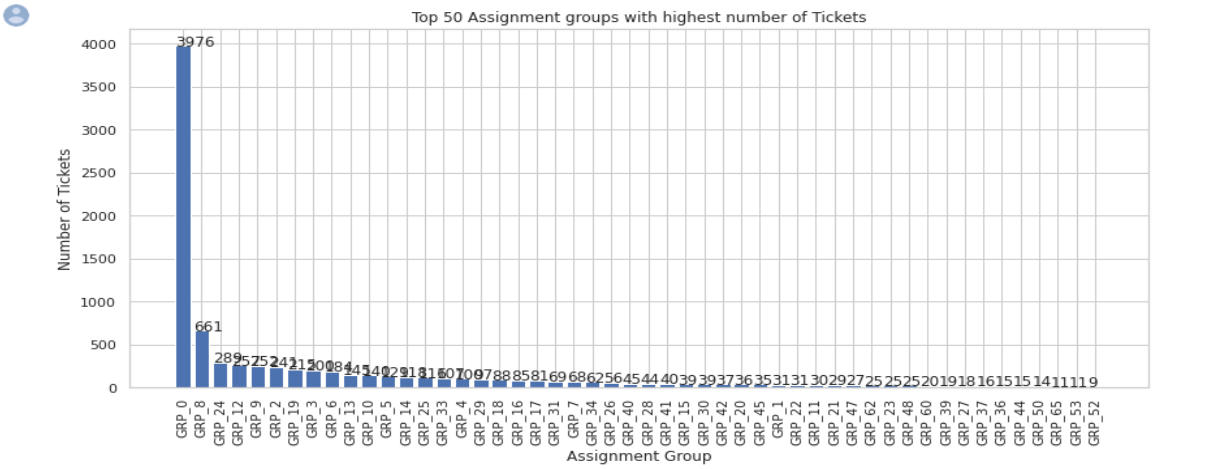


Image 6.3.3.2: Count plot for Top 50 frequent assignment groups

Image 6.3.3.3: Word Cloud for Assignment Group

It has been observed that there are **total 74 unique groups** and the distribution of them is **highly imbalanced**.**GRP_0 is the most frequent group which has 3976 tickets**.GRP_8 is the second most frequent group. That is why **GRP_0 and GRP_8** is very prominent in word cloud.

Visualization of Caller Column:

We have found top 10 callers and plotted them in a bar graph to see the variations. Also, we have plotted Assignment group distribution for each of the top callers in the pie charts to find if there is any importance of the caller column or not.

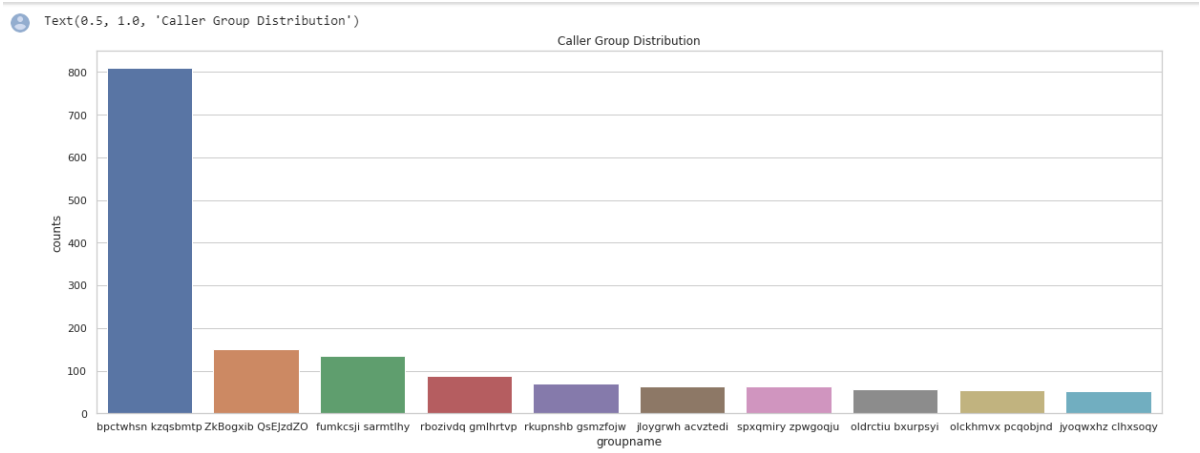


Image 6.3.3.4: Bar graph for top 10 callers

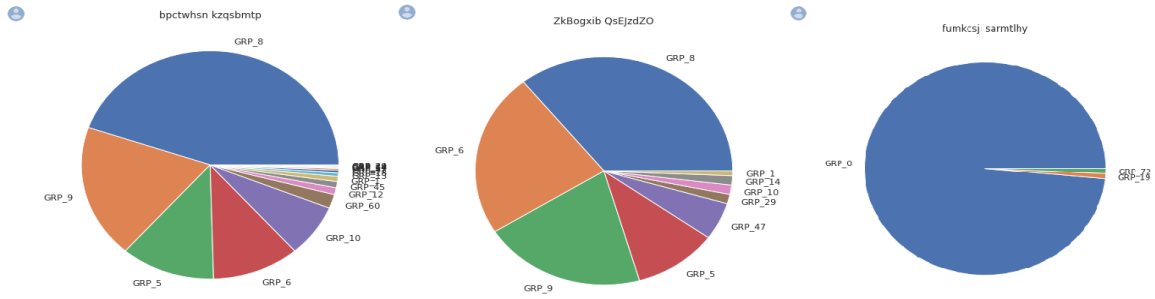


Image 6.3.3.5: Pie charts for each top caller

From the bar graph of top 10 callers it is observed that "`bpctwhsnkzqsbmtp`" is the **top caller having 801 tickets** raised for different assignment groups.

From the pie charts we have also inferred that as **each of the top 10 callers have raised requests in multiple Assignment groups. So, Caller column has no significance on target variable (Assignment Group). We can drop the column.**

Despite of GRP_0 has maximum tickets we see that GRP_8 has maximum of the top callers.

Visualization of number of words of description and short description:

We have plotted the distribution of number of words for description and short description. Also, calculated the maximum and minimum number of words in it.

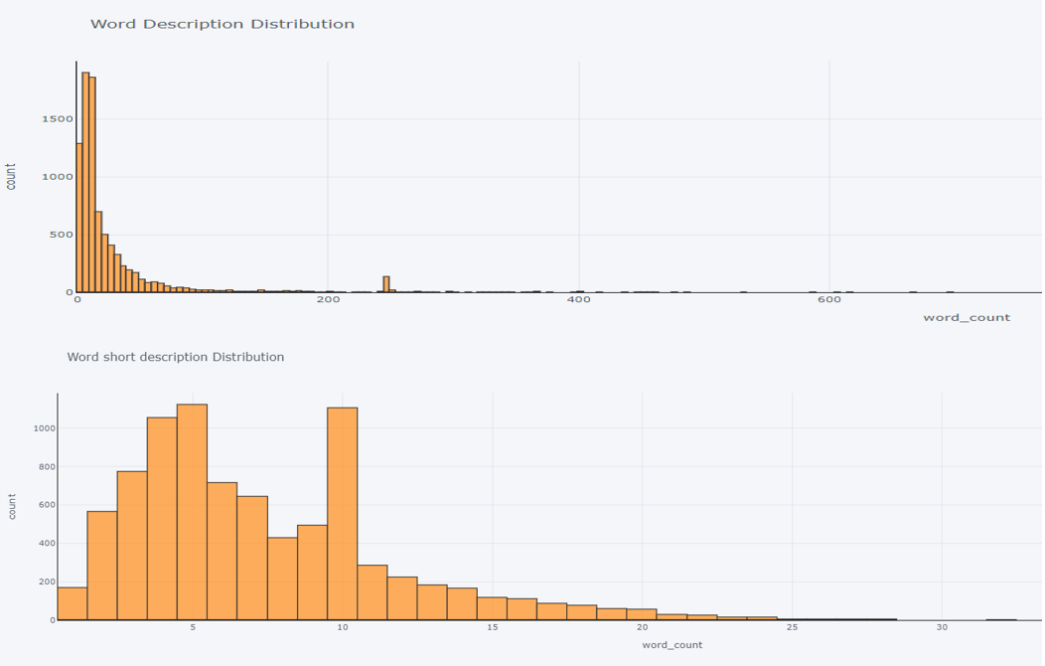


Image6.3.3.6: Word count plot for Description and short description

From the above visualizations we have inferred that **the length of the description is highly imbalanced.**

6.4. Dealing with missing values

We have combined column short description and Description and created a combined column named as "Summary". Here we have handled the missing values by concatenating with '**The**' in place of all missing values which will be automatically removed with stop words later.

```
(8500, 5)
Short description    0
Description          0
Caller               0
Assignment group     0
Summary              0
dtype: int64
```

Short description	Description	Caller	Assignment group	Summary
i am locked out of skype	The	viyglzfo ajtfzpkb	GRP_0	i am locked out of skype The

Image6.4: Handling missing values of Summary column

Word Cloud Visualization for Summary column

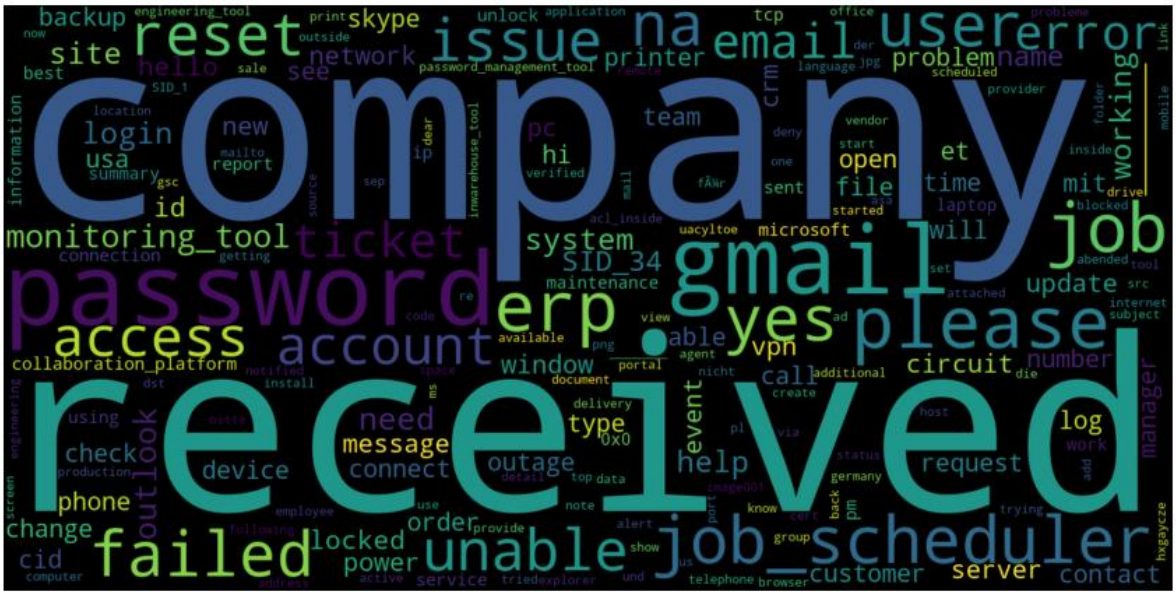


Image6.4: Word Cloud Visualization of Summary column

We can visualize that the **word cloud is messy**. Moreover, if we observe carefully, we can see there are words with **non-English languages** too. **This observation highlights the inconsistency in summary column with the presence of non-English rows.**

6.5. Detection of non-English rows and Translation

Lang detect and polyglot library is used to detect the number of English rows. A separate column is created to save the language type. The language accuracy is checked after that for both of the libraries. We have performed the same activity on cleaned/preprocessed Summary and checked the language accuracy. Following that, the dataset is divided into two parts, one with **English** language rows and other with **non-English** language rows. Translation is applied on non-English rows by **Google translator**, after which the non-English data frame is merged with English data frame to form the final data frame. The language accuracy is checked again after the translation. The final feature is ready now for text pre preprocessing.

```
newdf["language1"].value_counts(
```

en	7097
de	376
af	264
it	121
fr	119
no	75
sv	74
da	72

```
newdf["lang"].value_counts()
```

English	7935
German	401
Danish	43
un	28
Latin	12
Portuguese	10
Hungarian	8
Waray	7
Nauru	6
Scots	5
Luxembourgish	5

Image6.5.1: Comparison of Lang detect and polyglot results on language accuracy for raw Summary

For **language detection** we have observed that the **detection works better for unprocessed text rather than processed text. Image 6.5.2 depicts that we have a smaller number of English rows (7878) with clean Summary.**

The second observation is that **polyglot works better in terms of language accuracy with 7928 English rows whereas for Lang detect it is 7097**.

```
newcleaneddf["lang"].value_counts()
```

English	7878
German	396
Danish	43
un	29
Latin	22
Portuguese	14
Samoan	10
Welsh	10
Nauru	9
Luxembourgish	6
Hungarian	6

Image 6.5.2: Polyglot detection on cleaned and processed Summary

English	7935
German	401
Danish	43
un	28
Latin	12
Portuguese	10
Hungarian	8
Waray	7
Nauru	6
Scots	5
Luxembourgish	5
Polish	5
Uzbek	3
Zhuang	3

English	8318
German	76
un	28
Danish	14
Latin	9
Nauru	6
Waray	5
Hungarian	4
Polish	4
Scots	4
Uzbek	3
Portuguese	3
Ganda	2

Image6.5.3: Comparison of English rows for before and after translation

After translation, it is seen that almost 8300+ rows got translated. Out of the remaining rows, it is seen that most of the rows (almost 75%) are in English and the rest in non-English. Due to some words in the remaining part, it is detected as non-English. So, with this, we are good to go and we start doing further visualizations and text processing.

