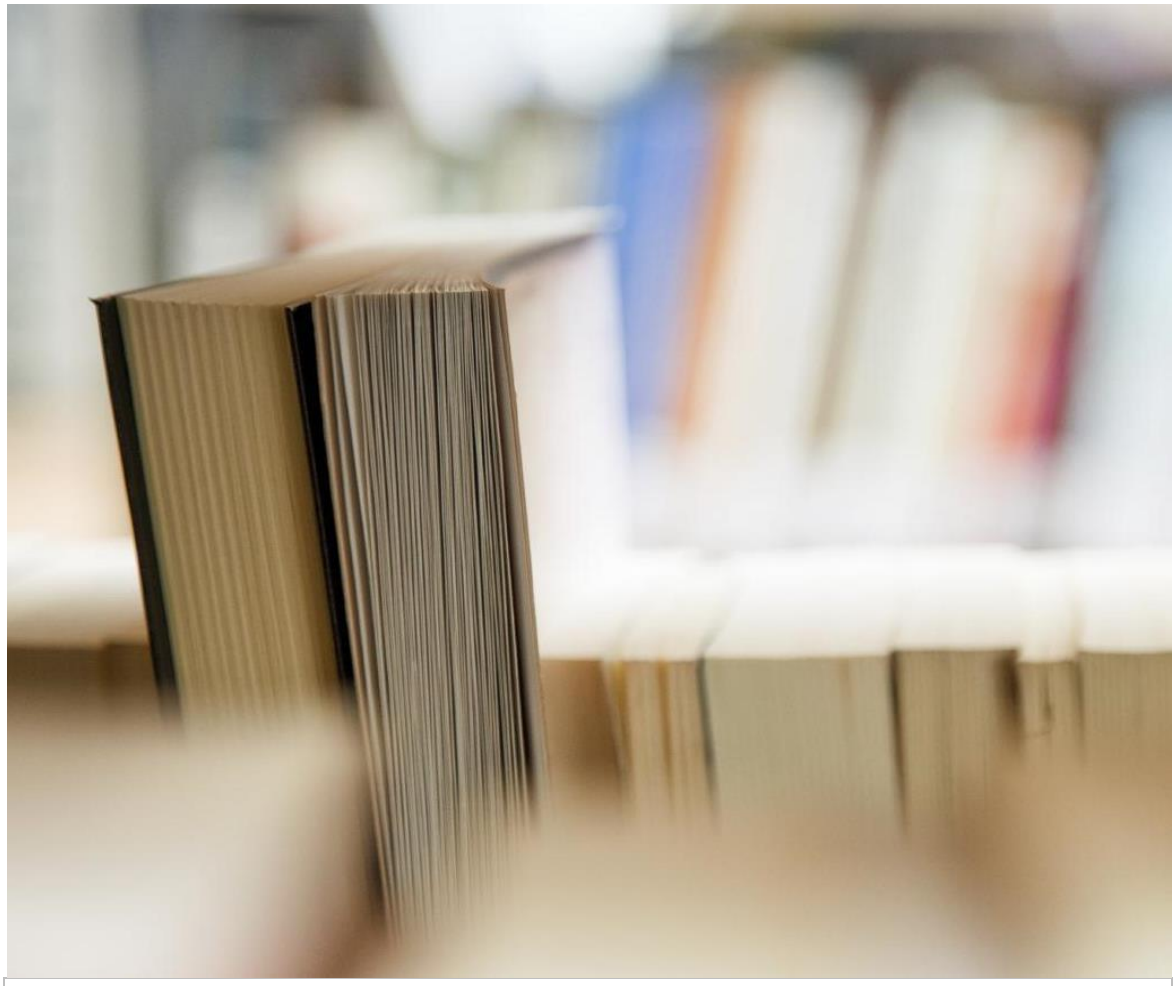


AUTOMATIC IT TICKET ASSIGNMENT

NATURAL LANGUAGE PROCESSING

Pulkit Malhotra
M V Satya Mohan
Chaitanya Sagar

09-JAN-2021

FINAL REPORT

CONTENTS

SUMMARY OF PROBLEM STATEMENT, DATA AND FINDINGS

One of the key activities of any IT function is to “Keep the lights on” to ensure there is no impact to the Business operations. IT leverages Incident Management process to achieve the above Objective. An incident is something that is unplanned interruption to an IT service or reduction in the quality of an IT service that affects the Users and the Business. The main goal of Incident Management process is to provide a quick fix / workarounds or solutions that resolves the interruption and restores the service to its full capacity to ensure no business impact. In most of the organizations, incidents are created by various Business and IT Users, End Users/ Vendors if they have access to ticketing systems, and from the integrated monitoring systems and tools. Assigning the incidents to the appropriate person or unit in the support team has critical importance to provide improved user satisfaction while ensuring better allocation of support resources. The assignment of incidents to appropriate IT groups is still a manual process in many of the IT organizations. Manual assignment of incidents is time consuming and requires human efforts. There may be mistakes due to human errors and resource consumption is carried out ineffectively because of the misaddressing. On the other hand, manual assignment increases the response and resolution times which result in user satisfaction deterioration / poor customer service.

Guided by powerful AI techniques that can classify incidents to right functional groups can help organizations to reduce the resolving time of the issue and can focus on more productive tasks.

So, the problem at hand was to train an ML model which can predict and classify the correct assignment group for an IT Ticket based on the ticket description, caller, etc.

1. DATA SAMPLE

Dataset location:

https://drive.google.com/file/d/1OZNJm81JXucV3HmZroMq6qCT2m7ez7IJ/view

```
data.head()
```

	Short description	Description	Caller	Assignment group
0	login issue	-verified user details.(employee# & manager name)\r\n-checked the user name in ad and reset the password.\r\n-advised the user to login and check.\r\n-caller confirmed that he was able to login.\r\n-issue resolved.	spxjnwir pjlcoqds	GRP_0
1	outlook	\r\n\r\nreceived from: hmjdrvpb.komuaywn@gmail.com\r\n\r\nhello team,\r\n\r\nmy meetings/skype meetings etc are not appearing in my outlook calendar, can somebody please advise how to correct this?\r\n\r\nkind	hmjdrvpb komuaywn	GRP_0
2	cant log in to vpn	\r\n\r\nreceived from: eylqgodm.ybqkwiam@gmail.com\r\n\r\nhi\r\n\r\ni cannot log on to vpn\r\n\r\nbest	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

2. DATA DESCRIPTION

a. **Short description**: High level details about the incident.
b. **Description**: Detailed information about the incident.
c. **Caller**: Person who has lodged this incident.
d. **Assignment group**: Group/Team which is intended to resolve the incident.

3. DATASET ANALYSIS

a. **Volume**: Total of 8500 incident information available.

```
# Understand data structure
data.info()
```

```
<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
```

b. **Null values:** Short description has 8 rows having null value.

```
# Check missing values in short description
data[data['Short description'].isna()]
```

	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

c. **Null values:** Description has 1 row having null value.

```
# Check missing values in Description
data[data['Description'].isna()]
```

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

d. No. of Unique values in each column

```
# No. of unique values in all columns

for col in data.columns:
    print(f"{col}: {data[col].nunique()}")
```

```
Short description: 7482
Description: 7818
Caller: 2950
Assignment group: 74
```

e. **Duplicates:** There are 591 duplicate rows

```
copy_df = data.copy()
duplicates = copy_df[copy_df.duplicated()]
print('Number of duplicates are {}'.format(len(duplicates)))
duplicates
```

```
Number of duplicates are 591
```

	Short description	Description	Assignment group
51	call for ecwtrjnq jpecxuty	call for ecwtrjnq jpecxuty	GRP_0
81	erp SID_34 account locked	erp SID_34 account locked	GRP_0
123	unable to display expense report	unable to display expense report	GRP_0
157	ess password reset	ess password reset	GRP_0
229	call for ecwtrjnq jpecxuty	call for ecwtrjnq jpecxuty	GRP_0
...	...	...	...
8424	windows account lockout	windows account lockout	GRP_0
8450	unable to connect to wifi	unable to connect to wifi	GRP_0
8451	password reset erp SID_34	password reset erp SID_34	GRP_0

f. Distribution of tickets among groups

```
dist["count_perc"] = round((dist["Num tickets"]/data.shape[0])*100,2)
dist.sort_values(["count_perc"], axis=0,
                 ascending=False, inplace=True)
dist.head()
```

	Assignment group	Num tickets	count_perc
0	GRP_0	3429	43.36
72	GRP_8	645	8.16
17	GRP_24	285	3.60
4	GRP_12	256	3.24
73	GRP_9	252	3.19

As per above its clear that majority of the data belongs to GRP_0 and rest other groups have less than 10% of the overall tickets.

g. Observed that majority of the incidents are lodged in English language. However few of the incidents are lodged in languages other than English. We have used language detector and translator to convert them to English.

h. Caller name is present in the description column for around 10% of the incidents.

OVERVIEW OF THE FINAL PROCESS

OBSERVATIONS FROM EDA AND PRE-PROCESSING:

a. The target class distribution is skewed. Close to 50% data belongs to GRP_0 which might make modelling building a challenge.
b. There are some groups with really low number of entries and might make modelling training for these groups harder as training samples would be very low in number, so we decide to classify these as a miscellaneous group (GRP_MISC). This may also reduce the imbalance to some extent

DATA CLEANING STEPS:

a. Dropped the caller field as it should not make any contribution to model building.
b. Replaced the null values in short description and description columns with empty strings.
c. Concatenated short description and description to create a final description columns where description was not equal to short description whereas final description was mapped to description where it was equal to short description.
d. Contraction words were removed from the final description column.
e. Converted the final description to lower case.
f. Removed hashtags, hyperlinks, URLs, punctuations, HTML tags and non-ASCII characters from final description.
g. Language translation of all descriptions to English.
h. Stop word removal
i. Lemmatization.

MODELLING ALGORITHMS USED:

a. Multinomial naïve bayes with all data groups on unsampled data.
b. Multinomial naïve bayes with all data groups on resampled data.
c. Multinomial naïve bayes with top 10 groups unsampled data.
d. Multinomial naïve bayes with top 6 groups unsampled data.
e. Word2Vec word embeddings with Bi-LSTM.
f. Glove word embeddings with Bi-LSTM.
g. Fast-Text word embeddings with Bi-LSTM.
h. Elmo word embeddings with Bi-LSTM.
i. BERT with unsampled data.
j. BERT with resampled data.

