Automatic Ticket Assignment

Problem

One of the key activities of any IT function is to ensure there is no impact to the Business operations. **IT leverages the Incident Management process to achieve the above Objective.** An incident is something that is an unplanned interruption to an IT service or reduction in the quality of an IT service that affects the Users and the Business. ***The main goal of the Incident Management process is to provide a quick fix / workarounds or solutions that resolves the interruption and restores the service to its full capacity to ensure no business impact.***

In most of the organizations, incidents are created by various Business and IT Users, End Users/ Vendors if they have access to ticketing systems, and from the integrated monitoring systems and tools. **Assigning the incidents to the appropriate person or unit in the support team has critical importance to provide improved user satisfaction while ensuring better allocation of support resources.**

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.

Objective

We have to build a model using AI techniques to make the process automated and less error prone so as to save on both time and effort .

The goal is to data pre-processing , EDA and necessary visualization . We will then proceeed to model building and testing the model. All this will be done using NLP techniques and try out different models so that we can get the best accuracy.

Table of Content

Business Domain Value

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

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

L1 / L2 needs to spend time reviewing Standard Operating Procedures (SOPs) before assigning to Functional teams (Minimum ~25-30% of incidents needs to be reviewed for SOPs before ticket assignment). 15 min is being spent for SOP review for each incident. Minimum of ~1 FTE effort needed only for incident assignment to L3 teams. During the process of incident assignments by L1 / L2 teams to functional groups, there were multiple instances of incidents getting assigned to wrong functional groups. Around ~25% of Incidents are wrongly assigned to functional teams. Additional effort needed for Functional teams to re-assign to right functional groups. During this process, some of the incidents are in queue and not addressed timely resulting in poor customer service.

Milestone

1. Milestone 1: Pre-Processing, Data Visualisation and EDA
    1. Exploring the given Data files
    2. Understanding the structure of data
    3. Missing points in data
    4. Finding inconsistencies in the data
    5. Visualizing different patterns
    6. Visualizing different text features
    7. Dealing with missing values

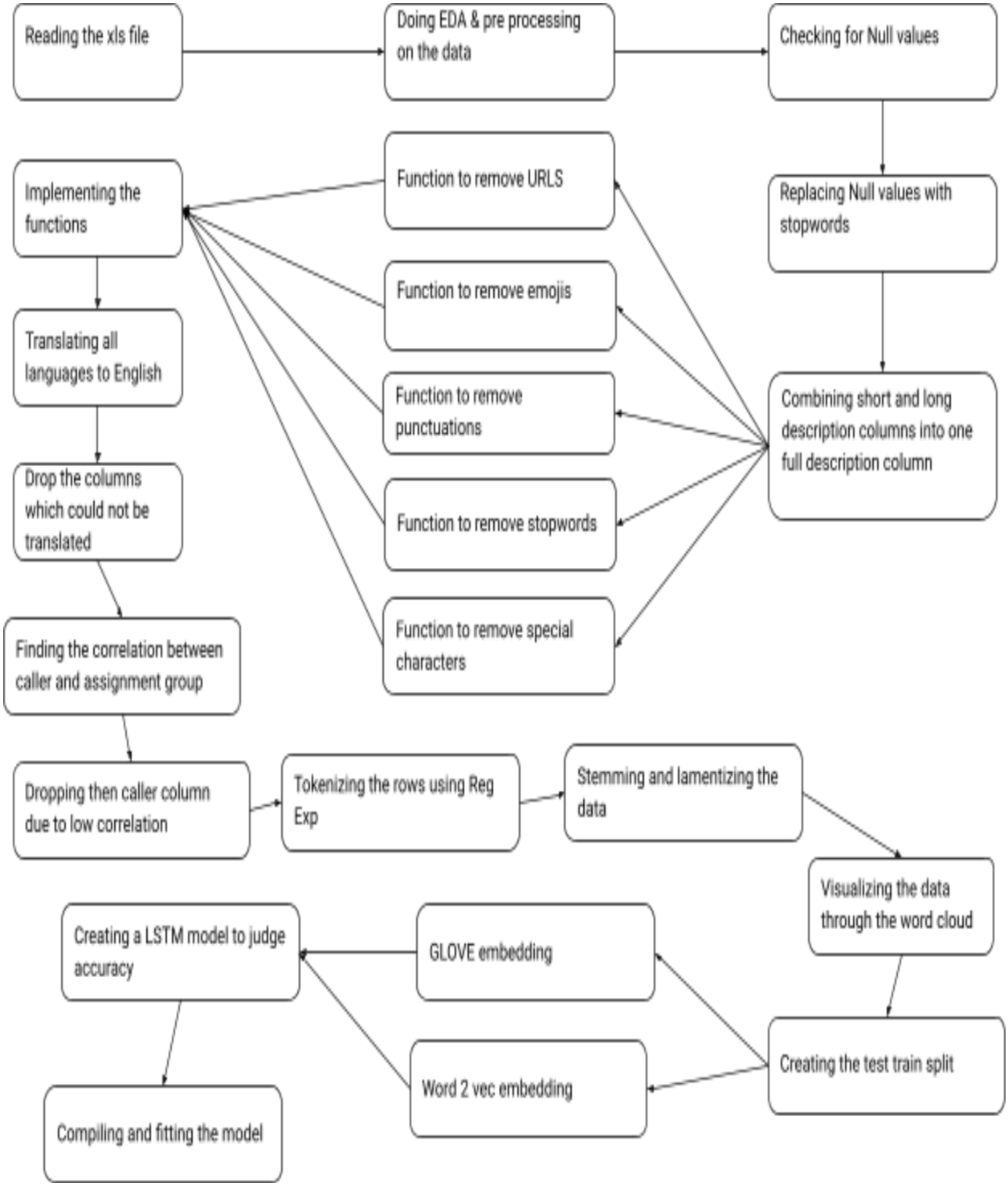
8. Text preprocessing
9. Creating word vocabulary from the corpus of report text data
10. Creating tokens as required