6.6. Text Preprocessing

In the preprocessing, **all the numeric characters, special characters, URL's, Email ids, Extra spaces, newlines and HTML's are removed.** Then the **abbreviations** are **expanded**.
Lemmatization is applied after this to have the root form of the words. Once the text preprocessing is done then the word cloud on the cleaned summary is displayed.

```
0     login issue verify user detail employee manage...
1     outlook receive from hello team my meeting sky...
2     can not log in to vpn receive from hi i can no...
3                        unable to access hr tool page
4           unable to log in engineering tool and skype
5     event critical hostname company com the value ...
6     ticket no employment status new non employee e...
7                  unable to disable add in on outlook
8                            ticket update on inplant
9     engineering tool say not connect and unable to...
10                hr tool site not load page correctly
```

Image6.6.1: First 10 rows of cleaned summary

6.7. Removal of Stop words and Tokenization

After the text preprocessing is done**, the stop words list is made by dropping out the negative words.** Then we remove customized stop words from the Summary to create the final feature column**.** Once the final feature is created, it is tokenized to form the ultimate vocabulary ready for embedding.

```
0    login issue verify user detailsemployee manage...
1    outlook receive hello team meetingsskype meeti...
2              not log vpn receive hi not log vpn best
3    unable access hrtool page unable access hrtool...
4    unable log engineering tool skype unable log e...
Name: CleanTextAfterRemoveStopword, dtype: object
```

```
stop_words = set(stopwords.words('english')) - set(["hadn't", "didn't", "shouldn't","shan't","hasn
print("\nList of fresh stopwords in English:")
print(stop_words)
```

Image6.7.1: Head of CleanTextAfterRemoveStopword column and stop words list without negative words

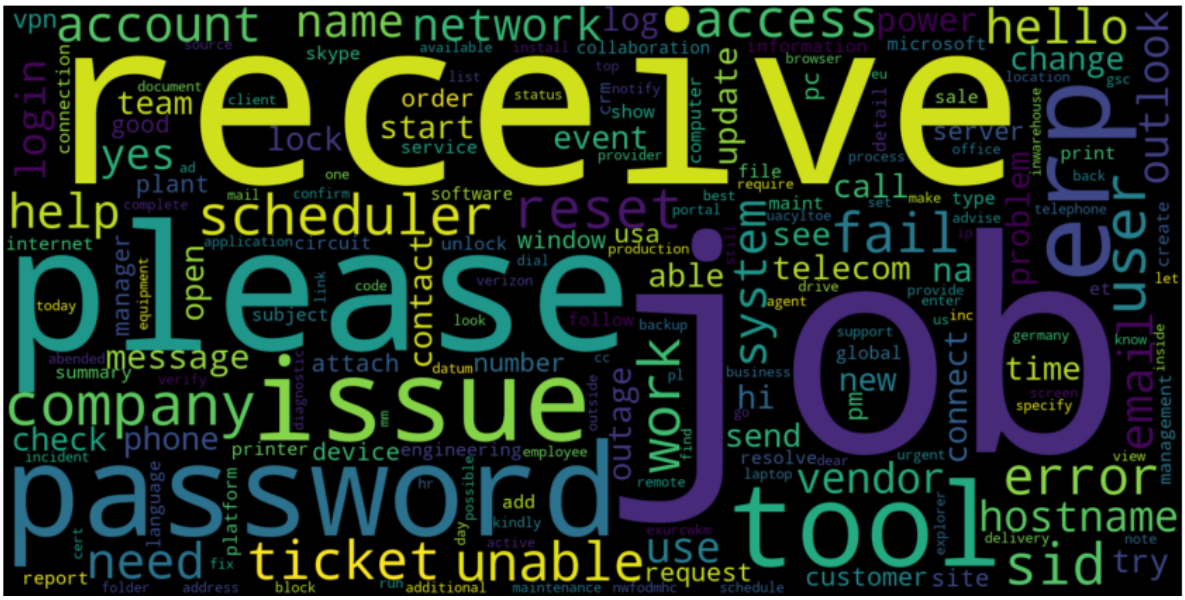


Image6.7.2: Word Cloud comparison of cleaned summary without stop words

When we plot the word cloud, **we observe that we get a better word cloud without special characters and the words are well lemmatized.**

	Assignment group	CleanTextAfterRemoveStopword	tokenizesummaryafterstopword
0	GRP_0	login issue verify user detail employee manage...	[login, issue, verify, user, detail, employee,...
1	GRP_0	outlook receive hello team meeting skype meeti...	[outlook, receive, hello, team, meeting, skype...
2	GRP_0	not log vpn receive hi not vpn best	[not, log, vpn, receive, hi, not, vpn, best]
3	GRP_0	unable access hr tool page	[unable, access, hr, tool, page]
4	GRP_0	unable log engineering tool skype	[unable, log, engineering, tool, skype]

Image6.7.3: Final data frame for Model Building

7.Model Building

Based on the above observations and our understanding regarding the problem, we start building our model. Before that we should try upon different featurization like **count vectorizer**, **TF-IDF vectorizer**, **both of them** and **glove embedding** to get the optimization done in a structured way. Different models are built so as to compare the **conventional** as well as the **deep learning** ones and see which gives us the best final accuracy and other metrices.

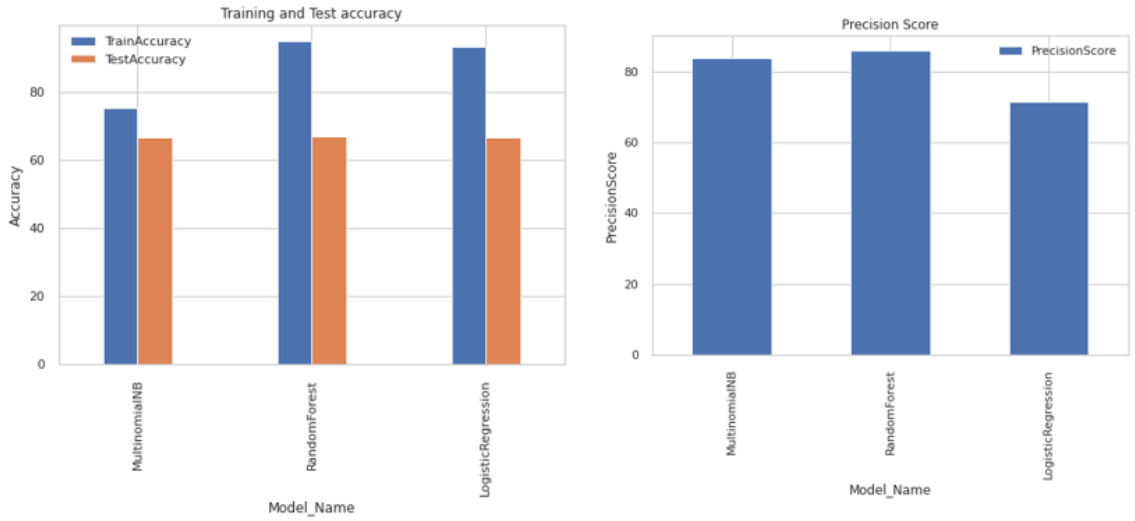
7.1. Feature Creation and Build Conventional ML models

TF-IDF and Count vectorizer and both in pipeline are tried on the data. Other than this transformed input is applied to different conventional Machine Learning models. The vocabulary is built and the target column is converted in to label encoded output.

7.1.1. Output of ML Models with Count Vectorizer

	ModelName	MultinomialNB	RandomForest	LogisticRegression
0	TrainAccuracy	70.573529	94.617647	93.500000
1	TestAccuracy	65.058824	66.588235	65.176471
2	PrecisionScore	88.205077	85.848585	70.413316
3	RecallScore	65.058824	66.588235	65.176471
4	F1Score	73.232678	73.226292	67.156307

Image7.1.1.1: Comparison of ML models with count vectorizer



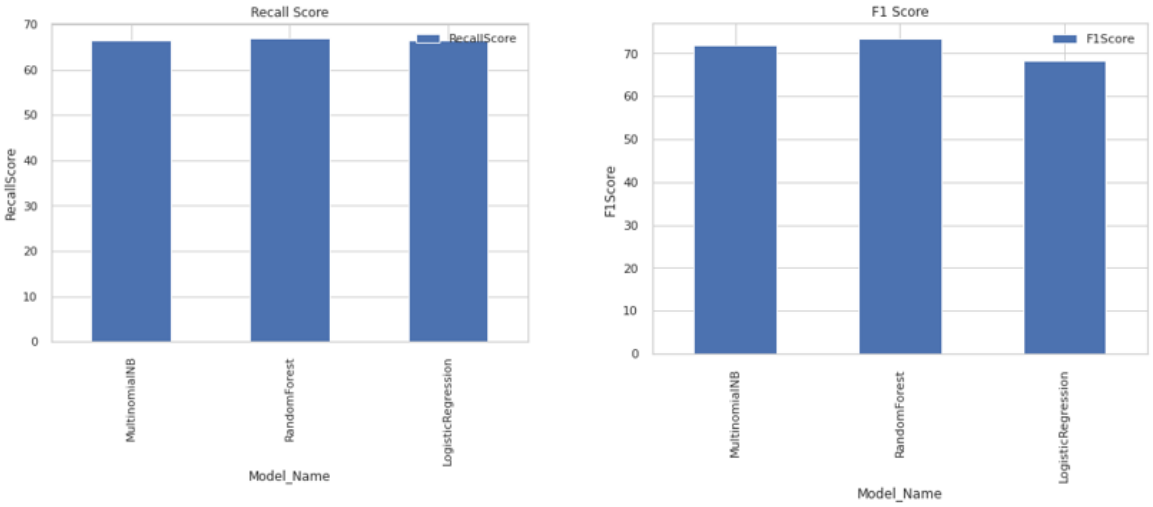


Image7.1.1.2: Plot different metrices to compare ML models with count vectorizer

From the comparison mentioned above it is inferred that **though Logistic regression and Random forest gives better training accuracy but the test accuracy comes out to be very poor and large overfit is there**. **Multinomial NB gives 70% training accuracy in the training dataset and 65% in the testing data which shows that it has less amount of overfit. So, for count vectorizer, multinomial NB gives better result in terms of accuracy and precision**. But it is observed that all the models have low recall.

Classification Report:

Multinomial Naive Bayes

	precision	recall	f1-score	support
0	0.66	0.99	0.79	829
1	0.00	0.00	0.00	5
2	1.00	0.07	0.13	28
3	0.00	0.00	0.00	4
4	0.46	0.41	0.44	41
5	0.47	0.41	0.44	34
6	0.62	0.36	0.46	22
7	1.00	0.17	0.29	6
8	0.00	0.00	0.00	22
9	0.92	0.69	0.79	16
10	0.44	0.47	0.45	15
11	0.44	0.09	0.15	45
12	0.52	0.26	0.34	43
13	1.00	0.12	0.22	8
14	0.00	0.00	0.00	6
15	0.00	0.00	0.00	8
16	0.00	0.00	0.00	1
17	0.90	0.78	0.83	55
18	0.75	0.15	0.25	20
19	0.00	0.00	0.00	14
20	0.00	0.00	0.00	3
21	0.00	0.00	0.00	8
22	0.80	0.19	0.31	21
23	0.40	0.06	0.11	33
24	0.00	0.00	0.00	8
25	0.00	0.00	0.00	11
27	0.60	0.29	0.39	21
28	1.00	0.15	0.27	13
29	0.00	0.00	0.00	1
30	0.00	0.00	0.00	5
31	0.00	0.00	0.00	2
32	0.00	0.00	0.00	1
33	0.00	0.00	0.00	1
34	0.89	0.28	0.42	29
35	0.00	0.00	0.00	11
36	1.00	0.20	0.33	5
37	0.00	0.00	0.00	12
38	0.00	0.00	0.00	1
39	0.00	0.00	0.00	3
40	0.00	0.00	0.00	8
42	0.00	0.00	0.00	4
43	0.00	0.00	0.00	6
44	0.00	0.00	0.00	2
45	0.40	0.33	0.36	18
46	0.00	0.00	0.00	3
47	0.00	0.00	0.00	2
48	0.00	0.00	0.00	1
49	0.00	0.00	0.00	3
50	0.00	0.00	0.00	1
51	0.00	0.00	0.00	2
53	0.00	0.00	0.00	1
55	0.00	0.00	0.00	1
56	0.85	0.31	0.46	35
57	0.00	0.00	0.00	4
59	1.00	0.17	0.29	6
62	0.00	0.00	0.00	1
63	0.00	0.00	0.00	2
66	0.00	0.00	0.00	1
67	0.00	0.00	0.00	12
68	0.00	0.00	0.00	1
70	0.00	0.00	0.00	1
72	0.59	0.90	0.71	130
73	0.50	0.11	0.19	44
accuracy			0.65	1700
macro avg	0.27	0.13	0.15	1700
weighted avg	0.59	0.65	0.56	1700

Image7.1.1.3: Classification Report for Multinomial NB

7.1.2. Output of ML Models with TF-IDF Vectorizer

	ModelName	MultinomialNB_TFIdf	RandomForestClassifier_TFIdf	LogisticRegression_TFIdf	SGDClassifier_TFIdf
0	TrainAccuracy	55.867647	94.867647	94.573529	72.397059
1	TestAccuracy	58.352941	66.764706	68.470588	65.235294
2	PrecisionScore	98.253895	87.054571	74.516741	86.552126
3	RecallScore	58.352941	66.764706	68.470588	65.235294
4	F1Score	72.514705	73.540693	70.516011	72.125994

Image7.1.2.1: Comparison of ML models with TF-IDF vectorizer

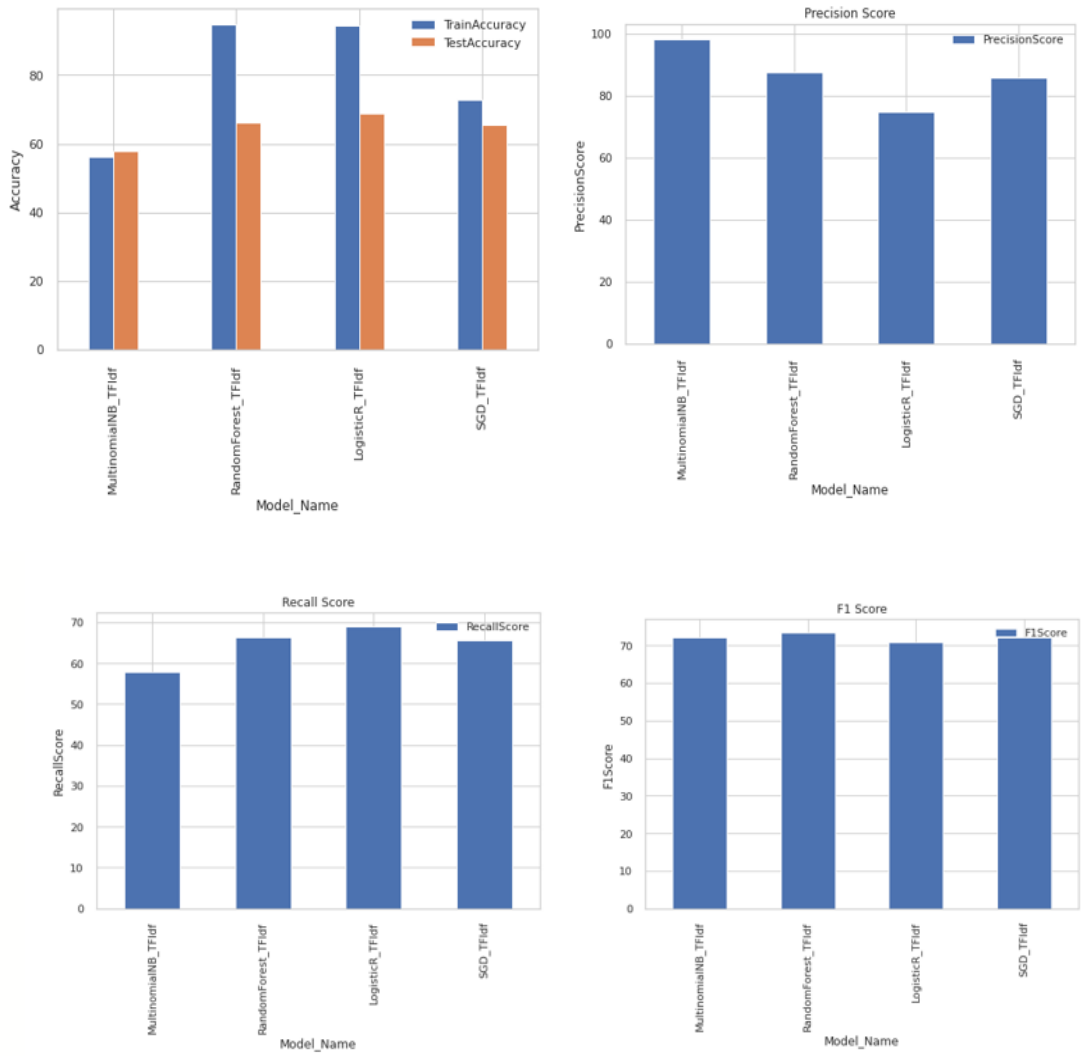


Image7.1.2.2: Plot different metrices to compare ML models with TF-IDF vectorizer

From the comparison mentioned above it is inferred that **though Logistic regression and Random forest gives better training accuracy but the test accuracy comes out to be very poor. SGD Classifier gives 72% training accuracy in the training dataset and 65% in the testing data which shows that it has less amount of overfit. So, for TF-IDF vectorizer, SGD Classifier gives better result in terms of accuracy. But if we consider precision score also Multinomial NB gives best**. It is observed that all the models have low recall.