STEP-BY-STEP WALK THROUGH OF THE SOLUTION

EDA AND DATA PRE-PROCESSING

Starting with Exploratory data analysis of the data, the approach followed was to highlight any inconsistencies in the data, clean the data and prepare the data ready to be used for the model.

Below steps were followed as a part of EDA and Pre-processing.

1. TAKING CARE OF NULLS IN THE DATA

While doing analysis on the data, we found that there are some nulls in the Short Description and Description columns.

```
[5]: # Check missing values in short description
     data[data['Short description'].isna()]
```

[5]:

	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

```
[6]: # Check missing values in Description
     data[data['Description'].isna()]
```

[6]:

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

But we see, for rows where Short Description is null, Description is populated and for row where Description is null, Short Description is populated.

Hence we decided not to drop these nulls and later use them to concatenate these two to create a final description column. So we just replace the nulls with empty string ‘’.

2. DROPPING OFF THE CALLER COLUMN

After a bit of more analysis into the data, we discovered that the column Caller representing the caller name might not serve any purpose in helping predict the correct assignment group. Hence, we decided the drop off the Caller column.

3. DROPPING OFF THE DUPLICATES IN DATA

As already highlighted in the introduction, there were 591 duplicates in the data. So we dropped them off.

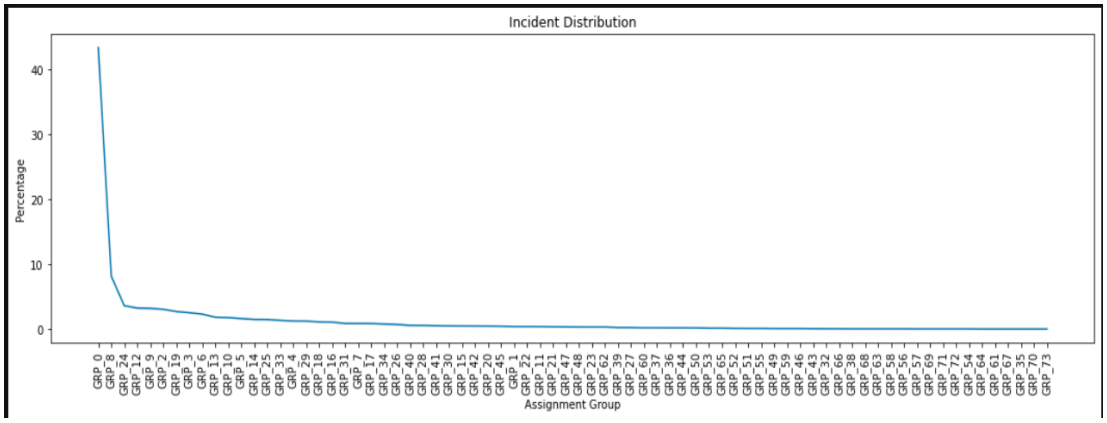
4. ANALYSING DISTRIBUTION OF TICKETS AMONG ASSIGNMENT GROUPS

We checked the distribution of data among different assignment groups to check whether the data has some skew. The image shows distribution of top 5 groups.

```
[14]: dist["count_perc"] = round((dist["Num tickets"]/data.shape[0])*100,2)
      dist.sort_values(["count_perc"], axis=0,
                       ascending=False, inplace=True)
      dist.head()
```

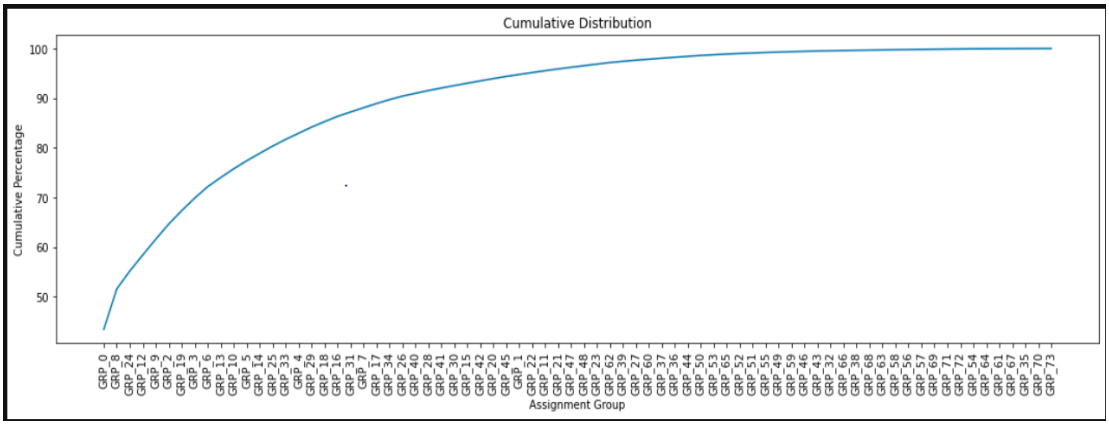
[14]:

	Assignment group	Num tickets	count_perc
0	GRP_0	3429	43.36
72	GRP_8	645	8.16
17	GRP_24	285	3.60
4	GRP_12	256	3.24
73	GRP_9	252	3.19



As we clearly see that about 43.36% of data belongs to GRP_0. So the data skew is present in the data.

The second observation from this analysis was there were a number of groups with really less numbers of rows which might become a hurdle while training the model.



As we see, there is an elbow in the cumulative distribution graph, which makes us think that many groups have really less number of rows. And we see an elbow at GRP_30. Now groups till GRP_30 have about 92% of total data. And the rest groups had only 8% of data, we decided to classify all these groups into a group GRP_MISC.

```
[20]: # We can see a clear elbow at grp_30 and we will classify all further groups into a misc group because they a

      cum_dist[cum_dist['Assignment group']=='GRP_30']
```

[20]:

	Assignment group	Num tickets	count_perc	Cumulative Num Tickets	Cumulative Perc
24	GRP_30	39	0.49	7319	92.55

```
[21]: # Identify all the assignment groups which have no more than 39 tickets
      dist[dist['Num tickets'] <= 39]
      print('Groups having number of tickets less than 39 are : {}'.format(len(dist[dist['Num tickets'] <= 39])))

      Groups having number of tickets less than 39 are : 47
```

There were 47 such groups which got classified into GRP_MISC. Now, the new group count after this activity was 28.

5. CONCATENATING SHORT DESCRIPTION AND DESCRIPTION TO CREATE FINAL DESCRIPTION

As discussed in point 1 as well, we decided to concatenate these two columns to create a final description column. But before we did that, we checked whether there are some rows where Short Description was exactly equal to Description. And we found out there were a few.

```
[25]: # Check whether description is equal to Short Description
      data[data['Short description'] == data['Description']]
```

[25]:

	Short description	Description	Assignment group
3	unable to access hr_tool page	unable to access hr_tool page	GRP_0
4	skype error	skype error	GRP_0
5	unable to log in to engineering tool and skype	unable to log in to engineering tool and skype	GRP_0
7	ticket_no1550391- employment status - new non-...	ticket_no1550391- employment status - new non-...	GRP_0
8	unable to disable add ins on outlook	unable to disable add ins on outlook	GRP_0
...	...	...	...
8486	ticket update on ticket_no0427635	ticket update on ticket_no0427635	GRP_0
8492	hr_tool etime option not visitble	hr_tool etime option not visitble	GRP_0
8494	tablet needs reimaged due to multiple issues w...	tablet needs reimaged due to multiple issues w...	GRP_3
8496	telephony_software issue	telephony_software issue	GRP_0
8497	vip2: windows password reset for tifpdchb pedx...	vip2: windows password reset for tifpdchb pedx...	GRP_0

2348 rows × 3 columns

So there are 2348 rows where Short Description was equal to Description. Now while concatenating, we kept this in mind, and took only Description for these rows where Short Description was equal to Description and concatenated Description and Short Description were not equal.

6. LANGUAGE DETECTION AND TRANSLATION

While performing EDA we came across some characters which did not belong to English language, so we check what all languages were used to describe the Tickets. We used pycl2 library for doing this.

So the Distribution of data among different languages seemed like this

[31]:

	language	Num tickets	count_perc	Cumulative Num Tickets	Cumulative Perc
2	ENGLISH	7146	90.35	7146	90.35
11	Unknown	396	5.01	7542	95.36
4	GERMAN	341	4.31	7883	99.67
8	PORTUGUESE	10	0.13	7893	99.80
6	LATIN	5	0.06	7898	99.86
0	DANISH	2	0.03	7900	99.89
7	POLISH	2	0.03	7902	99.92
9	SPANISH	2	0.03	7904	99.95
1	DUTCH	1	0.01	7905	99.96
3	GALICIAN	1	0.01	7906	99.97
5	INTERLINGUA	1	0.01	7907	99.98
10	TURKISH	1	0.01	7908	99.99
12	WARAY_PHILIPPINES	1	0.01	7909	100.00

As we can see, Most of the data is captured in either ENGLISH, GERMAN or Unknown language.
Let's see what does ‘Unknown’ group looks like

[32]:
```
data[data.language=='Unknown'].head(10)
```

[32]:

	final_description	Assignment group	language
39	call for ecwtrjnq jpecxuty	GRP_0	Unknown
126	blank call //gso	GRP_0	Unknown
146	erp_print_tool install.	GRP_0	Unknown
148	install acrobat standard	GRP_0	Unknown
211	insurance information	GRP_0	Unknown
222	support fÃ¼r fa.gstry \arexjftu ohxdwngl suppo...	GRP_24	Unknown
230	blank call	GRP_0	Unknown
255	probleme mit laufwerk z: \laeusvjo fvaihgpx	GRP_24	Unknown

It seems that some of them are English, some are German but they somehow were not detected properly by pycld2. (Upon deeper inspection of all the rows, we see there are few non-ascii descriptions which are all gibberish)

It would be good if we can try to translate the German + Unknown group to English which enables us to keep > 99.5% of the data.

Hence we used Google translate python library to translate our descriptions to English.

255	probleme mit laufwerk z: \laeusvjo fvaihgpx	GRP_24	Unknown	problems with drive z: \ laeusvjo fvaihgpx
270	neues passwort fÃ¼r accountname tgryhu hgygrtu...	GRP_0	GERMAN	new password for account name tgryhu hgygrtui ...

Left shows the descriptions before translation and right shows the translated descriptions.
This seems to have worked well and it translated words with accented chars as well. Now we only keep rows with language as English, German or Unknown (as they have been converted)
Note: The gibberish descriptions' translation will still be gibberish, but they will be removed in data-pre-processing stage.