2. Milestone 2: Model Building
    1. Building a model architecture which can classify.
    2. Trying different model architectures by researching state of the art for similar tasks.
    3. Train the model
    4. To deal with large training time, save the weights so that you can use them when training the model for the second time without starting from scratch.
3. Milestone 3: Test the Model, Fine-tuning and Repeat
    1. Test the model and report as per evaluation metrics
    2. Try different models
    3. Try different evaluation metrics
    4. Set different hyper parameters, by trying different optimizers, loss functions, epochs, learning rate, batch size, checkpointing, early stopping etc..for these models to fine-tune them
    5. Report evaluation metrics for these models along with your observation on how changing different hyper parameters leads to change in the final evaluation metric.

Team Member

1. Anurag Mohapatra
2. Pathuri Chanukya
3. Sumedh Ojha
4. Shubham Sharma
5. Ravi Kiran Shashidharan

Workflow Diagram



PREPROCESSING

Using Pandas we read the excel file and then create a dataframe to do basic EDA and preprocessing. We read the shape of the file and have a view of the data. We also check for NAN and null values.

Here we have an Assignment group as the output column which we will use for future predictions.

We check for the languages in the description column and the type of data present in the columns

We also check for characters and words other than English that are present. We have to remove the characters incase they are present

We check for the possibility of combining two columns (short description & description) to reduce the number of columns

The correlation between columns caller and assignment group is checked

INSIGHTS

Reading the file and analyzing the data below are the key elements that we found :

No of columns : 4

Data type of the columns : object

Output column : Assignment group

Shape of the data : (8500,4)

While analyzing the columns and checking for null values , we have 8 Null values in the Short_description and 1 null value for description.

We replace the null values with words that could be removed and hence will have no impact on the data already there .