7.1.3. Output of ML Models with Count with TF-IDF Vectorizer in pipeline

```
text_clf = Pipeline([('vect', CountVectorizer()),('tfidf', TfidfTransformer()), ('clf', MultinomialNB())])
#text_clf.fit(x_train_tfid, y_train_tfidf)
```

```
data_m=predict_ml_model(text_clf,x_train_tfid,y_train_tfidf,x_test_tfidf,y_test_tfidf)
modecomparedf['MultinomialNB_CountandTFIdf'] = data_m
```

	ModelName	MultinomialNB_CountandTFIdf	LogisticRegression_CountandTFIdf	RandomForestClassifier_CountandTFIdf	SGDClassifier_CountandTFIdf
0	TrainAccuracy	55.966387	94.739496	94.924370	72.873950
1	TestAccuracy	56.549020	66.431373	64.745098	64.627451
2	PrecisionScore	97.969329	73.104627	86.762216	88.243194
3	RecallScore	56.549020	66.431373	64.745098	64.627451
4	F1Score	70.929721	68.605443	71.977776	72.782591

Image7.1.3: Code snippet and comparison ML models with Count and TF-IDF vectorizer in pipeline

From the comparison mentioned above it is inferred that though **Logistic regression and Random forest gives better training accuracy but the test accuracy comes out to be very poor. SGD Classifier NB gives 73% training accuracy in the training dataset and 65% in the testing data which shows that it has less amount of overfit. So, for Count and TF-IDF vectorizer, SGD Classifier gives better result in terms of accuracy**. **But if we consider precision score also Multinomial NB gives best**. Applying both vectorizer gives same result as TF-IDF It is observed that all the models have low recall.

7.2. Building deep learning models

We build the basic model with 3 layers (**one embedding, LSTM and Dense layer**) with and without the weight matrix. After which we calculate the accuracy for the same. This base model is compiled having loss as **categorical_crossentropy** and **Adam** as optimizer. **We have applied early stopping for reducing execution time.**

7.2.1. Basic LSTM model without weight matrix

```
60/60 [==============================] - 92s 2s/step - loss: 0.7973 - accuracy: 0.7914 - val_loss: 1.8642 - val_accuracy: 0.5925
Epoch 15/30
60/60 [==============================] - 92s 2s/step - loss: 0.7494 - accuracy: 0.8077 - val_loss: 1.8719 - val_accuracy: 0.6008
Epoch 16/30
60/60 [==============================] - 92s 2s/step - loss: 0.6645 - accuracy: 0.8286 - val_loss: 1.8528 - val_accuracy: 0.5925
Epoch 17/30
60/60 [==============================] - 96s 2s/step - loss: 0.6123 - accuracy: 0.8410 - val_loss: 1.9000 - val_accuracy: 0.5949
```

Image7.2.1.1: Result of Basic LSTM models without the weight matrix

It is observed that **the model is highly over fit as the training accuracy is coming high but the testing is coming maximum around 60% only**.

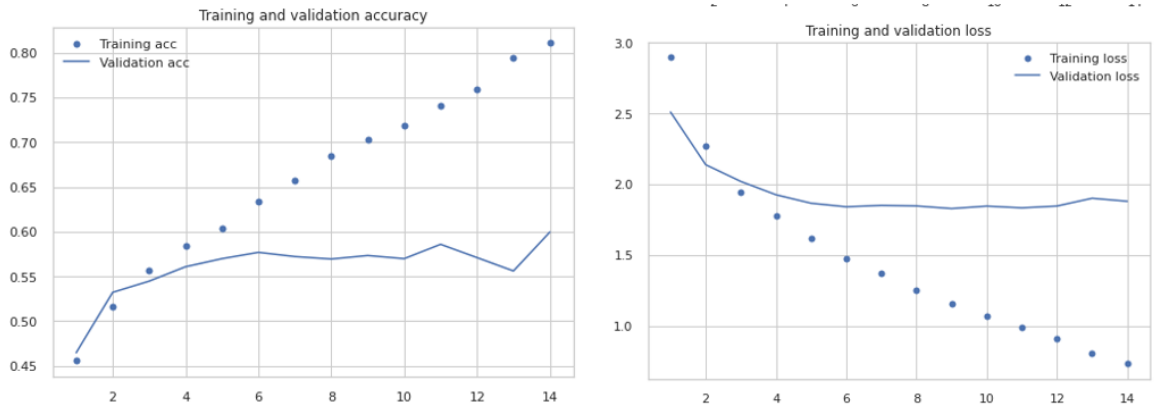


Image7.2.1.2: Error plots depicting Accuracy and validation loss

In Training and Validation loss, it can be observed **that validation loss got saturated for testing data whereas training data it went decreasing and gives an overfit.**

7.2.2. Basic LSTM model with weight matrix

We build the basic model with 3 layers (one embedding, LSTM and Dense layer) with the GloVe weight matrix. After which we checked the accuracy.

```
Epoch 13/30
60/60 [==============================] - 128s 2s/step - loss: 0.6564 - accuracy: 0.8286 - val_loss: 1.5487 - val_accuracy: 0.6314
Epoch 14/30
60/60 [==============================] - 134s 2s/step - loss: 0.6017 - accuracy: 0.8410 - val_loss: 1.5691 - val_accuracy: 0.6447
Epoch 15/30
60/60 [==============================] - 129s 2s/step - loss: 0.5469 - accuracy: 0.8550 - val_loss: 1.5827 - val_accuracy: 0.6365
```

Image7.2.2.1: Result of Basic LSTM models with the weight matrix

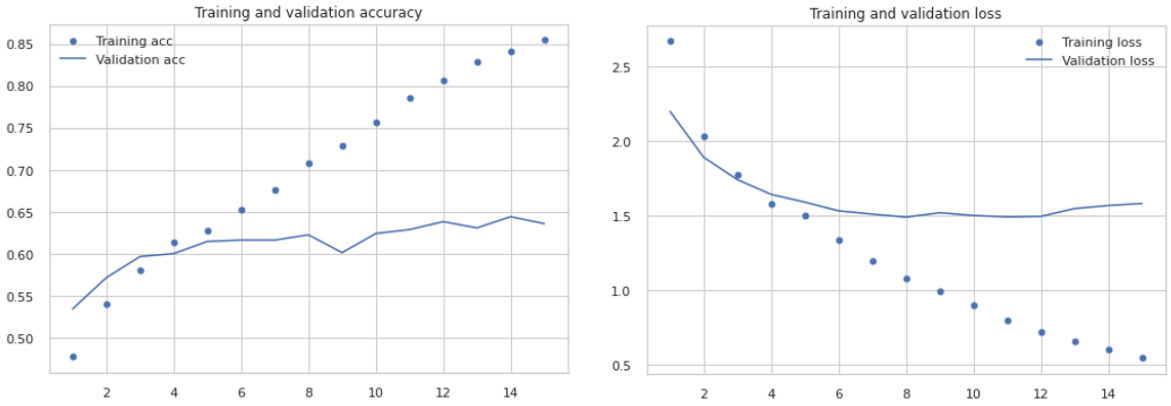


Image7.2.2.2: Error plots depicting Accuracy and validation loss

It is observed **that the model has the training accuracy high but the testing is coming maximum around 64% only. In Training and Validation loss, it can be observed that validation loss got saturated for testing data whereas training data it went decreasing and gives an overfit. But with weight matrix there is a 4% increase in accuracy.**

	precision	recall	f1-score	support
0	0.83	0.87	0.85	1223
1	0.40	0.20	0.27	10
2	0.34	0.49	0.40	39
3	0.25	0.09	0.13	11
4	0.52	0.52	0.52	77
5	0.51	0.40	0.45	48
6	0.34	0.62	0.44	32
7	0.38	0.29	0.33	17
8	0.34	0.63	0.44	19
9	0.89	0.89	0.89	19
10	0.38	0.33	0.36	24
11	0.23	0.27	0.25	66
12	0.51	0.44	0.47	73
13	0.00	0.00	0.00	13
14	0.00	0.00	0.00	7
15	1.00	0.08	0.15	12
16	0.20	0.17	0.18	6
17	0.71	0.91	0.80	86
18	0.31	0.45	0.37	33
19	0.00	0.00	0.00	20
20	0.00	0.00	0.00	3
21	0.50	0.06	0.11	16
22	0.28	0.29	0.28	28
23	0.20	0.27	0.23	48
24	0.00	0.00	0.00	12
25	0.43	0.35	0.39	17
26	0.00	0.00	0.00	1
27	0.38	0.28	0.32	39
28	0.11	0.10	0.10	21
29	0.00	0.00	0.00	0
30	0.00	0.00	0.00	4
31	0.50	0.17	0.25	6
32	0.00	0.00	0.00	1
33	0.50	0.67	0.57	3
34	0.32	0.17	0.22	35
micro avg	0.64	0.64	0.64	2550
macro avg	0.23	0.18	0.18	2550
weighted avg	0.61	0.64	0.61	2550
samples avg	0.64	0.64	0.64	2550

35	0.36	0.22	0.28	18
36	0.43	0.23	0.30	13
37	0.00	0.00	0.00	10
38	0.00	0.00	0.00	2
39	0.00	0.00	0.00	4
40	0.00	0.00	0.00	11
41	0.00	0.00	0.00	1
42	1.00	0.20	0.33	10
43	0.20	0.43	0.27	7
44	0.00	0.00	0.00	2
45	0.61	0.36	0.45	39
46	0.00	0.00	0.00	7
47	0.00	0.00	0.00	4
48	0.00	0.00	0.00	1
49	0.00	0.00	0.00	3
50	0.00	0.00	0.00	0
51	0.00	0.00	0.00	2
52	0.00	0.00	0.00	0
53	0.00	0.00	0.00	0
54	0.00	0.00	0.00	1
55	0.00	0.00	0.00	2
56	0.51	0.51	0.51	53
57	1.00	0.12	0.22	8
58	0.00	0.00	0.00	1
59	0.00	0.00	0.00	7
60	0.00	0.00	0.00	0
61	0.00	0.00	0.00	0
62	0.00	0.00	0.00	4
63	0.00	0.00	0.00	2
64	0.00	0.00	0.00	0
65	0.00	0.00	0.00	0
66	0.00	0.00	0.00	0
67	0.63	0.55	0.59	22
68	0.00	0.00	0.00	0
69	0.00	0.00	0.00	1
70	0.00	0.00	0.00	0
71	0.00	0.00	0.00	0
72	0.53	0.84	0.65	179
73	0.43	0.13	0.20	67

Image7.2.2.3: Classification Report for LSTM with Glove

In classification report we can see that GRP_0 has a good score as it has maximum and large records but for the groups which has smaller number of records the score is not well.

8.Model Descriptions of LSTM variations with Glove 100D

Now our base model with LSTM is ready, we will try different modifications and LSTM variations to analyze the performance and to obtain accuracy and losses for different deep learning models. We will also analyze and compare different model performances for selecting best model. First, we will use **Glove 100 weight matrix then Glove 300** respectively. We have run all models with larger number of epochs and applied early stopping so Epochs mentioned in the Model output is a result of early stopping.

Data Set Restriction: First we ran all the models on whole data set but observed that the accuracy is not satisfactory and data has large imbalance. So, we have taken top 5 group data set and calculated the result on the top of it. Top 5 Group data set have 5435 rows and same number of columns compare to complete data set.

```
finalticketsummary['Assignment group'].value_counts()

GRP_0     3976
GRP_8      661
GRP_24     289
GRP_12     257
GRP_9      252
Name: Assignment group, dtype: int64
```

Image8: Tickets count for top 5 groups

8.1. Models with Complete data set

Sno.	Model Name	Model Description	Model Output
1	Basic model with one Embedding,LSTM and Dense layer without weight matrix	This model has 1 embedding layer 1 lstm layer with output 100 and return sequence as false and onse dense layer with output 74 as 74 groups are there for prediction. Loss we have used catagorical_crossentropy and optimizer is adam and metrics is accuracy	We can observe that there are 1,263,874 trainable parameters and 0 nontrainable parameters in model. The accuracy after 11 epochs comes with 83% in train and 58% in test which has a very large overfit and too less training loss compare to validation loss. The error plot also explains the overfit.
2	Base model with glove 100d weight matrix	This model has 1 embedding layer with weight matrix of glove 100, 1 lstm layer with output 128 and return sequence as false and one dense layer with softmax activation Loss we have used catagorical_crossentropy and optimizer is adam and metrics is accuracy	We have created a simple LSTM model with 3 layers with glove embedding weight matrix and checked the accuracy.The training accuracy is coming high but testing is max 64%.We need to explore more to improve accuracy There is a hudge overfit and the scores for some groups are coming not well
3	One layer bidirectional LSTM deep learning model	This model has 1 embedding layer with glove 100d weight matrix, 1 Bi directional LSTM layer with output 128 and return sequence as false and one dense layer with soft max in dense layer as activation Loss we have used catagorical_crossentropy and optimizer is adam and metrics is accuracy	We can observe that there are 1,411,814 trainable parameters and 0 nontrainable parameters in model. The accuracy after 14 epochs comes with 86% in train and 61% in test which has a larger overfit and too less training loss compare to validation loss.
4	One layer bidirectional LSTM deep learning model with flatten and return sequence as True	This model has 1 embedding layer with glove 100d 1 bi directional LSTM layer with output 128 and return sequence as true, one flatten layer and one dense layer with output 74 Loss we have used catagorical_crossentropy and optimizer is adam and metrics is accuracy	We can observe that there are 14,142,182 trainable parameters and 0 nontrainable parameters in model. The accuracy after 8 epochs with early stopping comes with 90% in train and 64% in test which has a very large overfit and too less training loss compare to validation loss.