7. DATA CLEANING

As a part of this step we achieved the following things:

a. Stop word removal, removed punctuations, removed image tags, removed HTML tags, removed common email words like hi, hello, received, outlook, gmail, cc, bcc, etc., removed ascii characters, lower casing all the text, removed contacts in descriptions (like abc@outlook.com). All this was done by a function clean_text() in our code.

```
def clean_text(text):

    # Remove HTML tags and image content placeholders
    text = text.lower()
    text = re.sub(html_image_cnt_tags, ' ', text)

    # Remove punctuation + digits + line-breaks
    text = re.sub(puncs, ' ', text) # Replace all punctuations with ' ', so words can still be tokenized properly

    # Remove stop words and common mail words
    words = word_tokenize(text)
    words = [w for w in words if w not in stop_words]
    sentence = ' '.join(words)

    return sentence
```

b. Lemmatization: Lemmatisation (or **lemmatization**) in linguistics is the process of grouping together the inflected forms of a word so they can be analysed as a single item, identified by the word's lemma, or dictionary form. Example: rocks □ rock, corpora □ corpus, better □ good. We chose lemmatization over stemming because it does morphological analysis of the words. We achieved this using the NLTK library with using the UDF lemmatize_words().

```
def lemmatize_words(sentence):
    #tokenize the sentence and find the POS tag for each token
    nltk_tagged = nltk.pos_tag(nltk.word_tokenize(sentence))
    #tuple of (token, wordnet_tag)
    wordnet_tagged = map(lambda x: (x[0], nltk_tag_to_wordnet_tag(x[1])), nltk_tagged)
    lemmatized_sentence = []
    for word, tag in wordnet_tagged:
        if tag is None:
            #if there is no available tag, append the token as is
            lemmatized_sentence.append(word)
        else:
            #else use the tag to lemmatize the token
            lemmatized_sentence.append(lemmatizer.lemmatize(word, tag))
    return " ".join(lemmatized_sentence)
```

8. DATA CLEANING AFTER LEMMATIZATION AND TRANSLATION

Now, after lemmatization and translation of text, we might see some nulls (sentences that could not be converted to English) and some duplicates. Hence we remove them.

```
# Check missing values in proc_final_description
data[data['proc_final_description'].isna()].head()
```

	Assignment group	proc_final_description
1036	GRP_MISC	NaN
1125	GRP_MISC	NaN
1383	GRP_MISC	NaN
1617	GRP_MISC	NaN
1618	GRP_MISC	NaN

```
# check for dupliactes after translation and lemmatization
data[data.duplicated()]
```

	Assignment group	proc_final_description
14	GRP_0	ticket update inplant
39	GRP_0	ticket update inplant
57	GRP_8	job mm zscr dly merktc fail job scheduler rece...
66	GRP_8	job job fail job scheduler receive job job fai...
67	GRP_8	job job fail job scheduler receive job job fai...
...	...	...
7835	GRP_8	abended job job scheduler job receive abended ...
7844	GRP_9	abended job job scheduler job receive abended ...
7846	GRP_9	abended job job scheduler job receive abended ...
7850	GRP_8	abended job job scheduler bkwin hostname inc r...
7870	GRP_0	ticket update ticket

1110 rows × 2 columns

So we have **43** rows with NaNs and a further **1110** duplicate rows. After dropping them, we are left with **6730** rows.

MODELLING

Approach to building a modelling solution is to first verify if the data is still consistent after pre-processing. Then we prepare the target column and split the data into train and test. Then we create a baseline model based on Bag of Words and ML techniques and then try a more complex LSTM based model.

1. DATA PREPARATION FOR MODELLING

We label encode the target column and keep only relevant columns

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
# Label encode Assignment group
data['Assignment group'] = le.fit_transform(data['Assignment group'])

# Keep only relevant columns
data = data[['proc_final_description', 'Assignment group']]
data.head()
```

	proc_final_description	Assignment group
0	login issue verify user detail employee manage...	0
1	receive team meeting skype meeting etc appear ...	0
2	cant log vpn receive log vpn best	0
3	unable access hr tool page	0
4	unable log engineering tool skype	0

We then split the data into train and test sets in a stratified way which will preserve the class imbalances.

```
from sklearn.model_selection import train_test_split

X = data['proc_final_description']
y = data['Assignment group']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, stratify=y, random_state=12)
```

2. BASELINE ML BASED MODEL

For the baseline model, we create Bag-of-words considering only top 10k frequent words and then use these in Tf-Idf vectorizer to convert string into vector representations.

```
from sklearn import feature_extraction, model_selection, naive_bayes, preprocessing, feature_selection, metrics

# initialize vectorizer with max vocab of 10000 words
vectorizer = feature_extraction.text.TfidfVectorizer(max_features=10000, ngram_range=(1,2))
```

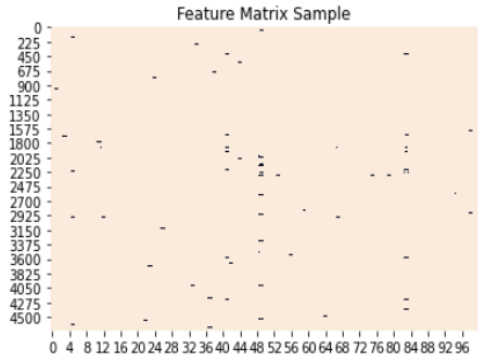
```
# Fit and transform the vectorizer on train set
X_train_tfidf_trans = vectorizer.fit_transform(X_train)

# Have a look at vectorizer vocabulary
vectorizer.vocabulary_['skype meeting']
```

```
8060
```

```
# Visualize the feature matrix
sns.heatmap(X_train_tfidf_trans.todense()
            [:,np.random.randint(0,X_train_tfidf_trans.shape[1],100)]==0, vmin=0, vmax=1, cbar=False) \
    .set_title('Feature Matrix Sample')
```

```
Text(0.5, 1.0, 'Feature Matrix Sample')
```



Then we use Multinomial Naive-Bayes model for classification

```
# Let's build a train a classifier on above data
# We will start with Multinomial Naive-Bayes
from sklearn.naive_bayes import MultinomialNB

# Define and train the model
classifier = MultinomialNB()
classifier.fit(X_train_tfidf_trans, y_train)

# transform the test set
X_test_tfidf_trans = vectorizer.transform(X_test)

# Make predictions
predicted = classifier.predict(X_test_tfidf_trans)
predicted_prob = classifier.predict_proba(X_test_tfidf_trans)
```

And get the following results:

```
Accuracy: 0.54
Auc: 0.8
Detail:
              precision    recall  f1-score   support

           0       0.51      1.00      0.68       931
           1       0.00      0.00      0.00        25
           2       0.79      0.20      0.32        74
           3       0.00      0.00      0.00        43
           4       1.00      0.06      0.11        35
           5       0.00      0.00      0.00        25
           6       0.00      0.00      0.00        20
           7       0.00      0.00      0.00        26
           8       0.00      0.00      0.00        64
           9       0.67      0.22      0.33        72
          10       1.00      0.35      0.51        81
          11       0.00      0.00      0.00        35
          12       0.00      0.00      0.00        16
          13       0.00      0.00      0.00        13
          14       0.00      0.00      0.00        29
          15       0.00      0.00      0.00        60
          16       0.00      0.00      0.00        17
          17       0.00      0.00      0.00        32
          18       0.00      0.00      0.00        19
          19       0.00      0.00      0.00        29
          20       0.00      0.00      0.00        13
          21       0.00      0.00      0.00        12
          22       0.00      0.00      0.00        16
          23       0.00      0.00      0.00        24
          24       0.00      0.00      0.00        20
          25       0.67      0.90      0.76        96
          26       0.00      0.00      0.00        24
          27       0.56      0.03      0.06       168

    accuracy                           0.54      2019
   macro avg       0.19      0.10      0.10      2019
weighted avg       0.43      0.54      0.40      2019
```

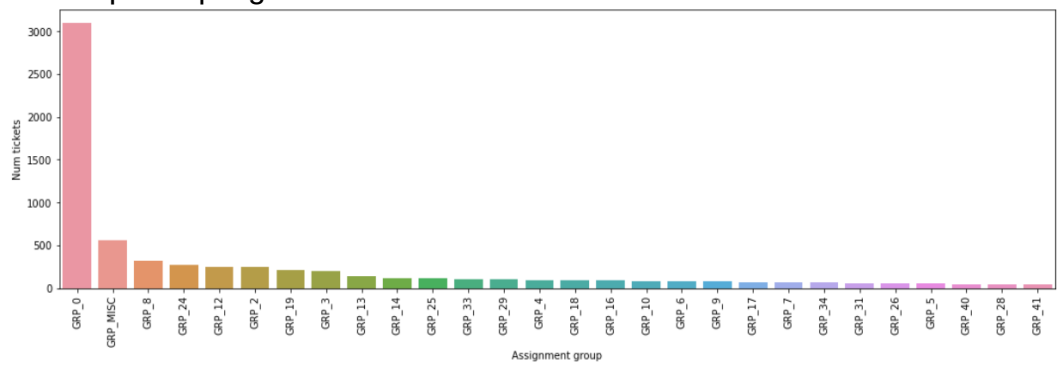
As we see, we have poor precision and recall for some classes. This is due to vast class imbalance, hence we will try to reduce this by up-sampling.

3. REDUCE CLASS IMBALANCE BY UP-SAMPLING

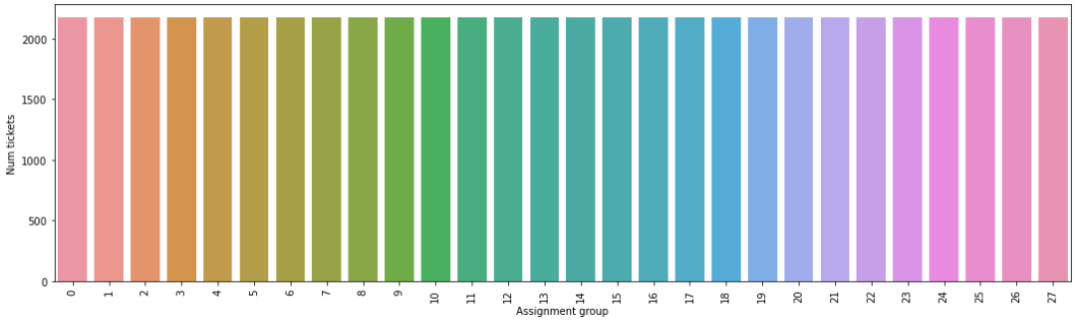
We use sklearn resample method to up-sample the data which reduces class imbalance.