Language : 51% English and others non English

Description contains special characters, emojis, URLs , HTML and stopwords

The columns short_ description and description could be combined

The correlation between caller and assignment column is 0 .11

CLEANING THE DATA & EDA

We clean the data by removing the emojis, special characters, URLs and HTML data present

The non English words are then translated to English

We check to see if we have skewed data in the column assignment group

We also separate the assignment group column

INSIGHTS

Functions are defined for removing HTML ,URLs , emojis, stopwords and special characters

To translate the language to English we convert all the rows to English using Microsoft translator

5 rows could not be converted so we drop those rows

The new shape is (8495,2)

Assignment group is dropped and assigned to a blank list separately

VISUALIZING THE DATA

To visualize the data we use world cloud

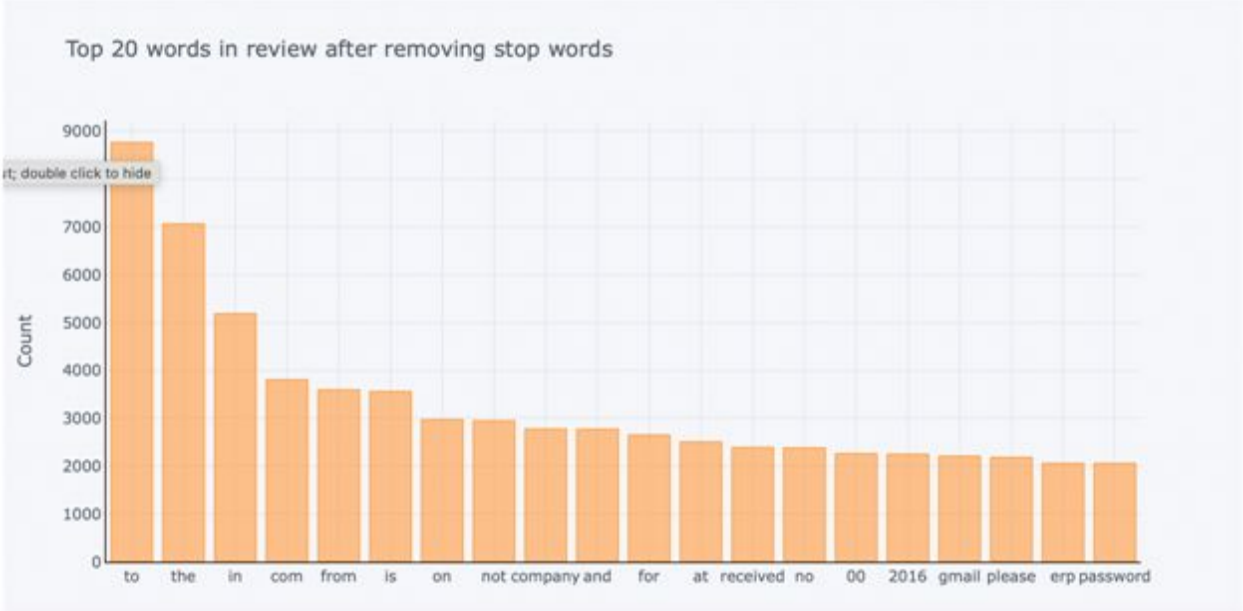
We also find the top 20 words

INSIGHTS

Below is the world cloud

As it can be seen erp,received , job scheduler occur frequently

Below are Top 20 Words in the Dataset



TOKENIZING & LEMMATIZING

We use keras word preprocessing tokenizer to tokenize the string.

To get to the root words we lemmatize the data

INSIGHTS

We create 26288 tokens

GLOVE EMBEDDINGS

Since our model is a neural network in order to find similarity between the words we need to get the weights of the words present

By using this we can get an embedding matrix

INSIGHTS

We use the Wikipedia 200 dimension weights to generate the embedding matrix

Each token is now assigned 200 dimensions which would be used as weights while feeding the tokens to model

WORD 2 VEC

We use word 2 vec to get another embedding matrix and choose the better one between GLOVE and WORD 2 VEC as to which one generates the more accuracy

INSIGHTS

We use pre-trained weight matrix generated using Google news

We get 4422 tokens each having 300 dimensions

GLOVE embeddings generate more accuracy so we are using them rather than word 2 vec

CREATING THE MODEL

The data is split into train , test and validation data

LSTM is to be used to get the contextual data as the data is in the form of sentences.