Sno.	Model Name	Model Description	Model Output
5	One layer bidirectional LSTM deep learning model with GlobalMaxPool1D and return sequence as True	This model has 1 embedding layer with glove 100d embedding and 1 bidirectional LSTM layer with output 128 and return sequence as true and one globalmaxpool1D layer and 3 dense layers with final layer having output 74 Loss we have used catagorical_crossentropy and optimizer is adam and metrics is accuracy	We can observe that there are 1,450,270 trainable parameters and 0 nontrainable parameters in model. The accuracy after 10 epochs with early stopping comes with 84% in train and 61% in test which has a very large overfit and validation loss is very high
6	Two layers bidirectional LSTM deep learning model with return sequence as True and glove 100 embedding and 1 dense layer and 1 flatten layer	This model has 1 embedding layer 2 bidirectional LSTM layers with weight matrix as glove 100 and return sequence as True and one flatten and one dense layer with output 74 Loss we have used catagorical_crossentropy and optimizer is adam and metrics is accuracy	We can observe that there are 14,554,122 trainable parameters and 0 nontrainable parameters in model. The accuracy after 8 epochs comes with 87% in train and 61% in test which has a larger overfit and too less training loss compare to validation loss.
7	Two layers bidirectional LSTM deep learning model with return sequence as True and glove 100 embedding and 1 dense layer and 1 flatten layer ,1 drop out layer and 1 spatial drop out layer	This model has 1 embedding layer 2 bidirectional LSTM layers with weight matrix as glove 100 and return sequence as True and one flatten and one dense layer with 1 drop out layer and 1 spatial drop out layer . Drop out we have used 0.3 and spatial drop out as 0.4 Loss we have used catagorical_crossentropy and optimizer is adam and metrics is accuracy	We can observe that there are 14,554,122 trainable parameters and 0 nontrainable parameters in model. The accuracy after 10 epochs comes with 77% in train and 63% in test which has a less overfit compare to all other models and one of the best performance we got so far.Though there is high imbalance

Image 8.1.1: Model Descriptions and Output

Model Name	Data Set Description	Training Accuracy	Testing Accuracy	Training Loss	Testing Loss	Explanation Of Classification Report
Basic model with one Embedding,LSTM and Dense layer without weight matrix	Complete	83.14%	58.08%	0.6543	1.9152	Classification report depicts that the scores are not good we will try to improve more.The classification report did not went well due to hudge imbalance in complete data
Base model with glove 100d weight matrix	Complete	89.33%	64.63%	0.4001	1.5795	Classification report depicts that the scores are not good we will try to improve more.The classification report did not went well due to hudge imbalance in complete data
One layer bidirectional LSTM deep learning model	Complete	86.30%	61.49%	0.5158	1.6269	The classification Report depicts the f1 score came poor for many groups among 74 along with that due to large overfit we cannot select this model.
One layer bidirectional LSTM deep learning model with flatten and return sequence as True	Complete	90.47%	64.16%	0.4111	2.168	The classification Report depicts the f1 score came poor for many groups among 74 along with that due to large overfit we cannot select this model.
One layer bidirectional LSTM deep learning model with GlobalMaxPool1D and return sequence as True	Complete	84.45%	61.65%	0.5433	1.8845	The classification Report depicts the f1 score came poor for many groups among 74 along with that due to large overfit we cannot select this model.The validation loss is very high

Model Name	Data Set Description	Training Accuracy	Testing Accuracy	Training Loss	Testing Loss	Explanation Of Classification Report
Two layers bidirectional LSTM deep learning model with return sequence as True and glove 100 embedding and 1 dense layer and 1 flatten layer	Complete	87.24%	61.41%	0.4432	1.959	The classification Report depicts the f1 score came poor for many groups among 74 along with that due to large overfit we cannot select this model.The validation loss is very high
Two layers bidirectional LSTM deep learning model with return sequence as True and glove 100 embedding and 1 dense layer and 1 flatten layer ,1 drop out layer and 1 spatial drop out layer	Complete	77.92%	63.18%	0.739	1.8563	The classification Report depicts the f1 score came poor for many groups among 74 but the overall performance is best among all models we have tried so far on complete data set.

Image 8.1.2: Comparison Of accuracy, losses and classification report

8.2. Models with Top 5 Group data

Sno.	Model Name	Model Description	Model Output
1	Basic model with one Embedding,LSTM and Dense layer without weight matrix	This model has 1 embedding layer 1 lstm layer with output 100 and return sequence as false and onse dense layer with output 5 as 5 groups are there for prediction. Loss we have used catagorical_crossentropy and optimizer is adam and metrics is accuracy	There are 755,205 trainable parameters and 0 nontrainable parameters in model. The accuracy after 12 epochs comes with 96% in train and 90% in test which has a overfit and too less training loss compare to validation loss.
2	Base model with glove 100d weight matrix	This model has 1 embedding layer with weight matrix of glove 100, 1 lstm layer with output 128 and return sequence as false and one dense layer with softmax activation Loss we have used catagorical_crossentropy and optimizer is adam and metrics is accuracy	We can observe that there are 792193 trainable parameters and 0 nontrainable parameters in model. The accuracy after 8 epochs comes with 94% in train and 91% in test which has a overfit.
3	One layer bidirectional LSTM deep learning model	This model has 1 embedding layer with glove 100d weight matrix, 1 Bi directional LSTM layer with output 128 and return sequence as false and one dense layer with output 5 as 5 groups are there for prediction with soft max in dense layer as activation Loss we have used catagorical_crossentropy and optimizer is adam and metrics is accuracy	There are 910,081 trainable parameters and 0 nontrainable parameters in model. The accuracy after 8 epochs comes with 94% in train and 91% in test which has a overfit and too less training loss compare to validation loss.
4	One layer bidirectional LSTM deep learning model with flatten and return sequence as True	This model has 1 embedding layer with glov 100d 1 bi directional LSTM layer with output 128 and return sequence as true, one flatten layer and one dense layer with output 5 as 5 groups are there for prediction. Loss we have used catagorical_crossentropy and optimizer is adam and metrics is accuracy	We can observe that there are 1,359,361 trainable parameters and 0 nontrainable parameters in model. The accuracy after 6 epochs with early stopping comes with 95% in train and 90% in test which has a larger overfit and to less training loss compare to validation loss.

5	One layer bidirectional LSTM deep learning model with GlobalMaxPool1D and return sequence as True	This model has 1 embedding layer with glove 100d embedding and 1 bidirectional LSTM layer with output 128 and return sequence as true and one globalmaxpool1D layer and 3 dense layers with final layer having output as 5 as 5 groups are there for prediction. Loss we have used catagorical_crossentropy and optimizer is adam and metrics is accuracy	We can observe that there are 942,981 trainable parameters and 0 nontrainable parameters in model. The accuracy after 8 epochs with early stopping comes with 96% in train and 90% in test which has a larger overfit and to less training loss compare to validation loss.
6	Two layers bidirectional LSTM deep learning model with return sequence as True and glove 100 embedding and 1 dense layer and 1 flatten layer	This model has 1 embedding layer 2 bidirectional LSTM layers with weight matrix as glove 100 and return sequence as True and one flatten and one dense layer with output 5 as 5 groups are there for prediction. Loss we have used catagorical_crossentropy and optimizer is adam and metrics is accuracy	We can observe that there are 1,753,601 trainable parameters and 0 nontrainable parameters in model. The accuracy after 10 epochs comes with 95% in train and 89% in test which has a larger overfit and to less training loss compare to validation loss. The error plot also explains the overfit.
7	Two layers bidirectional LSTM deep learning model with return sequence as True and glove 100 embedding and 1 dense layer and 1 flatten layer ,1 drop out layer and 1 spatial drop out layer	This model has 1 embedding layer 2 bidirectional LSTM layers with weight matrix as glove 100 and return sequence as True and one flatten and one dense layer with 1 drop out layer and 1 spatial drop out layer . Drop out we have used 0.3 and spatial drop out as 0.4 Loss we have used catagorical_crossentropy and optimizer is adam and metrics is accuracy	We can observe that there are 1,753,601 trainable parameters and 0 nontrainable parameters in model. The accuracy after 10 epochs comes with 93% in train and 90% in test which has a less overfit compare to all other models and one of the best performance we got. The error plot is comparatively well

Image 8.2.1: Model Descriptions and Output

Model Name	Data Set Description	Training Accuracy	Testing Accuracy	Training Loss	Testing Loss	Explanation Of Classification Report
Basic model with one Embedding,LSTM and Dense layer without weight matrix	Top 5	96.50%	90.93%	0.0675	0.3368	If we see the classification report we can see the f1 score came poor for 5th highest group among top 5 but due to large overfit we cannot select this model.
Base model with glove 100d weight matrix	Top 5	94.43%	91.66%	0.1159	0.2393	If we see the classification report we can see the f1 score came bad for the 5th highest groups and also due to large overfit we cannot select this model.

Model Name	Data Set Description	Training Accuracy	Testing Accuracy	Training Loss	Testing Loss	Explanation Of Classification Report
One layer bidirectional LSTM deep learning model	Top 5	94.16%	91.23%	0.1264	0.2509	we can see the f1 score came poor for 5th highest group among top 5 along with that due to large overfit we cannot select this model.
One layer bidirectional LSTM deep learning model with flatten and return sequence as True	Top 5	95.03%	90.25%	0.1043	0.3339	we can see the f1 score came low for 5th highest group and also due to large overfit we cannot select this model
One layer bidirectional LSTM deep learning model with GlobalMaxPool1D and return sequence as True	Top 5	96.00%	90.01%	0.0814	0.2896	If we see the classification report we can see the f1 score came not well for 4th and 5th highest group and due to large overfit we cannot select this model
Two layers bidirectional LSTM deep learning model with return sequence as True and glove 100 embedding and 1 dense layer and 1 flatten layer	Top 5	95.82%	89.76%	0.0862	0.404	If we see the classification report we can see the f1 score came bad for 5th highest group and due to large overfit we cannot select this model.
Two layers bidirectional LSTM deep learning model with return sequence as True and glove 100 embedding and 1 dense layer and 1 flatten layer ,1 drop out layer and 1 spatial drop out layer	Top 5	93.56%	90.62%	0.1467	0.2717	If we see the classification report we can see F1 is not very good but it has less over fit compare to others.So after comparing all we will select this model

Image 8.2.2: Comparison Of accuracy, losses and classification report

9.Model Descriptions of LSTM variations with Glove 300D

Now we will use glove 6b 300d weight matrix to observe and compare performance.

9.1. Models with Complete data set

Sno.	Model Name	Model Description	Model Output
1	Base model with 300D glove embedding	This model has 1 embedding layer with glove 300d weight matrix, 1 LSTM layer with output 128 and return sequence as false and one dense layer with output 74 Loss we have used catagorical_crossentropy and optimizer is adam and metrics is accuracy	We can observe that there are 3,757,194 trainable parameters and 0 nontrainable parameters in model. The accuracy after 12 epochs comes with 91% in train and 66% in test which has a very large overfit and too less training loss compare to validation loss. The error plot also explains the overfit.
2	One layer bidirectional LSTM deep learning model	This model has 1 embedding layer with glove 300d weight matrix, 1 Bi directional LSTM layer with output 128 and return sequence as false and one dense layer with output 74 with soft max in dense layer as activation Loss we have used catagorical_crossentropy and optimizer is adam and metrics is accuracy	We can observe that there are 3,986,314 trainable parameters and 0 nontrainable parameters in model. The accuracy after 12 epochs comes with 90% in train and 65% in test which has a very large overfit and too less training loss compare to validation loss.
3	One layer bidirectional LSTM deep learning model with flatten and return sequence as True and glove 300 embedding	This model has 1 embedding layer with glov 100d 1 bi directional LSTM layer with output 128 and return sequence as true, one flatten layer and one dense layer with output 74 Loss we have used catagorical_crossentropy and optimizer is adam and metrics is accuracy	We can observe that there are 16,663,582 trainable parameters and 0 nontrainable parameters in model. The accuracy after 7 epochs comes with 91% in train and 64% in test which has a very large overfit and too less training loss compare to validation loss.

Sno.	Model Name	Model Description	Model Output
4	One layer bidirectional LSTM deep learning model with GlobalMaxPool1D and return sequence as True	This model has 1 embedding layer with glove 300d embedding and 1 bidirectional LSTM layer with output 128 and return sequence as true and one globalmaxpool1D layer and 3 dense layers with final layer having output as 74 Loss we have used catagorical_crossentropy and optimizer is adam and metrics is accuracy	We can observe that there are 4,007,070 trainable parameters and 0 nontrainable parameters in model. The accuracy after 9 epochs with early stopping comes with 90% in train and 63% in test which has a very large overfit and too less training loss compare to validation loss.
5	Two layers bidirectional LSTM deep learning model with return sequence as True and glove 300 embedding and 1 dense layer and 1 flatten layer ,1 drop out layer and 1 spatial drop out layer	This model has 1 embedding layer 2 bidirectional LSTM layers with weight matrix as glove 300 and return sequence as True and one flatten and one dense layer with 1 drop out layer and 1 spatial drop out layer . Drop out we have used 0.3 and spatial drop out as 0.4 Loss we have used catagorical_crossentropy and optimizer is adam and metrics is accuracy	We can observe that there are 17,110,922 trainable parameters and 0 nontrainable parameters in model. The accuracy after 9 epochs comes with 84% in train and 63% in test which has a less overfit compare to all other models and one of the comparatively well performance we got so far in 300d.Though there is high imbalance

Image 9.1.1: Model Descriptions and Output

Model Name	Data Set Description	Training Accuracy	Testing Accuracy	Training Loss	Testing Loss	Explanation Of Classification Report
Base model with 300D glove embedding	Complete	91.04%	66.00%	0.3233	1.5717	Classification report depicts that the scores are not good we will try to improve more.The classification report did not went well due to hudge imbalance in complete data
One layer bidirectional LSTM deep learning model	Complete	90.82%	65.80%	0.3393	1.5019	The classification Report depicts the f1 score came poor for many groups among 74 along with that due to large overfit we cannot select this model.
One layer bidirectional LSTM deep learning model with GlobalMaxPool1D and return sequence as True	Complete	90.86%	63.61%	0.3297	1.7712	The classification Report depicts the f1 score came poor for many groups among 74 along with that due to large overfit we cannot select this model.
One layer bidirectional LSTM deep learning model with flatten and return sequence as True and glove 300 embedding	Complete	91.73%	64.12%	0.3256	1.8768	The classification Report depicts the f1 score came poor for many groups among 74 along with that due to large overfit we cannot select this model.
Two layers bidirectional LSTM deep learning model with return sequence as True and glove 100 embedding and 1 dense layer and 1 flatten layer ,1 drop out layer and 1 spatial drop out layer	Complete	84.24%	63.61%	0.5168	1.8593	The classification Report depicts the f1 score came poor for many groups among 74 but the overall performance is best among all models fr 300 d we have tried so far on complete data set

Image 9.1.2: Comparison Of accuracy, losses and classification report

9.2. Models with Top 5 Group data

Sno.	Model Name	Model Description	Model Output
1	Base model with 300D glove embedding	This model has 1 embedding layer with glove 300d weight matrix, 1 LSTM layer with output 128 and return sequence as false and one dense layer with output 5 as 5 groups are there for prediction with soft max in dense layer as activation Loss we have used catagorical_crossentropy and optimizer is adam and metrics is accuracy	We can observe that there are 2,243,193 trainable parameters and 0 nontrainable parameters in model. The accuracy after 5 epochs comes with 95% in train and 90% in test which has overfit but compare to others except model 9 the performance is good.
2	One layer bidirectional LSTM deep learning model	This model has 1 embedding layer with glove 300d weight matrix, 1 Bi directional LSTM layer with output 128 and return sequence as false and one dense layer with output 5 as 5 groups are there for prediction with soft max in dense layer as activation Loss we have used catagorical_crossentropy and optimizer is adam and metrics is accuracy	We can observe that there are 2,463,481 trainable parameters and 0 nontrainable parameters in model. The accuracy after 7 epochs comes with 95% in train and 91% in test which has a larger overfit and to less training loss compare to validation loss.
3	One layer bidirectional LSTM deep learning model with GlobalMaxPool1D and return sequence as True	This model has 1 embedding layer with glove 300d embedding and 1 bidirectional LSTM layer with output 128 and return sequence as true and one globalmaxpool1D layer and 3 dense layers with final layer having output as 5 as 5 groups are there for prediction. Loss we have used catagorical_crossentropy and optimizer is adam and metrics is accuracy	We can observe that there are 2,496,381 trainable parameters and 0 nontrainable parameters in model. The accuracy after 6 epochs with early stopping comes with 96% in train and 90% in test
4	One layer bidirectional LSTM deep learning model with return sequence as True and glove 300 embedding and 1 dense layer and 1 flatten layer and 1 dropout	This model has 1 embedding layer 1 bidirectional LSTM layers with weight matrix as glove 300 and return sequence as True and one flatten ,one drop out and one dense layer with output 5 as 5 groups are there for prediction. Loss we have used catagorical_crossentropy and optimizer is adam and metrics is accuracy	We can observe that there are 2,695,705 trainable parameters and 0 nontrainable parameters in model. The accuracy after 8 epochs comes with 96% in train and 90% in test which has a larger overfit and to less training loss compare to validation loss.