```python
from sklearn.utils import resample
def resample_data(df, n_samples):
    df_resampled = df[0:0]
    for group in df['Assignment group'].unique():
        df_group = df[df['Assignment group'] == group]
        resampled = resample(df_group, replace=True, n_samples=n_samples, random_state=1)
        df_resampled = df_resampled.append(resampled)
    return df_resampled
```

Before up-sampling:



After up-sampling (only performed on train set):



We see that up-sampling reduces the class imbalance by duplicating the data for less frequent classes. Now all the classes have equal rows.

We trained the same model on up-sampled train data and we get below results:

```
Accuracy: 0.5
Auc: 0.92
Detail:
              precision    recall  f1-score   support

           0       0.84      0.47      0.60       931
           1       0.46      0.48      0.47        25
           2       0.64      0.59      0.62        74
           3       0.41      0.79      0.54        43
           4       0.26      0.43      0.32        35
           5       0.32      0.84      0.46        25
           6       0.27      1.00      0.43        20
           7       0.32      0.46      0.37        26
           8       0.29      0.50      0.36        64
           9       0.39      0.57      0.46        72
          10       0.85      0.74      0.79        81
          11       0.38      0.74      0.50        35
          12       0.16      0.29      0.20        17
          13       0.00      0.00      0.00        13
          14       0.47      0.59      0.52        29
          15       0.30      0.48      0.37        60
          16       0.22      0.41      0.29        17
          17       0.37      0.59      0.46        32
          18       0.26      0.63      0.36        19
          19       0.24      0.43      0.30        28
          20       0.17      0.31      0.22        13
          21       0.55      0.50      0.52        12
          22       0.24      0.75      0.37        16
          23       0.56      0.58      0.57        24
          24       0.29      0.65      0.40        20
          25       0.77      0.66      0.71        96
          26       0.39      0.54      0.46        24
          27       0.34      0.21      0.26       168

    accuracy                           0.50      2019
   macro avg       0.38      0.54      0.43      2019
weighted avg       0.62      0.50      0.52      2019
```

We see that precision and recall is now non-zero for most of the classes but still very low. But the accuracy is still very poor and just random at best.

Now we note that we have a lot of classes to predict, some of which actually form very less portion of our data. Hence, we will now be focussing on classifying data only belonging to top 6 classes.

4. TARGETING ONLY TOP 10 CLASSES

We tried to keep only top 10 classes (grp_misc) and see how our model performs:

```
# We see that we still have 28 groups, making the data extremely skewed. For the purpose of modelling,
# It is best if we keep a max of 10 classes to predict

# Get top 10 groups (excluding grp_misc)
top_10_grps = dist[dist['Assignment group'] != 'GRP_MISC']['Assignment group'].unique()[:10]

# Keep records only having assignment in top 10 groups
data_top10 = data[data['Assignment group'].isin(top_10_grps)]
data_top10['Assignment group'].value_counts()
```

```
GRP_0     3102
GRP_8      320
GRP_24     270
GRP_12     246
GRP_2      241
GRP_19     214
GRP_3      200
GRP_13     142
GRP_14     118
GRP_25     115
Name: Assignment group, dtype: int64
```

This reduces our data to **4969** rows.

We tried the same Multinomial Naïve Bayes Model on this data. Results are below:

```
Accuracy: 0.72
Auc: 0.85
Detail:
              precision    recall  f1-score   support

           0       0.70      1.00      0.82       931
           1       0.79      0.20      0.32        74
           2       0.00      0.00      0.00        43
           3       1.00      0.06      0.11        35
           4       0.00      0.00      0.00        64
           5       0.75      0.25      0.38        72
           6       1.00      0.31      0.47        81
           7       0.00      0.00      0.00        35
           8       0.00      0.00      0.00        60
           9       0.96      0.83      0.89        96

    accuracy                           0.72      1491
   macro avg       0.52      0.27      0.30      1491
weighted avg       0.65      0.72      0.63      1491
```

Since the accuracy is only 72% and f1 score of 4 groups here is 0, hence we decide to go ahead with top 6 groups only.

5. FOCUSSING ONLY ON TOP 6 CLASSES

We only keep top 6 classes (excluding grp_misc):

```
# We see that we still have 28 groups, making the data extremely skewed. For the purpose of modelling,
# It is best if we keep a max of 6 classes to predict

# Get top 6 groups (excluding grp_misc)
top_6_grps = dist[dist['Assignment group'] != 'GRP_MISC']['Assignment group'].unique()[:6]

# Keep records only having assignment in top 6 groups
data_top6 = data[data['Assignment group'].isin(top_6_grps)]
data_top6['Assignment group'].value_counts()
```

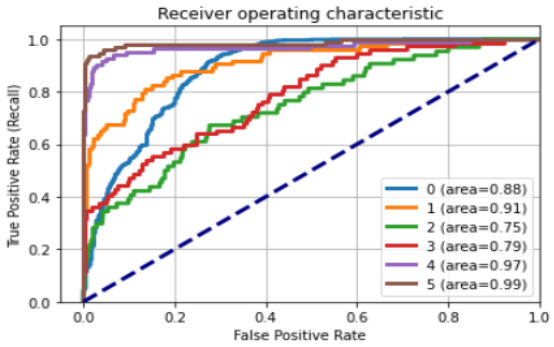
```
GRP_0     3102
GRP_8      320
GRP_24     271
GRP_12     246
GRP_2      241
GRP_19     214
Name: Assignment group, dtype: int64
```

This reduces our data to **4394** rows.
Now we try the same model on the new dataset and the results are below:

```
Accuracy: 0.81
Auc: 0.88
Detail:
              precision    recall  f1-score   support

           0       0.80      1.00      0.89       931
           1       0.88      0.20      0.33        74
           2       0.00      0.00      0.00        64
           3       0.86      0.25      0.39        72
           4       1.00      0.34      0.51        82
           5       0.96      0.85      0.91        96

    accuracy                           0.81      1319
   macro avg       0.75      0.44      0.50      1319
weighted avg       0.79      0.81      0.76      1319
```



We note that accuracy has considerably improved along with much better precision and recall values.

Now keeping this model as our baseline, we will build complex LSTM based models.

6. WORD2VEC WITH BI-DIRECTIONAL LSTM

We use the gensim library and train the Word2Vec model with the sentences in our train set to create the word embeddings.

```
import gensim

train_words = list()

for line in X_train:
    train_words.append(line.split())

#train word2vec model
model = gensim.models.Word2Vec(
    sentences=train_words,
    size=300,
    window=5,
    workers=4,
    min_count=1)

num_words = len(list(model.wv.vocab))
num_words
```

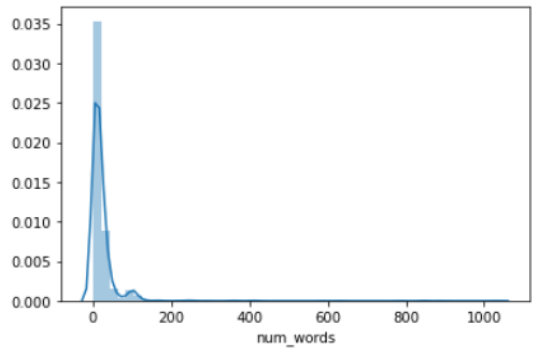
```
6317
```

We have 6317 words in our vocabulary. We save the embeddings file on disc which will later be used to create the embedding matrix.

Now we tokenize our data using the keras tokenizer and pad the sequences. But before that, we need to decide the ideal padding value for our sentences. For that we look at the distribution of sentence lengths

```
# Visualize the distribution
sns.distplot(temp_df.num_words)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x2acd8be53c8>
```



Now we take a look at average description length for each assignment group

```
# Lets look at the mean number of words per assignment group

data_avg_length = pd.DataFrame(temp_df.groupby('group').agg('mean')['num_words']) \
    .rename(columns={'num_words': 'avg_num_words'})
data_ticket_cnt = pd.DataFrame(temp_df.groupby('group').agg('count')['num_words']) \
    .rename(columns={'num_words': 'ticket_cnt'})
data_avg_length.join(data_ticket_cnt).sort_values(by='avg_num_words', ascending=False)
```

group	avg_num_words	ticket_cnt
3	69.165680	169
5	53.446429	224
1	29.046512	172
2	21.126667	150
0	15.556426	2171
4	7.328042	189

As we can see, the average is varying with the assignment group, it should be safe to keep truncate/pad the tickets at length of **100** words
Now we tokenize our data using the keras tokenizer and pad the sequences

```
# Create tokens for the description, we will use TF tokenizer
from tensorflow.keras.preprocessing.text import Tokenizer
tokenizer = Tokenizer()

# Fit on train texts and convert them to sequences
tokenizer.fit_on_texts(X_train) # X_train is np.array having training descriptions
X_train_sequences = tokenizer.texts_to_sequences(X_train)

# Convert the test texts to sequences
X_test_sequences = tokenizer.texts_to_sequences(X_test)
```

```
# Pad the sequences with max length
from tensorflow.keras.preprocessing.sequence import pad_sequences

# we take max_len as 100 after examining the average word count in the descriptions
max_len = 100

# Pad the sentences with 0 to mark them as unknown word
X_train_sequences_pad = pad_sequences(X_train_sequences, maxlen=max_len, padding='post', value=0)
X_test_sequences_pad = pad_sequences(X_test_sequences, maxlen=max_len, padding='post', value=0)

# Convert targets to numpy array
y_train = np.array(y_train)
y_test = np.array(y_test)
```

Now we create the embedding matrix using the trained word2vec embeddings and the trained tokenizer

```
# Create embedding matrix

embedding_size = 300

embeddings_index = {}
for o in open('w2v_emb_file_300.txt', encoding="utf8"):
    values = o.split()
    word = values[0]
    coefs = np.asarray(values[1:])
    embeddings_index[word] = coefs

# create a weight matrix for words in training docs
num_words = len(tokenizer.word_index)+1
embedding_matrix = np.zeros((num_words, embedding_size))

for word, i in tokenizer.word_index.items():
    if i > num_words:
        continue
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector
```

Now we define our LSTM model architecture, define loss and metrics and compile it

```
# Compile the model
model.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metrics=["accuracy"])
model.summary()
```