We also used Bi-Directional LSTM with more Layers and 300 dimensional Glove embeddings but, it didn’t improve the model performance better than the one mentioned below.

We create a LSTM model using the GLOVE embeddings , then we generate the model summary and get the model accuracy.

INSIGHTS

We use a sequential model using 200 dimensional embedding in the embedding layer, then we create a drop out layer along with LSTM followed by dense layers .

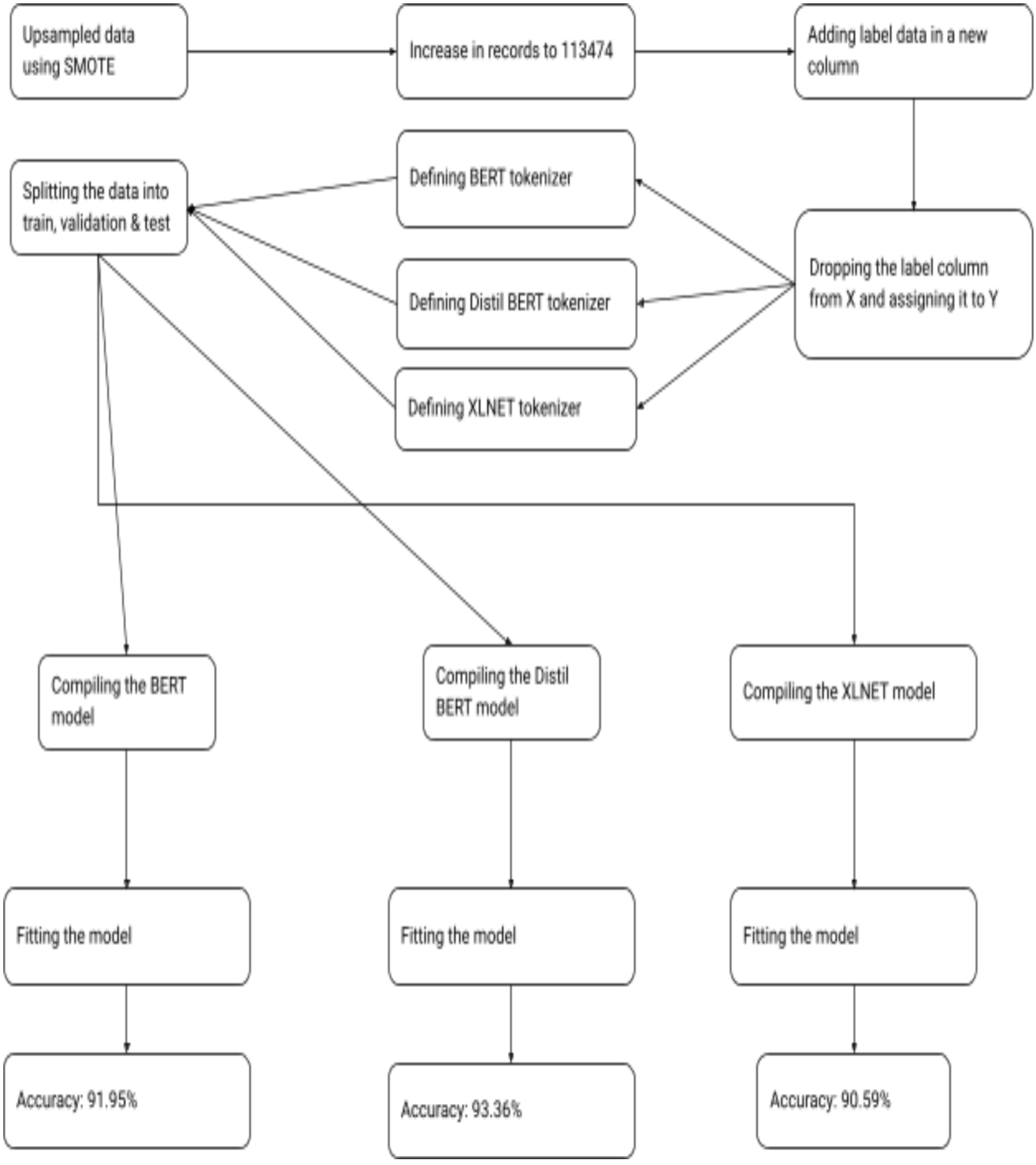
We are using the ‘sigmoid’ function to get the output.