Sno.	Model Name	Model Description	Model Output
5	Two layers bidirectional LSTM deep learning model with one has return sequence as True second one has false and glove 300 embedding	This model has 1 embedding layer with glov 300d 2 bi directional LSTM layers with one with return sequence as true and second is return sequence as false and one dense layer with output 5 as 5 groups are there for prediction. Loss we have used catagorical_crossentropy and optimizer is adam and metrics is accuracy	We can observe that there are 2,857,721 trainable parameters and 0 nontrainable parameters in model. The accuracy after 4 epochs with early stopping comes with 94% in train and 89% in test which has a larger overfit and to less training loss compare to validation loss.
6	Two layers bidirectional LSTM deep learning model with return sequence as True and glove 300 embedding and 1 dense layer and 1 flatten layer ,1 drop out layer and 1 spatial drop out layer	This model has 1 embedding layer 2 bidirectional LSTM layers with weight matrix as glove 300 and return sequence as True and one flatten and one dense layer with 1 drop out layer and 1 spatial drop out layer . Drop out we have used 0.3 and spatial drop out as 0.4 Loss we have used catagorical_crossentropy and optimizer is adam and metrics is accuracy	We can observe that there are 3,307,001 trainable parameters and 0 nontrainable parameters in model. The accuracy after 6 epochs comes with 94% in train and 91% in test which has a overfit.

Image 9.2.1: Model Descriptions and Output

Model Name	Data Set Description	Training Accuracy	Testing Accuracy	Training Loss	Testing Loss	Explanatin Of Classification Report
Base model with 300D glove embedding	Top 5	95.01%	90.74%	0.1296	0.2732	Classification Report depicts that the F1 score came quite well except 5th highest group
One layer bidirectional LSTM deep learning model	Top 5	95.35%	91.23%	0.1031	0.2412	The classification Report depicts the f1 score came poor for 5th highest group among top 5 along with that due to large overfit we cannot select this
One layer bidirectional LSTM deep learning model with GlobalMaxPool1D and return sequence as True	Top 5	95.35%	91.23%	0.1031	0.2412	The classification Report depicts the f1 score came poor for 5th highest group among top 5 along with that due to large overfit we cannot select this
One layer bidirectional LSTM deep learning model with return sequence as True and glove 300 embedding and 1 dense layer and 1 flatten layer and 1 dropout	Top 5	96.06%	90.56%	0.0801	0.3246	Classification reports depicts that 5th and 4th highest group have low F1 score and the model behaviour is fluctuating which is a concern.
Two layers bidirectional LSTM deep learning model with one has return sequence as True second one has false and glove 300 embedding	Top 5	94.58%	89.03%	0.1363	0.3131	The classification Report depicts the f1 score came poor for 5th highest group and 2nd highest group among top 5 along with that due to large overfit we cannot select this model
Two layers bidirectional LSTM deep learning model with return sequence as True and glove 300 embedding and 1 dense layer and 1 flatten layer ,1 drop out layer and 1 spatial drop out layer	Top 5	94.85%	91.11%	0.1132	0.389	The classification Report depicts the f1 score came very poor for 5th highest group among top 5 along with that due to large overfit we cannot select this model

Image 9.2.2: Comparison Of accuracy, losses and classification report

9.3. Best Bidirectional Model

Model Description:

This model has 1 embedding layer 2 bidirectional LSTM layers with weight matrix as glove 100 and return sequence as True and one flatten and one dense layer with 1 drop out layer and 1 spatial drop out layer.

Drop out we have used 0.3 and spatial drop out as 0.4

Loss we have used catagorical_crossentropy and optimizer is Adam and metrics is accuracy.

```
model009 = Sequential()
model009.add(Embedding(vocab_size, embedding_dim, weights = [embedding_matrix],input_length = 352,trainable=True))
model009.add(SpatialDropout1D(0.4))
model009.add(Bidirectional(LSTM(128, return_sequences=True)))
model009.add(Dropout(0.3))
model009.add(Bidirectional(LSTM(128, return_sequences=True)))
model009.add(Flatten())
model009.add(Dense(5,activation='softmax'))
model009.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
print(model009.summary())
```

```
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_9 (Embedding)      (None, 352, 100)          674300
_________________________________________________________________
spatial_dropout1d_1 (Spatial (None, 352, 100)          0
_________________________________________________________________
bidirectional_9 (Bidirection (None, 352, 256)          234496
_________________________________________________________________
dropout_2 (Dropout)          (None, 352, 256)          0
_________________________________________________________________
bidirectional_10 (Bidirectio (None, 352, 256)          394240
_________________________________________________________________
flatten_4 (Flatten)          (None, 90112)             0
_________________________________________________________________
dense_11 (Dense)             (None, 5)                 450565
=================================================================
Total params: 1,753,601
Trainable params: 1,753,601
Non-trainable params: 0
_________________________________________________________________
None
```

Image 9.3.1: Model parameters and architecture

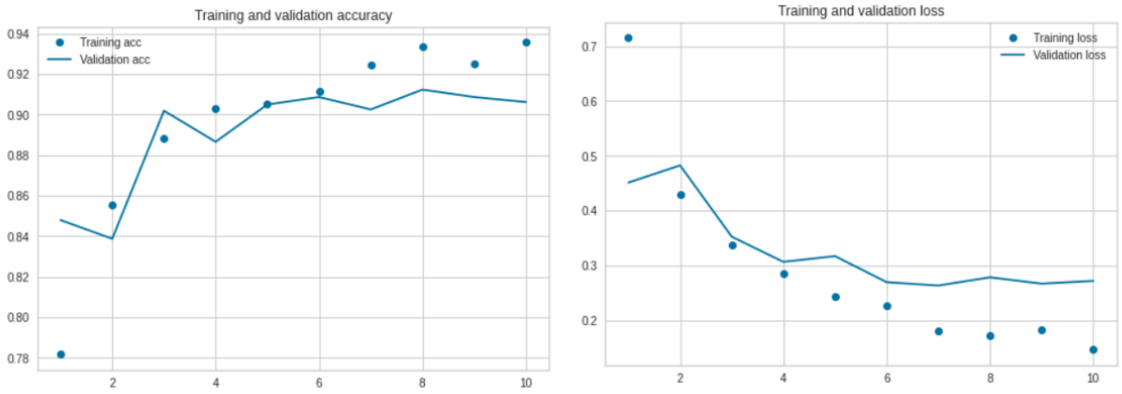


Image 9.3.2: Error plot

```
Epoch 9/10
39/39 [==============================] - 5s 136ms/step - loss: 0.1823 - accuracy: 0.9253 - val_loss: 0.2667 - val_accuracy: 0.9
086
Epoch 10/10
39/39 [==============================] - 5s 136ms/step - loss: 0.1467 - accuracy: 0.9356 - val_loss: 0.2717 - val_accuracy: 0.9
062
```

```
features=["usa file server hostname fail hard drive"]
testval = tokenizer.texts_to_sequences(features)
padded_texts = pad_sequences(testval, maxlen=352)
print(testval)
pred = modelchk.predict(padded_texts)
print (pred)
list=['Group_0','Group_12','Group_24','Group_8','Group_9']
print(list[np.argmax(pred)])
```

```
[[81, 120, 71, 38, 16, 237, 125]]
[[1.1758213e-02 9.4666022e-01 1.7867922e-05 3.8927320e-02 2.6363195e-03]]
Group_12
```

	precision	recall	f1-score	support
0	0.99	0.99	0.99	1194
1	0.86	0.93	0.90	74
2	0.98	0.99	0.98	90
3	0.97	0.64	0.77	200
4	0.48	0.88	0.62	73
micro avg	0.94	0.94	0.94	1631
macro avg	0.86	0.88	0.85	1631
weighted avg	0.96	0.94	0.94	1631
samples avg	0.94	0.94	0.94	1631

```
finalticketsummary.loc[finalticketsummary['CleanTextAfterRemoveStopword']=="usa file server hostname fail hard drive"]
```

	Assignment group	CleanTextAfterRemoveStopword	tokenizesummaryafterstopword	Assignment group_num	WordLength
493	GRP_12	usa file server hostname fail hard drive	[usa, file, server, hostname, fail, hard, drive]	1	7

Image 9.3.3: Final Test and train accuracy and classification report and prediction

We can observe that that this model has 93% train and 91% approx. test accuracy which is really good with less overfit. So, we are good to select this model.

10.Pretrained Models

Pre-trained models help data scientists start off on a new problem by providing an existing framework they can leverage. These pre-trained models have proven to be truly effective and useful. The introduction of transfer learning and pretrained language models in natural language processing (NLP) pushed forward the limits of language understanding and generation. Transfer learning and applying transformers to different downstream NLP tasks have become the main trend of the latest research advances.

10.1. Universal Language Model Fine-Tuning (ULMFiT)

Proposed by fast.ai's Jeremy Howard and NUI Galway Insight Center's Sebastian Ruder, ULMFiT is essentially a method to enable transfer learning for any NLP task and achieve great results without having to train models from scratch. It uses the process of inductive Transfer Learning which tackles exactly the challenges of memory and time and leads to the central concept ULMFiT is based on.

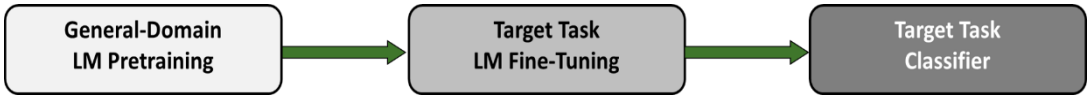


Image 10.1.1: Overview ULMFiT

- General-Domain LM Pretraining: In a first step, a LM is pretrained on a large general-domain corpus (in our case the WikiText-103 dataset). Now, the model is able to predict the next word in a sequence (with a certain degree of certainty
- Target Task LM Fine-Tuning: Following the transfer learning approach, the knowledge gained in the first step should be utilized for the target task. However, the target task dataset is likely from a different distribution than the source task dataset. To address this issue, the LM is consequently fine-tuned on the data of the target task.

- Target Task Classifier: Since ultimately, in our case, we do not want our model to predict the next word in a sequence but to assign the ticket to particular group, in a third step the pretrained LM is designed such a way that the final output is a probability distribution over the top 5 or all 74 groups.

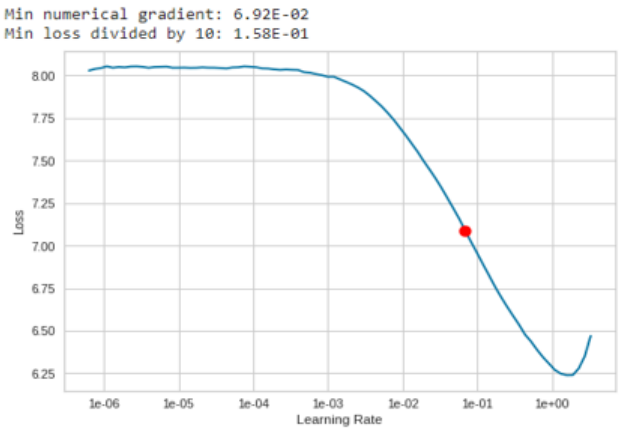
10.1.1. ULM Fit model Top 5 data set

In our data set first we have saved our test and train and full data set to CSV file. Before doing that **there should not be any blank list in final data set if that is there at the time of importing CSV we will get NA values** so we need to drop the empty list in independent variable and save it to CSV. Now we will use Text List which is part of the data block instead of using the factory methods of TextClasDataBunch and TextLMDataBunch because Text List is part of the API which is more flexible and powerful. We can use the data_lm to fine-tune a pre-trained language model. We can create a learner object, 'learn', that will directly create a model, download the pre-trained weights, and be ready for fine-tuning.

```
data_lm = (TextList.from_csv('/content/drive/My Drive/','finaldf.csv', cols='CleanTextAfterRemoveStopword')
    .split_by_rand_pct(0.3)
    .label_for_lm()
    .databunch(bs=48))
```

Image 10.1.1.1: data_lm object

Now we will fine-tune our model with the **weights of a model pre-trained on a larger corpus, Wikitext 103**. This model has been trained to predict the next word in the sentence provided to it as an input. Next, we will find the optimal learning rate & visualize it. The visualization will help us to spot the range of learning rates & choose from while training our model.



epoch	train_loss	valid_loss	accuracy	time
0	3.575911	3.583309	0.383503	00:01
1	3.394986	3.379483	0.412883	00:01
2	3.019526	3.362095	0.426020	00:01
3	2.538084	3.502287	0.430952	00:01
4	2.070207	3.685699	0.425043	00:01

```
CPU times: user 6.13 s, sys: 3.31 s, total: 9.44 s
Wall time: 9.47 s
```

Image 10.1.1.2: LR Plot for Language model and Result of language model after 4 Epochs

By default, the Learner object is frozen thus we need to train the embeddings at first. We will run it for 4 epochs to see how accuracy varies. The learning rate is picked with the help of the plot we got above.

We can observe that the accuracy improved slightly but still looming in the same range. This is because firstly the model was trained on a pre-trained model with different vocabulary & secondly, there were no labels, we had passed the data without specifying the labels.