```
Model: "functional_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 100)]             0
_________________________________________________________________
embedding (Embedding)        (None, 100, 300)          1895400
_________________________________________________________________
bidirectional (Bidirectional (None, 1000)              3204000
_________________________________________________________________
flatten (Flatten)            (None, 1000)              0
_________________________________________________________________
dense (Dense)                (None, 200)               200200
_________________________________________________________________
dropout (Dropout)            (None, 200)               0
_________________________________________________________________
dense_1 (Dense)              (None, 100)               20100
_________________________________________________________________
dropout_1 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_2 (Dense)              (None, 50)                5050
_________________________________________________________________
dropout_2 (Dropout)          (None, 50)                0
_________________________________________________________________
dense_3 (Dense)              (None, 6)                 306
=================================================================
Total params: 5,325,056
Trainable params: 5,325,056
Non-trainable params: 0
_________________________________________________________________
```

Now we define call-backs and train our model

```
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau
checkpoint = ModelCheckpoint('model-{epoch:03d}-{val_accuracy:03f}.h5', verbose=1, monitor='val_accuracy',
                             save_best_only=True, mode='auto')
reduceLoss = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=2, min_lr=0.0001)
callbacks = [checkpoint,reduceLoss]
```

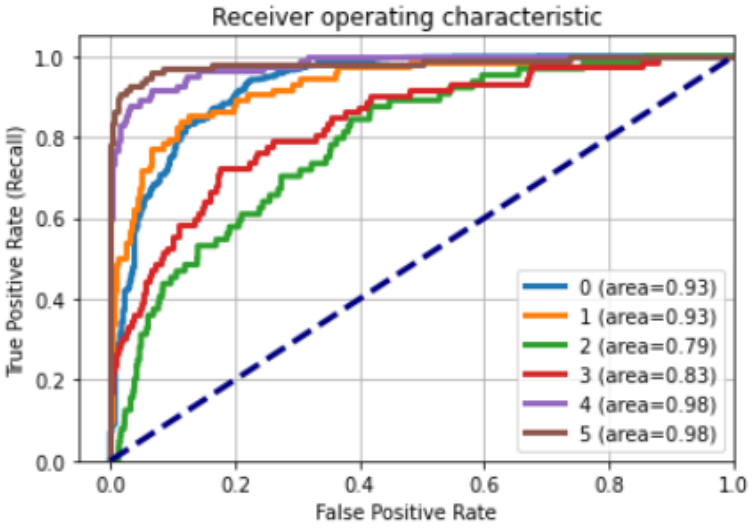
```
model.fit(X_train_sequences_pad, y_train, batch_size=128, epochs=10, callbacks=callbacks,
          validation_data=(X_test_sequences_pad, y_test), verbose=1)
```

We trained the model for 10 epochs. Below are the prediction results on test set

```
Accuracy: 0.84
Auc: 0.9
Detail:
              precision    recall  f1-score   support

           0       0.88      0.96      0.92       931
           1       0.73      0.51      0.60        74
           2       0.15      0.12      0.14        64
           3       0.48      0.32      0.38        72
           4       0.94      0.72      0.81        82
           5       0.93      0.86      0.90        96

    accuracy                           0.84      1319
   macro avg       0.69      0.58      0.63      1319
weighted avg       0.82      0.84      0.83      1319
```



We note that our model accuracy and AUC has improved and so has the precision and recall for all classes.

7. GLOVE WITH BI-DIRECTIONAL LSTM

We use the pre-trained glove embeddings from glove.6B.300d.txt which is a 300 dimensional embeddings with the sentences in our train set to create the embedding matrix.

```
# Create embedding matrix

embedding_size = 300

embeddings_index = {}
for o in open('glove.6B.300d.txt', encoding="utf8"):
    values = o.split()
    word = values[0]
    coefs = np.asarray(values[1:])
    embeddings_index[word] = coefs

# create a weight matrix for words in training docs
num_words = len(tokenizer.word_index)+1
embedding_matrix = np.zeros((num_words, embedding_size))

for word, i in tokenizer.word_index.items():
    if i > num_words:
        continue
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector
```

Now we define our LSTM model architecture, define loss and metrics and compile it as we did with Word2Vec.

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 100)]             0
_________________________________________________________________
embedding (Embedding)        (None, 100, 300)          1894800
_________________________________________________________________
bidirectional (Bidirectional (None, 1000)              3204000
_________________________________________________________________
flatten (Flatten)            (None, 1000)              0
_________________________________________________________________
dense (Dense)                (None, 200)               200200
_________________________________________________________________
dropout (Dropout)            (None, 200)               0
_________________________________________________________________
dense_1 (Dense)              (None, 100)               20100
_________________________________________________________________
dropout_1 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_2 (Dense)              (None, 50)                5050
_________________________________________________________________
dropout_2 (Dropout)          (None, 50)                0
_________________________________________________________________
dense_3 (Dense)              (None, 6)                 306
=================================================================
Total params: 5,324,456
Trainable params: 5,324,456
Non-trainable params: 0
_________________________________________________________________
```

Now we define call-backs and train our model

```
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau
checkpoint = ModelCheckpoint('model-{epoch:03d}-{val_accuracy:03f}.h5', verbose=1, monitor='val_accuracy',
                             save_best_only=True, mode='auto')
reduceLoss = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=2, min_lr=0.0001)
callbacks = [checkpoint,reduceLoss]
```

```
model.fit(X_train_sequences_pad, y_train, batch_size=128, epochs=20, callbacks=callbacks,
          validation_data=(X_test_sequences_pad, y_test), verbose=1)
```

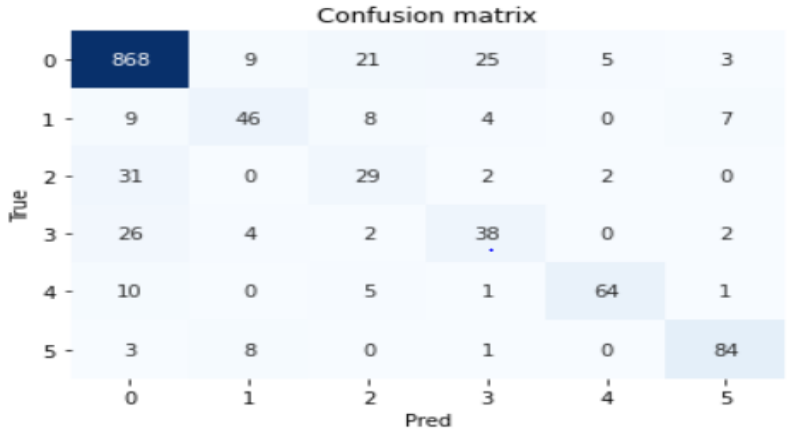
We trained the model for 20 epochs. Below are the prediction results on test set:

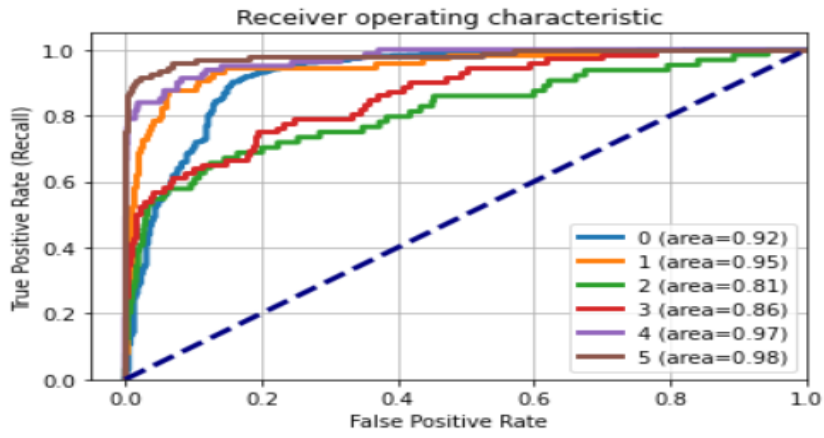
```
Accuracy: 0.86
Auc: 0.92
Detail:
              precision    recall  f1-score   support

           0       0.92      0.93      0.92       931
           1       0.69      0.62      0.65        74
           2       0.45      0.45      0.45        64
           3       0.54      0.53      0.53        72
           4       0.90      0.79      0.84        81
           5       0.87      0.88      0.87        96

    accuracy                           0.86      1318
   macro avg       0.73      0.70      0.71      1318
weighted avg       0.86      0.86      0.86      1318
```

We note that our model accuracy and AUC has improved and so has the precision and recall for all classes as compared to Word2Vec.





8. FASTTEXT WITH BI-DIRECTIONAL LSTM

We use the gensim library and train the FastText model with the sentences in our train set to create the word embeddings.

```
import gensim

train_words = list()

for line in X_train:
    train_words.append(line.split())

#train word2vec model
model = gensim.models.FastText(
    sentences=train_words,
    size=300,
    window=5,
    workers=4,
    min_count=1)

num_words = len(list(model.wv.vocab))
num_words
```

We have 6315 words in our vocabulary. We save the embeddings file on disc which will later be used to create the embedding matrix.