Now, we will create a new data object that only grabs the labeled data and keeps those labels.

```
data_clas=(TextList.from_csv('/content/drive/My Drive/','finaldf.csv', cols='CleanTextAfterRemoveStopword',vocab=data_lm.vocab)
      .split_by_rand_pct(0.3)
      .label_from_df('Assignment group')
      .add_test(TextList.from_csv('/content/drive/My Drive/','val.csv', vocab=data_lm.vocab))
      .databunch(bs=48))
```

Image 10.1.1.3. data_clas object

The classifier needs a little less dropout, so we pass drop_mult=0.5 to multiply all the dropouts by this amount. We will not load the pre-trained model, but instead our fine-tuned encoder will be loaded.

```
[ ]   learn.save_encoder('ticket_class_enc')
```

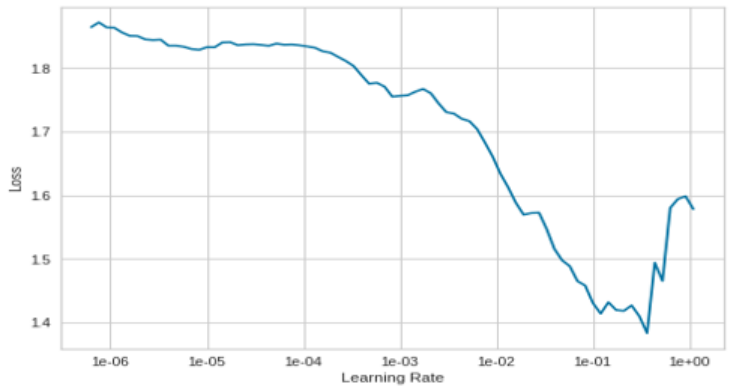
```
learn_clas = text_classifier_learner(data_clas, AWD_LSTM,drop_mult=0.5)
learn_clas.load_encoder('ticket_class_enc')
```

```
RNNLearner(data=TextClasDataBunch;

Train: LabelList (3804 items)
x: TextList
xxbos outlook receive hello team meeting skype meeting etc not appear calendar xxunk please
y: CategoryList
GRP_0,GRP_0,GRP_0,GRP_0,GRP_0
Path: /content/drive/My Drive;

Valid: LabelList (1630 items)
x: TextList
xxbos user unable tologin vpn connect user system use teamviewer help login company vpn lin
y: CategoryList
GRP_0,GRP_0,GRP_0,GRP_0,GRP_0
Path: /content/drive/My Drive;

Test: LabelList (1631 items)
x: TextList
xxbos xxup xxunk,xxbos xxup xxunk,xxbos xxup xxunk,xxbos xxup xxunk,xxbos xxup xxunk
```

Image 10.1.1.4: Classifier Model

Now we will plot the LR curve for classifier model.

```
## plot lr finder
learn_clas.recorder.plot()
```



epoch	train_loss	valid_loss	accuracy	time
0	0.639998	0.271598	0.907975	00:02

Image 10.1.1.5: LR plot for classifier model and Accuracy without unfreezing

Here we can see that the accuracy has drastically improved if we compare with the Language model in step 1 when we provide labels. Now we will partially train the model by unfreezing one layer at a time & differential learning rate. Here we will be using **the slice** attribute which will divide the specified learning rates among 3 groups of models.

```
learn_clas.freeze_to(-2) # unfreeze last 2 layers
learn_clas.fit_one_cycle(1, slice(1e-2/(2.6**4),1e-2), moms=(0.8,0.7))
```

epoch	train_loss	valid_loss	accuracy	time
0	0.434595	0.271365	0.903681	00:02

Image 10.1.1.6: Accuracy after first un freezing

Finally, after few steps we will unfreeze the whole model & visualize the learning rate to choose & use that for final training.

```
learn_clas.unfreeze() # unfreze all
learn_clas.fit_one_cycle(3, slice(1e-3/(2.6**4),1e-3), moms=(0.8,0.7))
```

epoch	train_loss	valid_loss	accuracy	time
0	0.206599	0.218034	0.925153	00:03
1	0.177014	0.203282	0.923313	00:04
2	0.177177	0.203070	0.926994	00:03

Image 10.1.1.7: Accuracy after full un freezing

10.1.2. Observations on ULM Fit model Top 5 data set

2	GRP_0	tensor(0)	[tensor(0.9922), tensor(0.0042), tensor(0.0007...	customer master receive hello help customer ma...
3	GRP_8	tensor(3)	[tensor(0.0007), tensor(0.0020), tensor(6.6138...	job job fail job scheduler receive job
4	GRP_12	tensor(1)	[tensor(0.0094), tensor(0.9763), tensor(0.0033...	system bankrd temperature alert monitor tool s...

Image 10.1.2.1: Prediction

```
print("Train Accuracy: ", accuracy_score(train['Assignment group'], train['pred_grp']))
```

```
Train Accuracy:  0.9503023928477518
```

```
report = classification_report(val['Assignment group'], val['pred_grp'])
print(report)
```

```
              precision    recall  f1-score   support

       GRP_0       0.99      0.99      0.99      1206
      GRP_12       0.86      0.83      0.85        77
      GRP_24       0.95      0.93      0.94        92
       GRP_8       0.82      0.94      0.87       186
       GRP_9       0.80      0.47      0.59        70

    accuracy                           0.95      1631
   macro avg       0.88      0.83      0.85      1631
weighted avg       0.95      0.95      0.95      1631
```

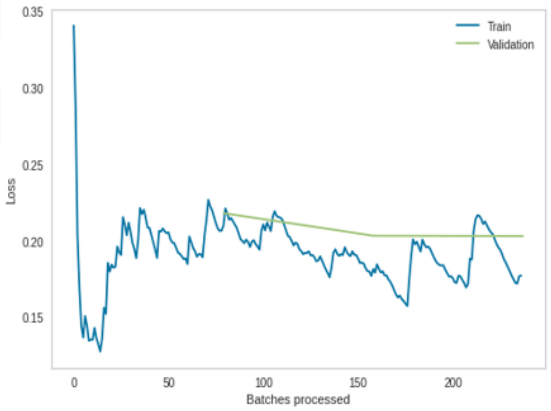


Image 10.1.2.2: Train accuracy and classification report and loss curve

So, we can observe that for **top 5 models the accuracy and losses came very good** and we will select this model. The test accuracy **comes around 92% and training accuracy as 95%.** If we see the classification report, we can observe that **the precision score is good for all the models but the recall and F1 is poor for 5th highest group**. If we also observe the loss curve, we can see that **the validation loss gets saturated** after a certain time.

10.1.3. ULM Fit model Complete data set

The same steps we have applied for complete data set and below is the snapshot of final outcome.

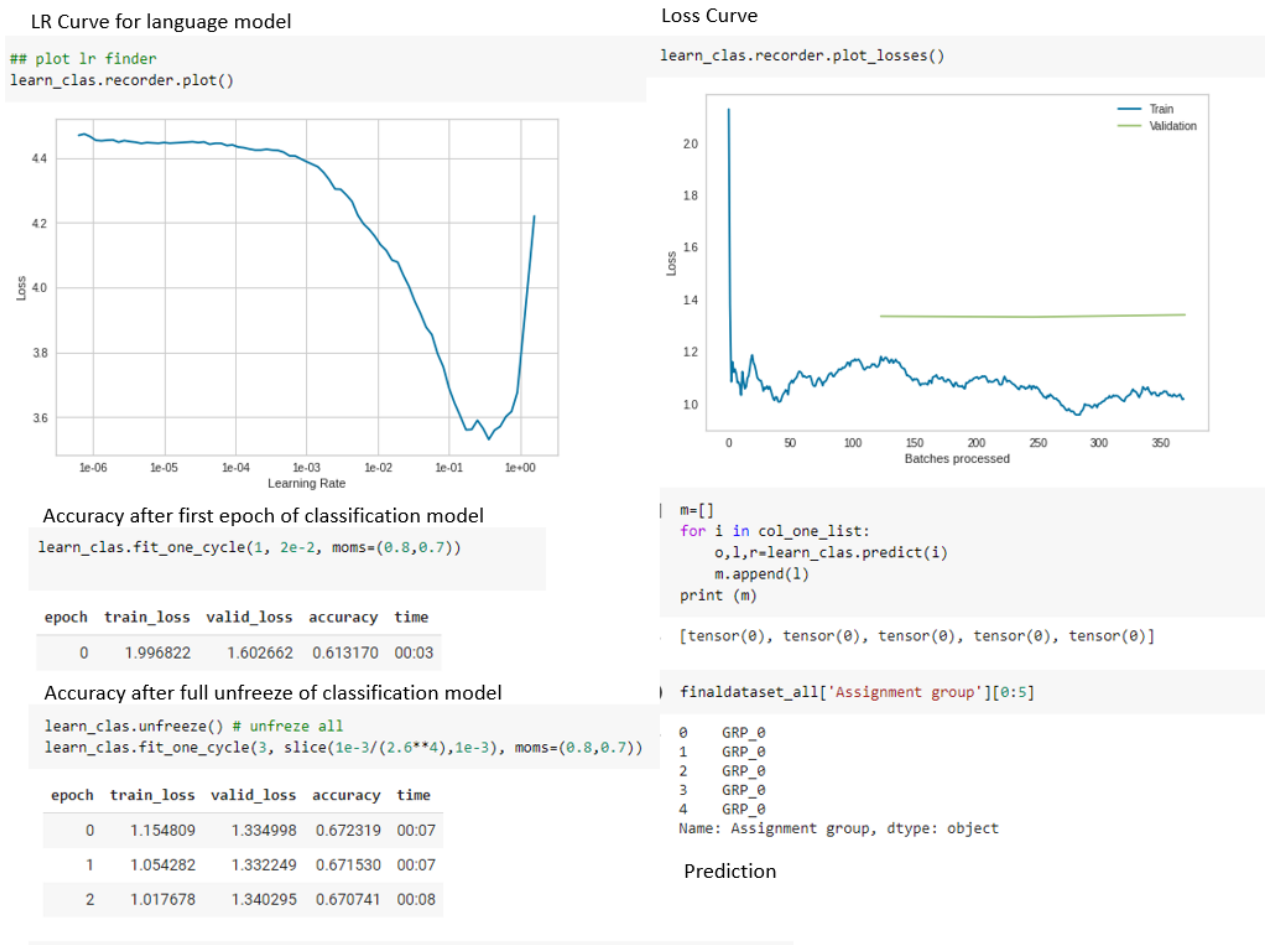


Image 10.1.3: Snap shot of complete model

Here we can observe that the model did not work very well on complete data set due to imbalance and losses are very high and the **test accuracy is 67%** but if we compare with other models for complete data set, we can say this model **performance is better** than other models on complete data set.

10.2. FastText

FastText is an open-source, free, lightweight library released by face book that allows users to learn text representations and text classifiers. It is a library for efficient learning of word representations

and sentence classification. It is written in C++ and supports multiprocessing during training. Before we jump upon the execution, there is a word of caution about the training file. **The default format of text file on which we want to train our model should be _ _ label _ _ <X> <Text>. Where _ _label_ _ is a prefix to the class and <X> is the class assigned to the document**. Also, there should not be quotes around the document and everything in one document should be on one line. There are few steps we need to perform for FastText execution after installing FastText:

- Train, Test/Input data preparation: As mentioned above we need to split our data frame to test and train and convert into text file with the format be _ _ label _ _ <X> <Text>.
- Model. Fit: We will be using the train.txt file to train the model and test.txt file to predict. We will use fasttext supervised for training and we can specify some parameters during the training time. For example, to specify the learning rate of the training process we can use the argument -lr to specify the learning rate. The other available parameters that can be tuned are:-lr : learning rate [0.1],-lrUpdateRate : change the rate of updates for the learning rate [100],-dim: size of word vectors [100],-ws: size of the context window [5],-epoch: number of epochs [5],-neg: number of negatives sampled [5],-loss: loss function {ns, hs, softmax} [ns],-thread: number of threads [12],-pretrainedVectors : pretrained word vectors for supervised learning []-saveOutput : whether output params should be saved [0]The values in the square brackets [] represent the default values of the parameters passed.
- Model. Test to check performance metrices and predict the new inputs.

10.2.1. FastText model Top 5 data set

As discussed earlier we will prepare our training and testing file in the format which fast text understands.

```
with open("/content/drive/My Drive/test.txt", "w") as test_file_handler:
    for X_test_entry, y_test_entry in zip(X_test, y_test):
        line_to_write = "__label__" + str(y_test_entry) + "\t" + str(X_test_entry) + "\n"
        try:
            test_file_handler.write(line_to_write)
        except:
            print(line_to_write)
            break
```

Image 10.2.1.1: Test.txt file

```
model = fasttext.train_supervised(input="/content/drive/My Drive/train.txt",epoch=30)
```

Train the model

```
def print_results(N, p, r):
    print("N\t" + str(N))
    print("P@{}\t{:.3f}".format(1, p))
    print("R@{}\t{:.3f}".format(1, r))

results = model.test("/content/drive/My Drive/test.txt")
print_results(*results)
```

```
N       544
P@1     0.921
R@1     0.921
```

```
model.predict("job job fail job scheduler receive job",2)
```

```
(('__label__GRP_8', '__label__GRP_9'), array([0.566953, 0.409367]))
```

```
model.save_model("/content/drive/My Drive/modelfasttext.bin")
```

```
model.predict("login disconnect")
```

```
(('__label__GRP_0',), array([1.000009]))
```

Model Performance and Prediction

Image 10.2.1.2: Overview of FastText top 5 model prediction and performance

Now to find the accuracy of train and test we will create two data frames having predicted column and actual column. Now as predicted column will come in _ _ label _ _ <X> <Text> format so we will apply some regular expressions and label encode the predicted and actual Assignment Group value to calculate accuracy.

	Pred	Confidence	CleanTextAfterRemoveStopword	Actual
0	(__label__GRP_0,)	[0.9410046935081482]	erp runtime error I eventually see receive goo...	GRP_0
1	(__label__GRP_12,)	[0.7766759991645813]	issue user profile hostname seem mess server n...	GRP_12
2	(__label__GRP_0,)	[1.0000067949295044]	blank call	GRP_0
3	(__label__GRP_0,)	[1.0000098943710327]	need see unread email	GRP_0
4	(__label__GRP_24,)	[0.9947632551193237]	calculator steli measuring device not work jio...	GRP_24

Image 10.2.1.3: Predicted Data frame for test set

```
def text_preprocessing_newt(text):
    strText = str(text)
    cleanText = re.sub(r'[^a-zA-Z0-9\s]',' ',strText)
    #cleanText = re.sub(r'[label\s]',' ',cleanText)
    return cleanText
```

```
df_pred_result["Prediction"]=df_pred_result["Pred"].apply(text_preprocessing_newt)
```

```
# label_encoder object knows how to understand word labels.
label_encoder = preprocessing.LabelEncoder()
# Encode labels in column 'species'.
df_pred_result["Prediction"]= label_encoder.fit_transform(df_pred_result["Prediction"])
df_pred_result["Act_acc"]= label_encoder.fit_transform(df_pred_result["Actual"])
```

Image 10.2.1.4: Prediction label creation for calculating accuracy for test set

```
print('Test Accuracy:')
print(accuracy_score(df_pred_result['Act_acc'], df_pred_result['Prediction']))
```

```
Test Accuracy:
0.9209558823529411
```

```
report = classification_report(df_pred_result['Act_acc'], df_pred_result['Prediction'])
print(report)
```

```
              precision    recall  f1-score   support

           0       0.96      0.98      0.97       398
           1       0.79      0.73      0.76        26
           2       0.96      0.79      0.87        29
           3       0.78      0.94      0.85        66
           4       0.64      0.28      0.39        25

    accuracy                           0.92       544
   macro avg       0.82      0.74      0.77       544
weighted avg       0.92      0.92      0.91       544
```

Image 10.2.1.5: Accuracy Calculation and Classification Report for test set

```
def print_results(N, p, r):
    print("N\t" + str(N))
    print("P@{}\t{:.3f}".format(1, p))
    print("R@{}\t{:.3f}".format(1, r))

results = model.test("/content/drive/My Drive/train.txt")
print_results(*results)
```

```
N       4891
P@1     0.969
R@1     0.969
```

```
print('Train Accuracy:')
print(accuracy_score(df_pred_train['Act_acc'], df_pred_train['Prediction']))
```

```
Train Accuracy:
0.9687180535677775
```

Image 10.2.1.6: Accuracy and scores for Train data

So, we can observe that **for top 5 models the accuracy came good test as 92% and train has 96% but overfit** is there. If we see the classification report, we can observe that **the scores are moderate but the recall and F1 is poor for 5th highest group**.

10.2.2. Fast Text model Complete data set

The same steps we have applied for complete data set and below is the snapshot of final outcome.