Now we create the embedding matrix using the trained fasttext embeddings and the trained tokenizer

```
# Create embedding matrix

embedding_size = 300

embeddings_index = {}
for o in open('fasttext_emb_file_300.txt', encoding="utf8"):
    values = o.split()
    word = values[0]
    coefs = np.asarray(values[1:])
    embeddings_index[word] = coefs

# create a weight matrix for words in training docs
num_words = len(tokenizer.word_index)+1
embedding_matrix = np.zeros((num_words, embedding_size))

for word, i in tokenizer.word_index.items():
    if i > num_words:
        continue
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector
```

Now we define our LSTM model architecture, define loss and metrics and compile it

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 100)]             0
_________________________________________________________________
embedding (Embedding)        (None, 100, 300)          1894800
_________________________________________________________________
bidirectional (Bidirectional (None, 1000)              3204000
_________________________________________________________________
flatten (Flatten)            (None, 1000)              0
_________________________________________________________________
dense (Dense)                (None, 200)               200200
_________________________________________________________________
dropout (Dropout)            (None, 200)               0
_________________________________________________________________
dense_1 (Dense)              (None, 100)               20100
_________________________________________________________________
dropout_1 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_2 (Dense)              (None, 50)                5050
_________________________________________________________________
dropout_2 (Dropout)          (None, 50)                0
_________________________________________________________________
dense_3 (Dense)              (None, 6)                 306
=================================================================
Total params: 5,324,456
Trainable params: 5,324,456
Non-trainable params: 0
```

Now we define call-backs and train our model

```
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau
checkpoint = ModelCheckpoint('model-{epoch:03d}-{val_accuracy:03f}.h5', verbose=1, monitor='val_accuracy',
                             save_best_only=True, mode='auto')
reduceLoss = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=2, min_lr=0.0001)
callbacks = [checkpoint,reduceLoss]
```

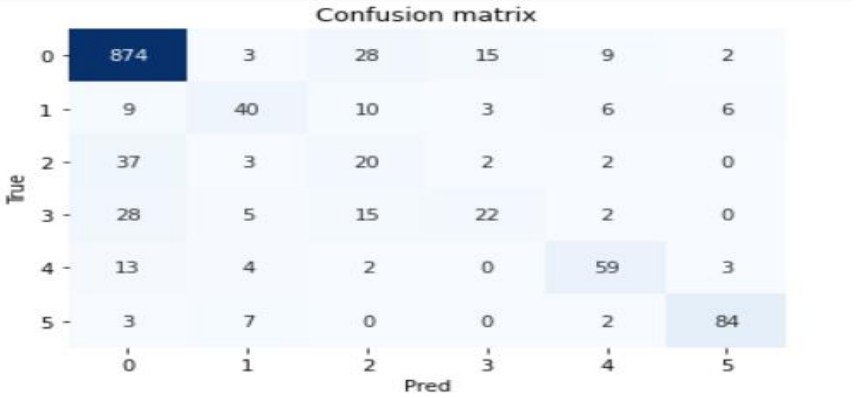
```
model.fit(X_train_sequences_pad, y_train, batch_size=128, epochs=20, callbacks=callbacks,
          validation_data=(X_test_sequences_pad, y_test), verbose=1)
```

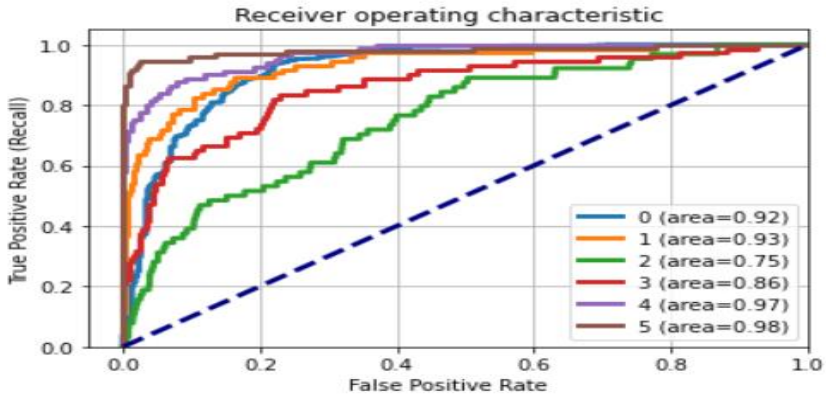
We trained the model for 20 epochs. Below are the prediction results on test set

```
Accuracy: 0.83
Auc: 0.9
Detail:
              precision    recall  f1-score   support

           0       0.91      0.94      0.92       931
           1       0.65      0.54      0.59        74
           2       0.27      0.31      0.29        64
           3       0.52      0.31      0.39        72
           4       0.74      0.73      0.73        81
           5       0.88      0.88      0.88        96

    accuracy                           0.83      1318
   macro avg       0.66      0.62      0.63      1318
weighted avg       0.83      0.83      0.83      1318
```





We note that our model accuracy and AUC has not improved much as compared to Glove and Word2Vec Embeddings.

9. ELMO WITH BI-DIRECTIONAL LSTM

We used the tensorflow hub library to create elmo emeddings. We transform our train and test dataset using this to be represented by elmo embeddings.

```
from tqdm import tqdm
```

```
elmo = hub.load("https://tfhub.dev/google/elmo/3")
```

```
WARNING:tensorflow:From C:\Users\Pulkit Malhotra\Anaconda3\lib\site-packages\tensorflow\python\ops\resource_variable_ops.py:1817: calling BaseResourceVariable.__init__ (from tensorflow.python.ops.resource_variable_ops) with constraint is deprecated and will be removed in a future version.
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
WARNING:tensorflow:From C:\Users\Pulkit Malhotra\Anaconda3\lib\site-packages\tensorflow\python\ops\resource_variable_ops.py:1817: calling BaseResourceVariable.__init__ (from tensorflow.python.ops.resource_variable_ops) with constraint is deprecated and will be removed in a future version.
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
```

```
def elmo_vectors(x):
    embeddings=elmo.signatures["default"](tf.constant(x))["elmo"]
    return embeddings
```

```
elmo_train = [elmo_vectors(x) for x in tqdm(list_train)]
```

```
100%|██████████████████████████████████████████████████████████████████████████| 62/62 [09:28<00:00,  9.17s/it]
```

```
elmo_train[0].shape
```

```
TensorShape([50, 100, 1024])
```

```
elmo_test = [elmo_vectors(x) for x in tqdm(list_test)]
```

```
  0%|                                                                                | 0/27 [00:00<?, ?it/s]
  4%|███                                                                            | 1/27 [00:09<03:56,  9.11s/it]
  7%|█████▉                                                                         | 2/27 [00:18<03:48,  9.14s/it]
 11%|████████▉                                                                      | 3/27 [00:27<03:42,  9.27s/it]
 15%|███████████▊                                                                   | 4/27 [00:38<03:41,  9.61s/it]
 19%|██████████████▊                                                                | 5/27 [00:49<03:44, 10.21s/it]
 22%|█████████████████▊                                                             | 6/27 [01:00<03:36, 10.29s/it]
 26%|████████████████████▋                                                          | 7/27 [01:10<03:23, 10.19s/it]
 30%|███████████████████████▋                                                       | 8/27 [01:19<03:10, 10.03s/it]
 33%|██████████████████████████▋                                                    | 9/27 [01:29<02:58,  9.94s/it]
 37%|█████████████████████████████▌                                                | 10/27 [01:39<02:48,  9.92s/it]
 41%|████████████████████████████████▌                                             | 11/27 [01:49<02:37,  9.86s/it]
```

We save the embedded train and test data on disc in form of a pickle file which will later be used for modelling.

```
pickle_out = open("elmo_train.pickle","wb")
pickle.dump(elmo_train_new, pickle_out)
pickle_out.close()

pickle_out = open("elmo_test.pickle","wb")
pickle.dump(elmo_train_new, pickle_out)
pickle_out.close()
```

Now we define our LSTM model architecture, define loss and metrics and compile it

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 100, 1024)]       0
_________________________________________________________________
bidirectional (Bidirectional (None, 1000)              6100000
_________________________________________________________________
flatten (Flatten)            (None, 1000)              0
_________________________________________________________________
dense (Dense)                (None, 200)               200200
_________________________________________________________________
dropout (Dropout)            (None, 200)               0
_________________________________________________________________
dense_1 (Dense)              (None, 100)               20100
_________________________________________________________________
dropout_1 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_2 (Dense)              (None, 50)                5050
_________________________________________________________________
dropout_2 (Dropout)          (None, 50)                0
_________________________________________________________________
dense_3 (Dense)              (None, 6)                 306
=================================================================
Total params: 6,325,656
Trainable params: 6,325,656
Non-trainable params: 0
_________________________________________________________________
```

Now we define call-backs and train our model

```
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau
checkpoint = ModelCheckpoint('elmo-model-{epoch:03d}-{val_accuracy:03f}.h5', verbose=1, monitor='val_accuracy',
                             save_best_only=True, mode='auto')
reduceLoss = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=2, min_lr=0.0001)
callbacks = [checkpoint,reduceLoss]
```

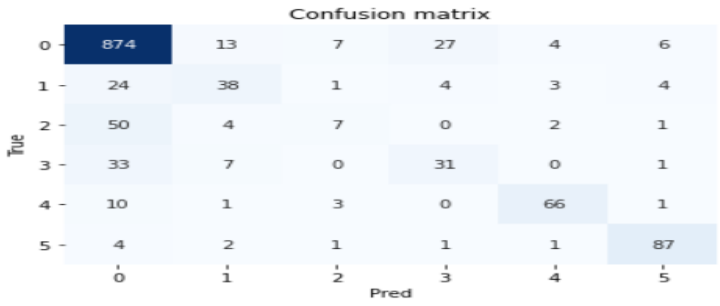
```
model.fit(elmo_train_new, y_train, batch_size=128, epochs=15, callbacks=callbacks,
          validation_data=(elmo_test_new, y_test), verbose=1)
```

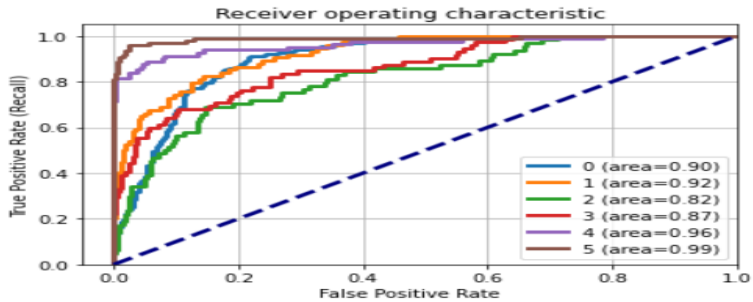
We trained the model for 15 epochs. Below are the prediction results on test set

```
Accuracy: 0.84
Auc: 0.91
Detail:
              precision    recall  f1-score   support

           0       0.88      0.94      0.91       931
           1       0.58      0.51      0.55        74
           2       0.37      0.11      0.17        64
           3       0.49      0.43      0.46        72
           4       0.87      0.81      0.84        81
           5       0.87      0.91      0.89        96

    accuracy                           0.84      1318
   macro avg       0.68      0.62      0.64      1318
weighted avg       0.81      0.84      0.82      1318
```





We note that our model accuracy and AUC has not improved much as compared to Glove, Word2Vec and Fast-text Embeddings. Model has performed the best with Glove embeddings till now.