Train model and scores

```
[ ] model = fasttext.train_supervised(input="/content/drive/My Drive/train_ALL.txt",epoch=30,lr=0.5)
```

7.12.Model performance and Prediction

```
[ ] def print_results(N, p, r):
        print("N\t" + str(N))
        print("P@{}\t{:.3f}".format(1, p))
        print("R@{}\t{:.3f}".format(1, r))

    results = model.test("/content/drive/My Drive/test_ALL.txt")
    print_results(*results)
```

```
N       849
P@1     0.653
R@1     0.653
```

```
def print_results(N, p, r):
    print("N\t" + str(N))
    print("P@{}\t{:.3f}".format(1, p))
    print("R@{}\t{:.3f}".format(1, r))

results = model.test("/content/drive/My Drive/train_ALL.txt")
print_results(*results)
```

```
N       7650
P@1     0.944
R@1     0.944
```

Test model and scores

	Pred	Confidence	CleanTextAfterRemoveStopword	Actual
0	(__label__GRP_2,)	[0.7038630247116089]	pls unlock reset window erp account user vvtgh...	GRP_0
1	(__label__GRP_30,)	[0.4917439818382263]	office excel powerpoint	GRP_31
2	(__label__GRP_16,)	[0.9886095523834229]	need access znqcljxt azvoespk email	GRP_26
3	(__label__GRP_9,)	[0.9034497737884521]	dob report show blank datum yzugpdco nsyapewg ...	GRP_9
4	(__label__GRP_19,)	[0.9474633932113647]	outlook issue receive unable open mail please ...	GRP_0

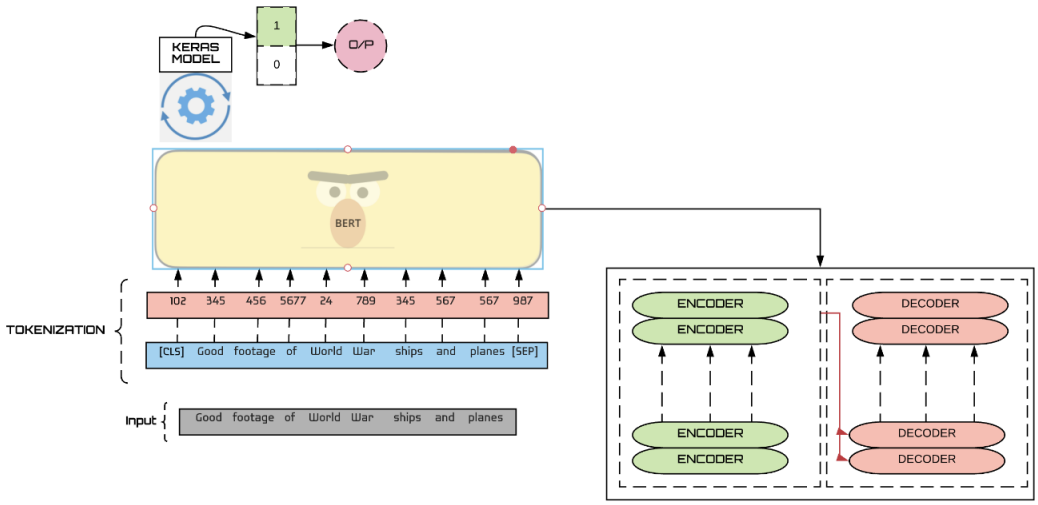
Image 10.2.2.1: Train and Test scores for complete data set and test data prediction

So, we can observe that for full data set the performance is not satisfactory due to high imbalance in group assignment**. The Training precision came as 94% and test as 65% which has high overfit.** As discussed earlier the complete data set is highly imbalanced so the model behaviors are not good for complete data set but for Top5 the result is good and the scores are highly improved.

10.3. BERT

BERT is a technique for NLP pre-training, developed by Google. It utilizes the Transformer, a novel neural network architecture that's based on a self-attention mechanism for language understanding. It was developed to address the problem of sequence transduction or neural machine translation. That means, it suits best for any task that transforms an input sequence to an

output sequence, such as speech recognition, text-to-speech transformation, etc. In its vanilla form, the transformer includes two separate mechanisms: an encoder (which reads the text input) and a decoder (which produces a prediction for the task). The goal of the BERT mechanism is to generate a language model. Thus, only the encoder mechanism is necessary.



BERT Architecture

10.3.1. BERT model 1 for Top 5 data set

Prerequisites: **Switch to GPU to avoid large training time and notebook crash. Install transformers. The text column is preprocessed and the dataset is pickled for using in pretrained models**. For the first model first create sentence and labels then to feed our text to BERT, it must be split into tokens, and then these tokens must be mapped to their index in the tokenizer vocabulary. The tokenization must be performed by the tokenizer included with BERT. We will be using the "uncased". Then with the help of tokenizer.encode function we will combine multiple steps for us. Those are:

- Split the sentence into tokens.
- Add the special [CLS] and [SEP] tokens.
- Map the tokens to their IDs.

This function can perform truncating but does not handle padding. Now we calculate the max length of words and we can see it is exceeding BERT'S max limit so we need to do truncation and padding. The attention mask simply makes it explicit which tokens are actual words versus which are padding. **The BERT vocabulary does not use the ID 0, so if a token ID is 0, then it is padding or it is a real token. As we are using pytorch library we will convert our input to tensor dataset to feed the model. There are a few different pre-trained BERT models available. "bert-base-uncased" means the version that has only lowercase letters ("uncased") and is the smaller version of the two ("base" vs "large").** We will use this model for our both the models.

```
model = BertForSequenceClassification.from_pretrained(
    "bert-base-uncased", # Use the 12-layer BERT model, with an uncased vocab.
    num_labels = 5, # The number of output labels--2 for binary classification.
                    # You can increase this for multi-class tasks.
    output_attentions = False, # Whether the model returns attentions weights.
    output_hidden_states = False, # Whether the model returns all hidden-states.
)
# Tell pytorch to run this model on the GPU.
model.cuda()
```

```
    )
    (encoder): BertEncoder(
      (layer): ModuleList(
        (0): BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
          )
```

Image 10.3.1.1: BERT uncased model for Top 5 data

Now we will use learning **rate as 2e-5 and epsilon 1e-8 as** and batch size **as 32** for tuning model. Below is our training loop. For each pass in our loop we have a training phase and a validation phase. At each pass we unpack our data inputs and labels load data onto the GPU for acceleration, clear out the gradients calculated in the previous pass**. In pytorch the gradients accumulate by default unless you explicitly clear them out. Forward pass (feed input data through the network) Backward pass (backpropagation) tell the network to update parameters with optimizer.step() and track variables for monitoring progress**. The final output is given below:

```
======== Epoch 1 / 1 ========
Training...
  Batch    40  of    153.    Elapsed: 0:00:55.
  Batch    80  of    153.    Elapsed: 0:01:49.
  Batch   120  of    153.    Elapsed: 0:02:44.

  Average training loss: 0.20
  Training epcoh took: 0:03:28

Running Validation...
  Accuracy: 0.88
  Validation took: 0:00:08

Training complete!
```

Image 10.3.1.3: BERT model 1 performance in terms of loss and accuracy

So, we can observe **that the validation accuracy is 88%. Though validation loss saturates but we will not consider this model for prediction as there is a huge overfit between test and train data**.

10.3.2. BERT model 2 for Top 5 data set

For this model we will split the train and test data as per target variables with the logic of **stratified split and will use BERT 'uncased' model**. We will tokenize our text with BERT tokenizer and create Encoder data train and validation to generate BERT inputs with attention masks and tokens and label as output.

```
encoded_data_val = tokenizer.batch_encode_plus(
    finaldataset[finaldataset.data_type=='val'].CleanTextAfterRemoveStopword.values,
    add_special_tokens=True,
    return_attention_mask=True,
    pad_to_max_length=True,
    max_length=256,
    return_tensors='pt'
)
input_ids_train = encoded_data_train['input_ids']
attention_masks_train = encoded_data_train['attention_mask']
labels_train = torch.tensor(finaldataset[finaldataset.data_type=='train'].label.values)
```

Image 10.3.2.1: BERT model 2 input preparation

Next, we will create Tensor data input for test and train and define model parameters **as lr=1e-5, eps=1e-8. Then we will load the GPU which we have selected earlier and train our data**.

```
tqdm.write(f'\nEpoch {epoch}')

loss_train_avg = loss_train_total/len(dataloader_train)
tqdm.write(f'Training loss: {loss_train_avg}')

val_loss, predictions, true_vals = evaluate(dataloader_validation)
train_loss, predictions_train, train_vals = evaluate(dataloader_train)
train_acc=accuracy_score_func(predictions_train,train_vals)
val_f1 = f1_score_func(predictions, true_vals)
val_acc=accuracy_score_func(predictions,true_vals)
tqdm.write(f'Validation ACC: {val_acc}')
tqdm.write(f'Training ACC: {train_acc}')
tqdm.write(f'Validation loss: {val_loss}')
tqdm.write(f'F1 Score (Weighted): {val_f1}')
```

```
100%|██████████| 1/1 [06:20<00:00, 380.12s/it]
Epoch 1: 100%|██████████| 1540/1540 [04:37<00:00, 6.03it/s, training_loss=0.003]

Epoch 1
Training loss: 0.3141631266527035
Validation ACC: 0.9068627450980392
Training ACC: 0.9285559644944793
Validation loss: 0.4102697855720608
F1 Score (Weighted): 0.8856655478030426
```

Image 10.3.2.2: BERT model 2 output and scores

So, we can observe that for this model 2 for top 5 data the test accuracy came as 90% and train as 92.8% which is really good and the overfit is very less and also difference between losses are less. So, we can select this model. Now we will check the prediction.

Group 0 prediction

```
[ ]  print(predictions[0])
     print(true_vals[0])
```

```
[ 7.751182  -2.0715246 -2.1298814 -1.9990308 -1.8517716]
0
```

3rd Group Prediction

```
[ ]  print(predictions[12])
     print(true_vals[12])
```

```
[-3.3716004 -1.5942291 -2.098827   4.7248225  1.6038826]
3
```

Image 10.3.2.3: BERT model 2 prediction

10.3.3. BERT model 1 and 2 for complete data set

We have performed the same steps on the complete data set for BERT and below is the snapshot of BERT model performance on complete data set.

Model 1 performance

```
======== Epoch 1 / 2 ========
Training...
  Batch    40  of    240.    Elapsed: 0:00:56.
  Batch    80  of    240.    Elapsed: 0:01:51.
  Batch   120  of    240.    Elapsed: 0:02:47.
  Batch   160  of    240.    Elapsed: 0:03:42.
  Batch   200  of    240.    Elapsed: 0:04:38.

  Average training loss: 0.84
  Training epcoh took: 0:05:32

Running Validation...
  Accuracy: 0.64
  Validation took: 0:00:14

======== Epoch 2 / 2 ========
Training...
  Batch    40  of    240.    Elapsed: 0:00:56.
  Batch    80  of    240.    Elapsed: 0:01:51.
  Batch   120  of    240.    Elapsed: 0:02:47.
  Batch   160  of    240.    Elapsed: 0:03:42.
  Batch   200  of    240.    Elapsed: 0:04:38.

  Average training loss: 0.71
  Training epcoh took: 0:05:32

Running Validation...
  Accuracy: 0.66
  Validation took: 0:00:14

Training complete!
```

Model 2 performance

```
100% 2/2 [1:34:32<00:00, 2836.48s/it]
Epoch 1: 100% 2409/2409 [07:15<00:00, 5.61it/s, training_loss=0.078]

Epoch 1
Training loss: 0.7557756910187806
Validation ACC: 0.6635294117647059
Validation loss: 1.7683448723374922
F1 Score (Weighted): 0.6334294436077801
Epoch 2: 100% 2409/2409 [07:15<00:00, 5.60it/s, training_loss=0.006]

Epoch 2
Training loss: 0.7303190800269541
Validation ACC: 0.6737254901960784
Validation loss: 1.7187363852010629
F1 Score (Weighted): 0.6428281150438785
```

Image 10.3.3: BERT model 2 prediction

So, we can observe that for full data set the performance is not satisfactory due to high imbalance in group assignment and there is a high overfit. **The validation accuracy comes around 67% but if we compare the losses the training loss is low with compare to validation loss.** As discussed earlier also the complete data set is highly imbalanced so the model behaviors are not good for complete data set but for Top5 the result is good and the scores are highly improved.

11. Model Performance comparison among the best final models

Model Name	Training Accuracy	Testing Accuracy
BERT	92%	90.40%
Two layers bidirectional LSTM deep learning model with return sequence as True and glove 100 embedding and 1 dense layer and 1 flatten layer ,1 drop out layer and 1 spatial drop out layer	93%	91%
ULM FIT	95%	92.50%
Fast Text	96%	92%

Image 11: Accuracy Comparison

So far, we got best scores for above models. **Due to some system configuration restriction we have did not able to select BERT for deployment. We have selected our best bidirectional LSTM model for model deployment. In terms of stability ULM fit performed outstanding but BERT and Bidirectional LSTM has lower overfit.**

12. Model Deployment

So far, we have developed many deep learning/ pretrained models, generated predictions on the testing data, and tested the results and we did everything offline. In reality, generating predictions is only part of a machine learning/AI project, although it is the most important. So,the model building will be irrelevant if we will not make a use of it. We already have our best models with us. Now we need to build other components to deploy our model. There are many ways you can deploy your model however we will be deploying our model through webpage.

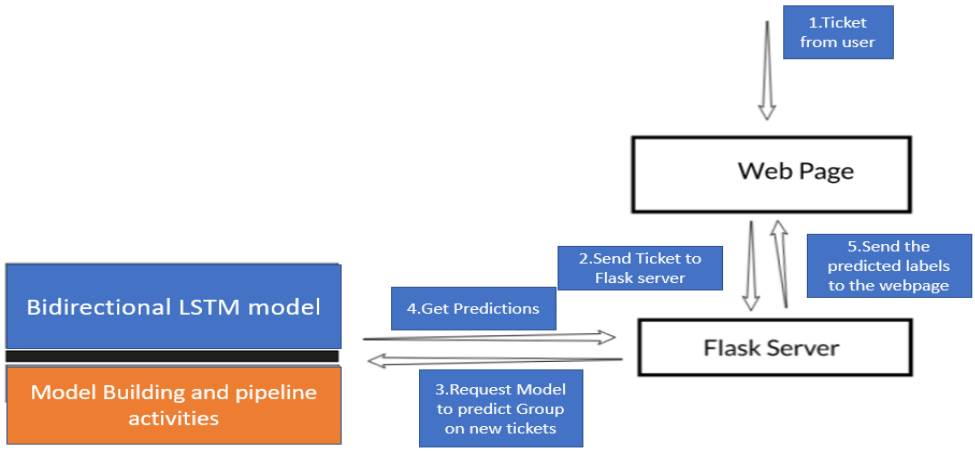


Image 12: Deployment architecture using flask

We will develop a web application that consists of a simple web page with a form field that lets us enter a message. After submitting the message to the web application, it will render it on a new page which gives us a result of Assignment Group. We will go by below steps.

12.1. Folder Structure:

First, we will create a folder for this project. We will explain each file.

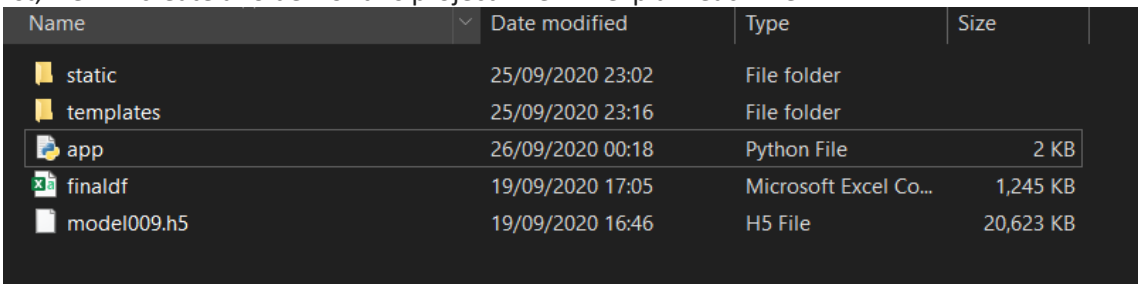


Image 12.1.1: Main Folder structure

So, we have two sub folders, one app.py file, one csv as our top 5 data set and our saved model as model009.h5. We have selected our bidirectional LSTM model for this.

Inside static and template sub folder we have below files:

Static→ template→

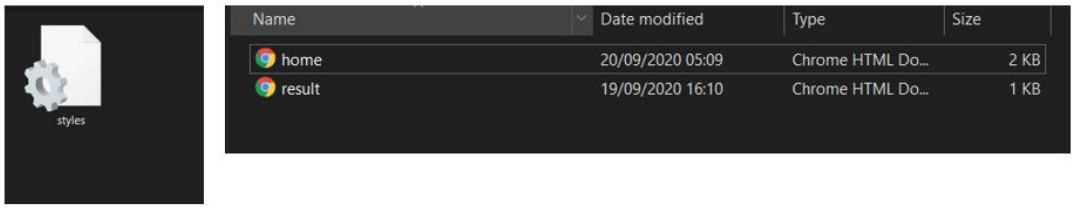


Image 12.1.2: Sub Folder Structure

Static folder comprises of css style file and template folder has home.html which is our first page in web page to receive ticket description as input and result.html will use to provide the predicted output.

12.2. app.py

The app.py file contains the main code that will be executed by the Python interpreter to run the Flask web application

```
app = Flask(__name__)
model1=load_model('model009.h5')
@app.route('/')
def home():
    return render_template('home.html')
@app.route('/predict', methods=['POST'])
def predict():
    features = [x for x in request.form.values()]
    finaldf=pd.read_csv('finaldf.csv')
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(finaldf['CleanTextAfterRemoveStopword'])
    test_final = tokenizer.texts_to_sequences(features)
    padded_texts = pad_sequences(test_final, maxlen=352, value=0.0)
    pred = model1.predict(padded_texts)
    list=['Group_0','Group_12','Group_24','Group_8','Group_9']
    prediction=list[np.argmax(pred)]

    return render_template('result.html', prediction = '{}'.format(prediction))

if __name__ == '__main__':
    app.run(debug=True)
```

Image 12.2.1: app.py code snippet

We will run our application as a single module. Thus, we will initialize a new Flask instance with the argument __name__ to let Flask know that it can find the HTML template folder (templates) in the same directory where it is located. Next, we will load the pre saved model which is model009.h5. Then we will use the route decorator (@app.route('/')) to specify the URL that should trigger the execution of the home function. Our home function will simply render the home.html HTML file, which is located in the templates folder.

Inside the predict function, we will get the vocab inference first from our top 5 data set and tokenize the text and access the padded text, and make predictions, then store the model. We access the new ticket message entered by the user and use our model to make a prediction for its label/group.

We will use the POST method to transport the form data to the server in the message body. Finally, by setting the debug=True argument inside the app.run method, we will further activate Flask's debugger.

Lastly, we will use the run function to only run the application on the server when this script will be directly executed by the Python interpreter it will be ensured using the if statement with __name__ == '__main__'.

12.3. Html files

```
<div class="ml-container">

        <form action="{{ url_for('predict')}}" method="POST">
            <h4 style="color:black;text-align:center;"><i>Enter Your Message Here</i></h4>

            <style>
                    div.justified {
                    display: flex;
                    justify-content: center;
                rows="3" cols="200";
                    }
            </style>

            <div class="justified">
                    <textarea style="border:2px solid black;" name="message" rows="3" cols="100";></textarea>
                <input style="border:2px solid black;text-align:center;" type="submit" class="btn-info" value="Predict">
            </div>

            <br/>


        </form>
```

Image 12.3.1: home.html snippet

```
<header>
    <div class="container">
    <div id="brandname">
        <h1 style="text-align:center;">***Automated Ticket Assignment App with Flask***</h1>
    </div>
        <h2 style="text-align:center;">The Predicted Group is:</h2>

    </div>
</header>
  <style>
        .blink {
        animation: blinker 0.6s linear infinite;
        color: #1c87c9;
        font-size: 30px;
        font-weight: bold;
        font-family: sans-serif;
    text-align:center;
        }
    @keyframes blinker {
            50% {
            opacity: 0;
            }
        }
</style>
<p class="blink">{{prediction}}</p>
```

Image 12.3.2: result.html snippet

12.4. Demonstration of Web application

After all these activities are completed, we need to open anaconda prompt and we need to navigate to the path of our project folder. Then we need to give command as **python app.py** and execute the command at end one URL will be generated we need to hit the URL in EDGE or chrome to check the application. The page in white back ground is home.html which is taking input and the page in pink background is result.html which is displaying the predicted result.

```
2020-09-19 17:58:45.857624: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1257] Device interconnect StreamExecut
with strength 1 edge matrix:
2020-09-19 17:58:45.858102: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1263]
 * Debugger is active!
 * Debugger PIN: 263-906-864
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Image 12.4.1: URL for the webpage

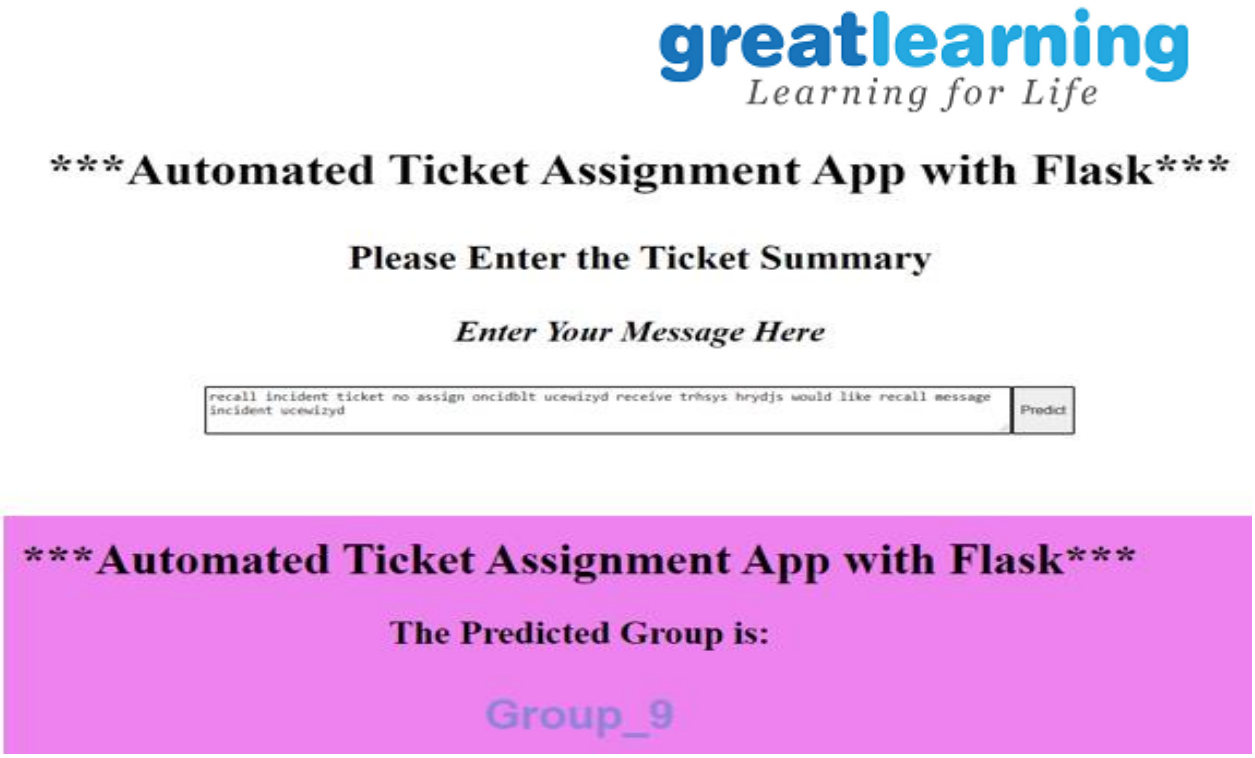


Image 12.4.2: Prediction of Group 9

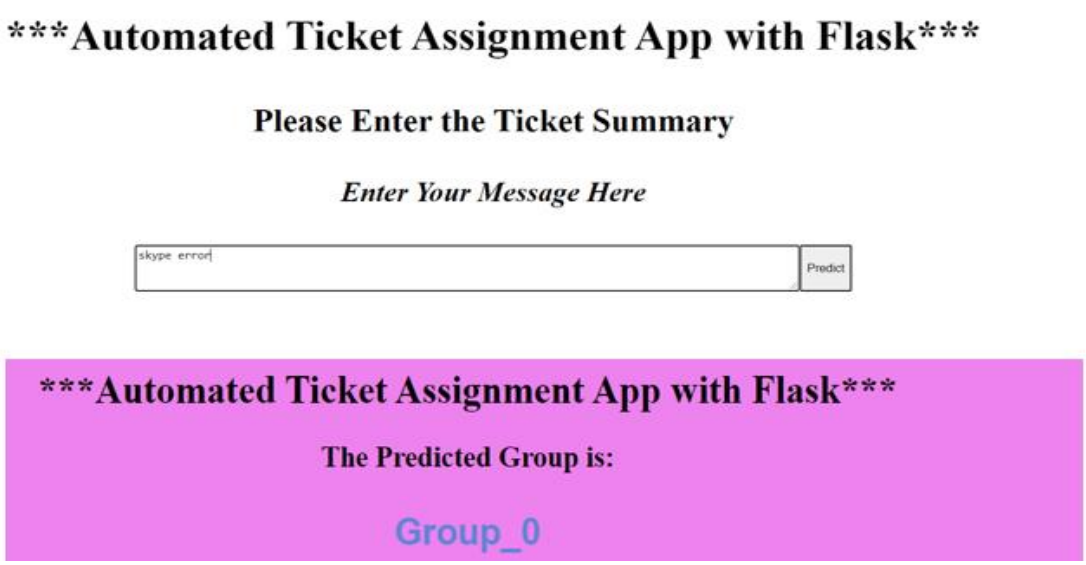


Image 12.4.3: Prediction of Group 0

Image 12.4.3: Prediction of Group 24

13.Implications

Our solution helps one to automate the ticket assignment process. Rather than having ETA of longer period of time while doing manual ticket classification and assignment, by using this, the tickets can be picked up faster. This reduces the number of manual labor and the possibility of error is also decreased, making the work much faster and increasing its efficiency. In current situation we have faced that our data was highly imbalanced and if we take a reference out of 8500 rows , then almost 5000+ rows were from top 5 group so the prediction for those groups were really good when we extracted our data for top 5 but for complete data set there were many groups who only had 1 ticket and causing a high imbalance. So, in real time scenario we need to add more data and make our group more balanced then our solution will be a perfect example of ticket assignment operation.

Moreover, our deployed webpage can be designed for the vendor who receives the tickets. Within a click after the model deployment is done, the ticket will be showing in the queue of the perfect group or person to whom it should be assigned according to description raised in the ticket. So, in real time our deployed solution will work with high confidence.

This solution can be implemented in all the Support Centre/Service Centre/Help desk where there is online ticketing to resolve any issue in banking, IT, Telecom, Retail, Healthcare, Travel industry.

14.Limitations

1.After the text preprocessing it is seen that there are some ids which were relevant and frequent like SID_43. The regular expression could not handle the ids and therefore the numbers and "_" got removed.

2. Some more text preprocessing can be done here like to handle most frequent rare words etc. That was not targeted due to time constraints.

3.As discussed earlier also the data imbalance was a main reason for the poor performance of the model for complete data set. So, if the scenarios where data is highly imbalanced are to be considered, then we have to revisit data source.

4. For pretrained models, the training time was high for few models which increases the time complexity.

5.As BERT is a heavy pretrained model with a large number of parameters, most of the times the notebook crashed or became unresponsive due to loading failure. So, in real time whenever we need to use BERT, high configuration GPU should be used to avoid this issue.

6.In ULM fit we have used Text list and got a very good score but in Realtime scenario we need to check text data bunch and other options also for verifying changes maximum scores.
7. To standardize the deployment process we need to use more advanced deployment option like deployment in Heroku, Azure, AWS Sage maker, Google cloud etc.

15.Challenges Faced

1.In our data set there were lots of non-English ticket descriptions with which it was difficult to proceed for text preprocessing. We have used polyglot library to detect non-English rows and then translated those non-English rows with the help of Google translator.
2.There were some NA values in the data set in description and short description column. We have used "THE" at the place of NA to avoid null value occurrence. As "THE" is a stop word so it will be removed automatically at the time of stop word removal.
3. We have split our data frame to non-English and English then translated the non-English rows and merged both of them to get the final data set.
4.Initially all the negative stop words were removed as a part of removing stop word process but for this problem statement negative words play a significant role so the negative stop words cannot be removed. We have made a customized stop word list which includes all the stop words, except for the negative ones.
5.Initially we were using Line plots to plot the accuracy, precision, recall and F1 score plot but the plots were not visible clearly due to which we switched to bar plot for getting a better visualization.
6. After the text preprocessing it is seen that there are some ids which were relevant and frequent like SID_43. The regular expression could not handle the ids and therefore the numbers and "_" got removed.
7.Initially with all deep learning models the training time was very high due to use of Non-GPU configuration. This problem was resolved by switching to notebook GPU configuration.
8.For BERT with tensor flow models we have faced page loading error due to a large number of parameters. So, we used pytorch representations of BERT to unravel the issue.
9.For BERT with the pytorch library sometimes we faced the issue of the model becoming unresponsive for a large number of epochs and higher batch size. We resolved this problem by running the model for 2 epochs each and a minimum batch size of 3.
10.For ULM fit initially we were using text data bunch with the data frame and got very poor accuracy and high losses. When we replaced text data bunch with text list and imported the dataset as csv then issue got resolved and the accuracy was improved considerably.
11.For Fast text we are getting predicted output in __label__(X) format. To calculate accuracy, we need predicted and true test values. To resolve the issue first we created a data frame with predicted and actual value and then applied regular expression on predicted value to remove special characters and "label" text. After which we have label encoded the actual and predicted columns and also calculated accuracy.
12. For deployment initially we did not take the reference of the top 5 dataset dictionary therefore for each new ticket description the tokenization was done afresh and the prediction was coming out to be wrong. To resolve this problem, we have loaded the top 5 data set and took the reference of its dictionary to get the proper tokenized sequence. After implementing this, the predicted groups are coming out to be correct.
13.For complete data set we have seen that the model performance was not coming well so we have selected top 5 group with highest ticket count and checked all the models' performance on top 5 data sets. After which we observed that the accuracy and F1 score had improved really well.

16.Conclusion

From the complete process from starting, till the end we have now learnt that every industry deal with a huge amount of data coming in on a daily basis. In this era of automation, it is always preferred that the manual labor is decreased and we get good accuracy/F1 score as much possible.

We have learned how to train different pre trained models in an efficient way, how to deploy the model, how to handle data inconsistency and how to apply modifications on different deep learning models.

In this project we have focused more on the models but now after trying upon so many different models and going through the comparisons we have come up to the conclusion that along with focusing on the models we should have also focused on the text preprocessing part with more customized regular expressions.

After printing the classification report, the F1 score should be checked for all the groups and the groups for which the score came out as poor, there we need to plot word cloud, check the word frequencies and patterns of the words for applying more text preprocessing so that the group can be assigned correctly.

According to the supervised dataset which was provided to us we have listed out the top models which gave us good results. It has also taught us what steps and approach should be followed when we come across datasets like this.

In future we will use this entire project experience to deal with any multiclass/binary text classification problem efficiently.

----------THE END----------