10. BERT

a) With Pooled Output:

We use the tensorflow hub library to download pre-trained model and fine-tune it. We use BERT of below version:

2 Encoders (L-2) with 128 length embedding for each word (H-128) and 2 Attention Heads (A-2).

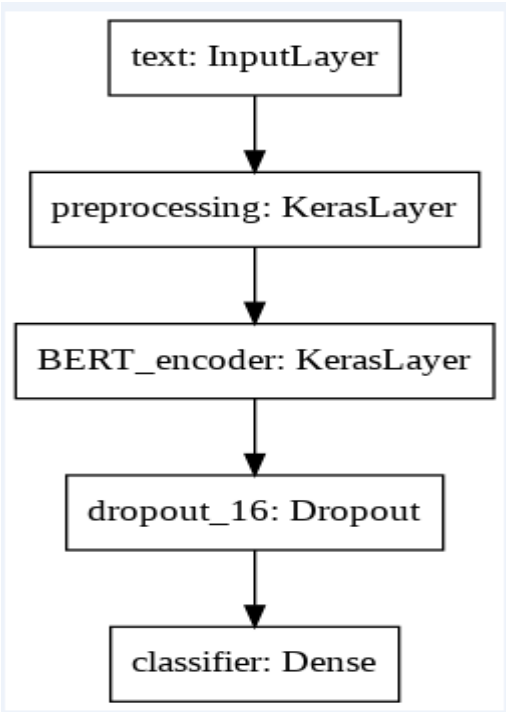
```
tfhub_handle_encoder = "https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-2_H-128_A-2/1"
tfhub_handle_preprocess = "https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/2"
```

We define a function to create a model with BERT encoding and dense layer network.

```
def build_classifier_model():
  text_input = tf.keras.layers.Input(shape=(), dtype=tf.string, name='text')
  preprocessing_layer = hub.KerasLayer(tfhub_handle_preprocess, name='preprocessing')
  encoder_inputs = preprocessing_layer(text_input)
  encoder = hub.KerasLayer(tfhub_handle_encoder, trainable=True, name='BERT_encoder')
  outputs = encoder(encoder_inputs)
  net = outputs['pooled_output']
  net = tf.keras.layers.Dropout(0.1)(net)
  net = tf.keras.layers.Dense(pd.Series(y_train).nunique(), activation='softmax', name='classifier')(net)
  return tf.keras.Model(text_input, net)
```

```
model = build_classifier_model()
```

Below is the model flow representation:



Now we define loss, metrics and compile it

```
# Compile the model
model.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metrics=["accuracy"]) # Compile the
model.summary()
```

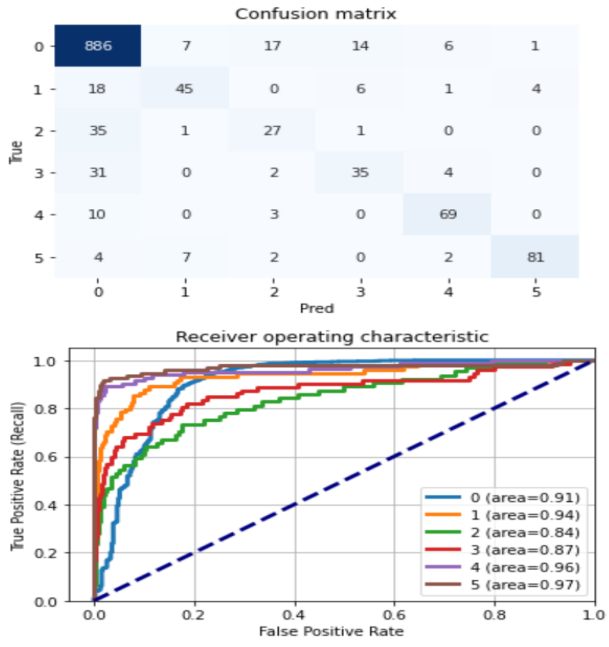
```
Model: "model_12"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
text (InputLayer)               [(None,)]            0
__________________________________________________________________________________________________
preprocessing (KerasLayer)      {'input_word_ids': ( 0           text[0][0]
__________________________________________________________________________________________________
BERT_encoder (KerasLayer)       {'encoder_outputs':  4385921     preprocessing[0][0]
                                                                 preprocessing[0][1]
                                                                 preprocessing[0][2]
__________________________________________________________________________________________________
dropout_16 (Dropout)            (None, 128)          0           BERT_encoder[0][3]
__________________________________________________________________________________________________
classifier (Dense)              (None, 6)            774         dropout_16[0][0]
==================================================================================================
Total params: 4,386,695
Trainable params: 4,386,694
Non-trainable params: 1
__________________________________________________________________________________________________
```

Now we define call-backs and train our model

```
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau
checkpoint = ModelCheckpoint('bert-L-2_H-128_A-2-model-{epoch:03d}-{val_accuracy:03f}.h5', verbose=1,
                             monitor='val_accuracy', save_best_only=True, mode='auto')
reduceLoss = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=2, min_lr=0.0001)
callbacks = [checkpoint,reduceLoss]
```

```
model.fit(X_train, y_train, batch_size=128, epochs=20, callbacks=callbacks,
          validation_data=(X_test, y_test), verbose=1)
```

We trained the model for 20 epochs. Below are the prediction results on test set



We note that our model accuracy and AUC has improved a bit when compared to other models that are tried till now. Still there is a scope of improvement.

b) With Sequence Output + Resampling:

We use the same above BERT encoder here. But instead of using pooled output we use sequence output from BERT to feed forward neural network. Also, we oversample non-majority classes and feed this as training data to model.

```
from tensorflow.keras.layers import TimeDistributed,ReLU,BatchNormalization
from tensorflow.keras.regularizers import L2
from imblearn.over_sampling import RandomOverSampler
```

```
ros=RandomOverSampler(sampling_strategy='not majority', random_state=7)
X_train,y_train=ros.fit_resample(np.array(X_train).reshape(-1, 1),y_train)
```

We define a function to create a model with BERT encoding and dense layer network.

```
def build_classifier_model():
  text_input = tf.keras.layers.Input(shape=(), dtype=tf.string, name='text')
  preprocessing_layer = hub.KerasLayer(tfhub_handle_preprocess, name='preprocessing')
  encoder_inputs = preprocessing_layer(text_input)
  encoder = hub.KerasLayer(tfhub_handle_encoder, trainable=True, name='BERT_encoder')
  outputs = encoder(encoder_inputs)
  net = outputs['sequence_output']
  model = TimeDistributed(Dense(units=64,name='Time_Distributed_Dense_Layer_1',kernel_regularizer=L2(l2=0.001)
  model = TimeDistributed(Dropout(rate=0.2,seed=seed))(model)
  model = TimeDistributed(BatchNormalization(axis=-1, momentum=0.99,name='Time_Distributed_Batch_Norm_Layer_1'
  model = TimeDistributed(ReLU())(model)
  model = TimeDistributed(Dense(units=32,name='Time_Distributed_Dense_Layer_2',kernel_regularizer=L2(l2=0.001)
  model = TimeDistributed(Dropout(rate=0.2,seed=seed))(model)
  model = TimeDistributed(BatchNormalization(axis=-1, momentum=0.99,name='Time_Distributed_Batch_Norm_Layer_2'
  model = TimeDistributed(ReLU())(model)
  model = Flatten()(model) # Flatten
  model = Dense(64, activation='relu')(model) # Dense layer 3
  model = Dropout(0.2)(model) # Dropout 3
  out = Dense(pd.Series(y_train).nunique(), activation='softmax')(model)
  return tf.keras.Model(text_input, out)
```

```
model = build_classifier_model()
```

Now we define loss, metrics and compile it

```
# Compile the model
model.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metrics=["accuracy"]) # Compile the mo
model.summary()
```

```
Layer (type)                 Output Shape              Param #   Connected to
==================================================================================================
text (InputLayer)            [(None,)]                 0
__________________________________________________________________________________________________
preprocessing (KerasLayer)   {'input_word_ids': (  0           text[0][0]
__________________________________________________________________________________________________
BERT_encoder (KerasLayer)    {'pooled_output': (N 4385921     preprocessing[0][0]
                                                              preprocessing[0][1]
                                                              preprocessing[0][2]
__________________________________________________________________________________________________
time_distributed_16 (TimeDistri (None, 128, 64)      8256      BERT_encoder[0][4]
__________________________________________________________________________________________________
time_distributed_17 (TimeDistri (None, 128, 64)      0         time_distributed_16[0][0]
__________________________________________________________________________________________________
time_distributed_18 (TimeDistri (None, 128, 64)      256       time_distributed_17[0][0]
__________________________________________________________________________________________________
time_distributed_19 (TimeDistri (None, 128, 64)      0         time_distributed_18[0][0]
__________________________________________________________________________________________________
time_distributed_20 (TimeDistri (None, 128, 32)      2080      time_distributed_19[0][0]
__________________________________________________________________________________________________
time_distributed_21 (TimeDistri (None, 128, 32)      0         time_distributed_20[0][0]
__________________________________________________________________________________________________
time_distributed_22 (TimeDistri (None, 128, 32)      128       time_distributed_21[0][0]
__________________________________________________________________________________________________
time_distributed_23 (TimeDistri (None, 128, 32)      0         time_distributed_22[0][0]
__________________________________________________________________________________________________
flatten_2 (Flatten)          (None, 4096)              0         time_distributed_23[0][0]
__________________________________________________________________________________________________
dense_4 (Dense)              (None, 64)                262208    flatten_2[0][0]
__________________________________________________________________________________________________
dropout_8 (Dropout)          (None, 64)                0         dense_4[0][0]
__________________________________________________________________________________________________
dense_5 (Dense)              (None, 6)                 390       dropout_8[0][0]
==================================================================================================
Total params: 4,659,239
Trainable params: 4,659,046
Non-trainable params: 193
__________________________________________________________________________________________________
```

Now we define call-backs and train our model

```
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau
checkpoint = ModelCheckpoint('bert-resample-seqout-L-2_H-128_A-2-model-{epoch:03d}-{val_accuracy:03f}.h5',
                             verbose=1, monitor='val_accuracy', save_best_only=True, mode='auto')
reduceLoss = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=2, min_lr=0.0001)
callbacks = [checkpoint,reduceLoss]
```

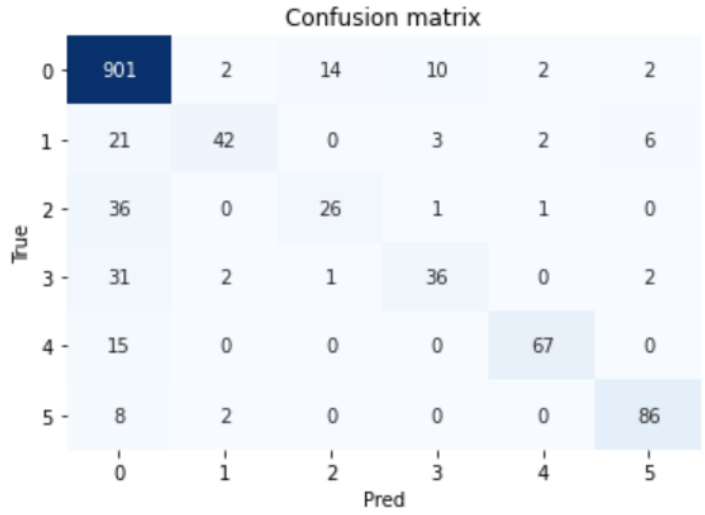
```
model.fit(X_train, y_train, batch_size=128, epochs=20, callbacks=callbacks,
          validation_data=(X_test, y_test), verbose=1)
```

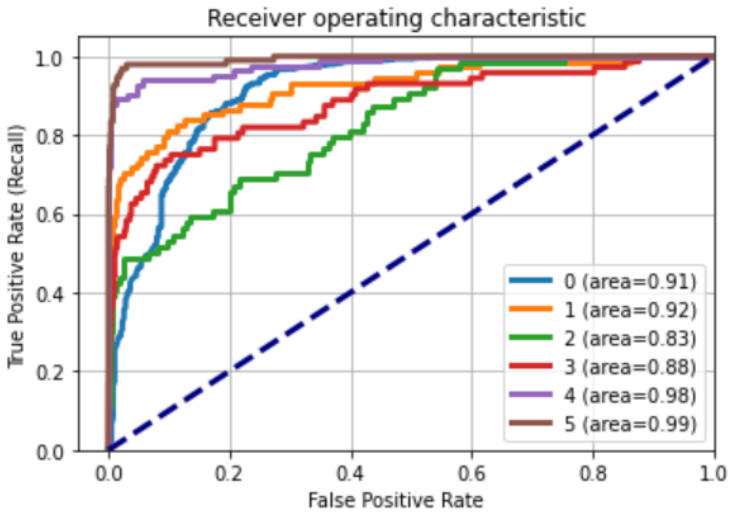
We trained the model for 20 epochs. Below are the prediction results on test set

```
Accuracy: 0.88
Auc: 0.92
Detail:
              precision    recall  f1-score   support

           0       0.89      0.97      0.93       931
           1       0.88      0.57      0.69        74
           2       0.63      0.41      0.50        64
           3       0.72      0.50      0.59        72
           4       0.93      0.82      0.87        82
           5       0.90      0.90      0.90        96

    accuracy                           0.88      1319
   macro avg       0.82      0.69      0.74      1319
weighted avg       0.87      0.88      0.87      1319
```





We note that our model accuracy and AUC has improved further when compared to BERT with pooled output.

MODEL EVALUATION

Stats for model evaluation:

Index	Model Name	Dataset	Re-Sampling(Y/N)	Call-backs	Test Accuracy %	Train Accuracy %
1	Multinomial Naïve Bayes	All groups	N	NA	54	
2	Multinomial Naïve Bayes	All groups	Y	NA	50	
3	Multinomial Naïve Bayes	Top 6 groups	N	NA	81	
4	Multinomial Naïve Bayes	Top 10 groups	N	NA	72	
5	Word2Vec Bi-LSTM	Top 6 groups	N	Y	84	93.89
6	Glove Bi-LSTM	Top 6 groups	N	Y	86	97.04
7	Fast-text Bi-LSTM	Top 6 groups	N	Y	84	93.3
8	Elmo Bi-LSTM	Top 6 groups	N	Y	84	91.02
9	BERT pooled	Top 6 groups	N	Y	86	99.17
10	BERT sequence output	Top 6 groups	Y	Y	88	99.9

From these stats, we can infer that Glove with Bi-LSTM and BERT with resampled training datasets perform the best with test accuracies going as high as 86% and 88% respectively. But, we choose to go ahead with BERT as it is marginally ahead of Glove with Bi-LSTM and might take care of some scenarios where Glove fails as it looks in both directions of the word and sentences to get the context.

COMPARISON TO BENCHMARK

As we saw during model evaluation, the baseline model trained using multinomial naïve bayes using all data groups had a test accuracy of only 54%, we then tried to use the same model on resampled data, although it did improve the f1 scores of all assignment groups, but still the test accuracy was only 50%. We then observed that many groups had a really low number of training rows, so got rid of them and chose to go ahead with building models for only top 6 groups and we then achieved a test accuracy of about 81% using the same multinomial naïve bayes model for top 6 groups.

To further improve upon the baseline model we went ahead with deep learning models with different word embeddings – Word2Vec, Glove, Fast-text, Elmo. Glove being the best among all with Bi-LSTM gave a test accuracy of about 86%.

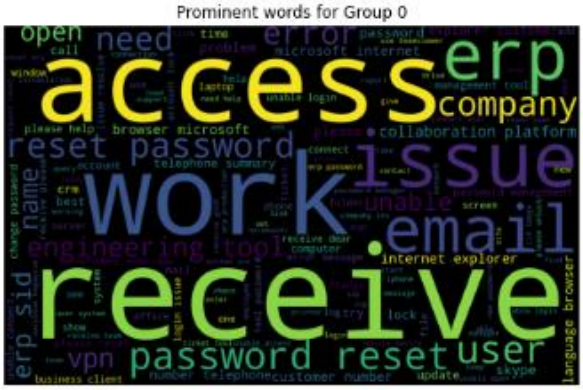
Furthermore, we also tried state-of-the-art transformer model BERT with resampled train data which gave us the best test accuracy, scoring 88%.

Hence, we were able to improve upon our baseline test accuracy of 54% to 88% by a bit of feature engineering, considering only top 6 groups, hyper-parameter optimizations and choosing state-of-the-art models to train.

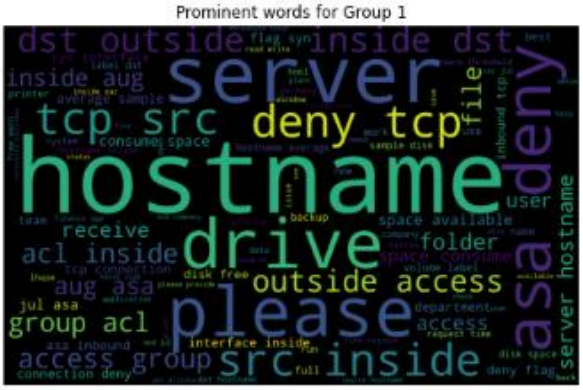
VISUALIZATIONS

In addition to all the graphs and charts, text-specific visualizations such as word clouds provide a deeper insight into the text data. Below are most prominent for each of the assignment groups

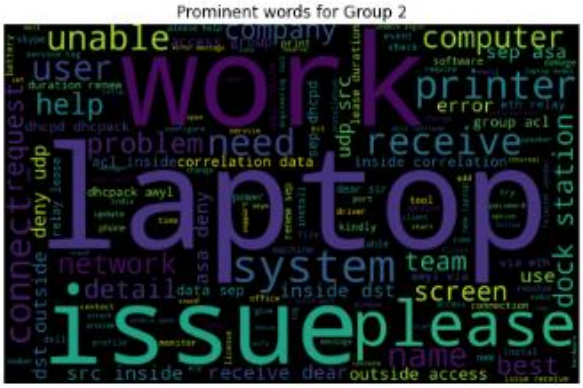
Prominent words for Group 0



Prominent words for Group 1

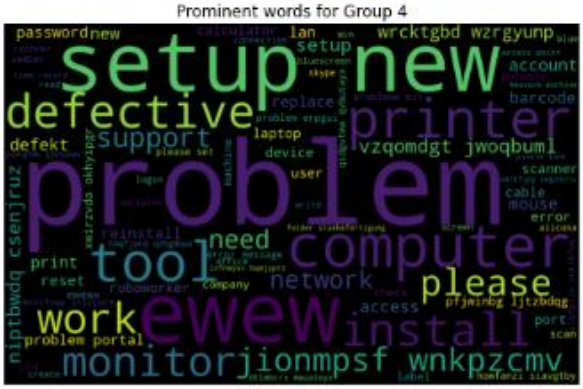


Prominent words for Group 2

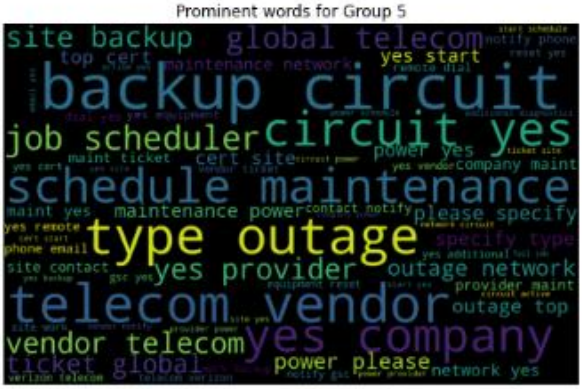


Prominent words for Group 3

Prominent words for Group 4



Prominent words for Group 5



IMPLICATIONS

Our solution uses a model which is trained to classify tickets belonging to the most frequent 6 assignment groups with 88% accuracy. This means, we cover ~65% of the tickets in the dataset and the rest 35% needs to be classified using some other method.

Our solution provides immense value to the business. Around 25-30% incidents require 15 minutes per incident for their assignment to functional groups. Even then, ~25% of tickets are wrongly assigned.

Our solution accomplishes this task in a small fraction of that time and with much better accuracy.

LIMITATIONS

Following are our solution's limitations:

1. Because of very few ticket samples for most of the assignment groups our model only works for the top 6 most frequent groups which comprises ~65% of the tickets. This can be overcome by adding enough samples for all the groups.
2. If any dominant assignment group is added, our model would require a complete retraining. This can be overcome by using incremental model updates / online training by reloading the saved weights and performing training for just the new group.
3. Our solution uses language detection and translation which is done using relatively lesser known open-source libraries. This dependency needs to be understood carefully and should be replaced with enterprise-grade tools.

CLOSING REFLECTIONS

We found that the data was present in multiple languages and had contacts, email-ids, chats, hyperlinks, non-ascii characters bringing in a lot of variability in the data that had to be trained for model building. Moreover, most groups had less than 40 tickets assigned to them which made model training on those groups difficult. So, we finally chose only the top 6 groups and trained the model on them. However, the business can improve the process of raising tickets through a unified IT Service portal to have some sort of a drop-down menu where users can select which product has problems, that way above variations may be controlled to a limit. And if we could capture more training examples for all groups that had a really low number of training examples, we could have trained the model better which could help the business further.