Below is the model summary:

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 4, 200)            5257600
_________________________________________________________________
spatial_dropout1d_2 (Spatial (None, 4, 200)            0
_________________________________________________________________
lstm_2 (LSTM)                (None, 4, 128)            168448
_________________________________________________________________
spatial_dropout1d_3 (Spatial (None, 4, 128)            0
_________________________________________________________________
lstm_3 (LSTM)                (None, 100)               91600
_________________________________________________________________
flatten_1 (Flatten)          (None, 100)               0
_________________________________________________________________
dense_3 (Dense)              (None, 15)                1515
_________________________________________________________________
dropout_1 (Dropout)          (None, 15)                0
_________________________________________________________________
dense_4 (Dense)              (None, 15)                240
_________________________________________________________________
dense_5 (Dense)              (None, 74)                1184
=================================================================
Total params: 5,520,587
Trainable params: 5,520,587
Non-trainable params: 0
```

Currently we are getting an accuracy of 57 while fitting the model

MODEL WORKFLOW



PREPROCESSING THE DATA

Upsampling the data as the data was tilted heavily towards Group 0 and the rest of the groups having fewer data. Post upsampling we have 77476 number of rows

INSIGHTS

Shape: 113474, 6

The samples for all groups except Group GRP_0 is 1500

All the groups with rows less than 1000 have 1500 rows post upsampling

Upsampling was necessary as even with 1000 records the accuracy was 68 percent

Here we have taken the top 50 groups

CLEANING THE DATA & EDA

We clean the data by removing the emojis, special characters, URLs and HTML data present

The non English words are then translated to English

We check to see if we have skewed data in the column assignment group

We also separate the assignment group column

INSIGHTS

Functions are defined for removing HTML, URLs, emojis, stopwords and special characters

To translate the language to English we convert all the rows to English using Microsoft translator

Assignment group is dropped and assigned to a blank list separately

SELECTION OF MODELS

We go for pre train models. We have used **BERT, DistilBERT and XLNET**

BERT: A bidirectional neural network learns from both the right and left side of the token. It is very ef-fective in catching the context in which the word is written. So it is highly effective.

DistillBERT: Distilled model, **DistilBERT**, has **about half** the total number of parameters of BERT base and retains 95% of BERT's performances on the language understanding part.

XLNET: **XLNet** is a method of learning language representation using the generalized autoregressive pretraining method. Its objective is to learn the language model. It has been trained on a large corpus using the permutation language modeling objective.

INSIGHTS

Bert is pre-trained model trained on bert based uncased

Distilbert is pre trained on Distilbert based uncased

XLNET is pre trained on XLNet based cased

SPLITTING THE DATA

Data is split into Train test and validation data

INSIGHTS

Validation data: 20 %

Training data: 80%

Test data: 1% of training data

TOKENIZATION

BERT tokenization is used for BERT

DistilBERT Tokenization is used for DistilBERT

XLNET Tokenization is used for XLNET

INSIGHTS

Each model has its own tokenization function so generate their own respective tokens

COMPILING THE MODELS

All three models are complied and summary is generated

INSIGHTS

Optimizer used is Adam

BERT summary:

```
Model: "tf_bert_for_sequence_classification"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
bert (TFBertMainLayer)       multiple                  109482240
_________________________________________________________________
dropout_75 (Dropout)         multiple                  0
_________________________________________________________________
classifier (Dense)           multiple                  38450
=================================================================
Total params: 109,520,690
Trainable params: 109,520,690
Non-trainable params: 0
_________________________________________________________________
```

DistilBERT summary:

```
Model: "tf_distil_bert_for_sequence_classification"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
distilbert (TFDistilBertMain multiple                  66362880
_________________________________________________________________
pre_classifier (Dense)       multiple                  590592
_________________________________________________________________
classifier (Dense)           multiple                  38450
_________________________________________________________________
dropout_95 (Dropout)         multiple                  0
=================================================================
Total params: 66,991,922
Trainable params: 66,991,922
```

XLNET summary:

```
Model: "tfxl_net_for_sequence_classification_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
transformer (TFXLNetMainLaye multiple                  116718336
_________________________________________________________________
sequence_summary (TFSequence multiple                  590592
_________________________________________________________________
logits_proj (Dense)          multiple                  38450
=================================================================
Total params: 117,347,378
Trainable params: 117,347,378
Non-trainable params: 0
_________________________________________________________________
```

LOSS AND ACCURACY

DistlBERT gives the best output, followed by BERT and then XLNET.

INSIGHTS

Bert had an accuracy of 91.95%

```
Epoch 1/5
1228/1228 [==============================] - 195s 101ms/step - loss: 2.4971 - accuracy: 0.3510 - val_loss: 0.3954 - val_accuracy: 0.8815
Epoch 2/5
1228/1228 [==============================] - 77s 62ms/step - loss: 0.3572 - accuracy: 0.8916 - val_loss: 0.3219 - val_accuracy: 0.9043
Epoch 3/5
1228/1228 [==============================] - 76s 62ms/step - loss: 0.2769 - accuracy: 0.9127 - val_loss: 0.2231 - val_accuracy: 0.9289
Epoch 4/5
1228/1228 [==============================] - 77s 62ms/step - loss: 0.2535 - accuracy: 0.9170 - val_loss: 0.2040 - val_accuracy: 0.9328
Epoch 5/5
1228/1228 [==============================] - 77s 63ms/step - loss: 0.2457 - accuracy: 0.9195 - val_loss: 0.2329 - val_accuracy: 0.9282
<tensorflow.python.keras.callbacks.History at 0x7ff8055a6b90>
```

Distilbert had an accuracy of 93.365%

```
Epoch 1/5
1228/1228 [==============================] - 122s 63ms/step - loss: 1.6680 - accuracy: 0.5813 - val_loss: 0.2698 - val_accuracy: 0.9117
Epoch 2/5
1228/1228 [==============================] - 47s 38ms/step - loss: 0.2610 - accuracy: 0.9159 - val_loss: 0.2251 - val_accuracy: 0.9257
Epoch 3/5
1228/1228 [==============================] - 47s 39ms/step - loss: 0.2041 - accuracy: 0.9299 - val_loss: 0.2096 - val_accuracy: 0.9270
Epoch 4/5
1228/1228 [==============================] - 47s 38ms/step - loss: 0.2081 - accuracy: 0.9291 - val_loss: 0.2240 - val_accuracy: 0.9253
Epoch 5/5
1228/1228 [==============================] - 47s 38ms/step - loss: 0.1910 - accuracy: 0.9336 - val_loss: 0.2001 - val_accuracy: 0.9349
<tensorflow.python.keras.callbacks.History at 0x7ff76e399350>
```

XLNET had an accuracy of 90.59%

```
Epoch 1/5
1228/1228 [==============================] - 240s 133ms/step - loss: 2.2081 - accuracy: 0.4118 - val_loss: 0.4958 - val_accuracy: 0.8458
Epoch 2/5
1228/1228 [==============================] - 103s 84ms/step - loss: 0.3967 - accuracy: 0.8832 - val_loss: 0.3178 - val_accuracy: 0.9035
Epoch 4/5
1228/1228 [==============================] - 103s 84ms/step - loss: 0.3756 - accuracy: 0.8871 - val_loss: 0.2538 - val_accuracy: 0.9184
Epoch 5/5
 844/1228 [===================>..........] - ETA: 29s - loss: 0.3102 - accuracy: 0.9059IOPub message rate exceeded.
The notebook server will temporarily stop sending output
to the client in order to avoid crashing it.
to change this limit, set the config variable
`--NotebookApp.iopub_msg_rate_limit`.

Current values:
NotebookApp.iopub_msg_rate_limit=1000.0 (msgs/sec)
NotebookApp.rate_limit_window=3.0 (secs)
```

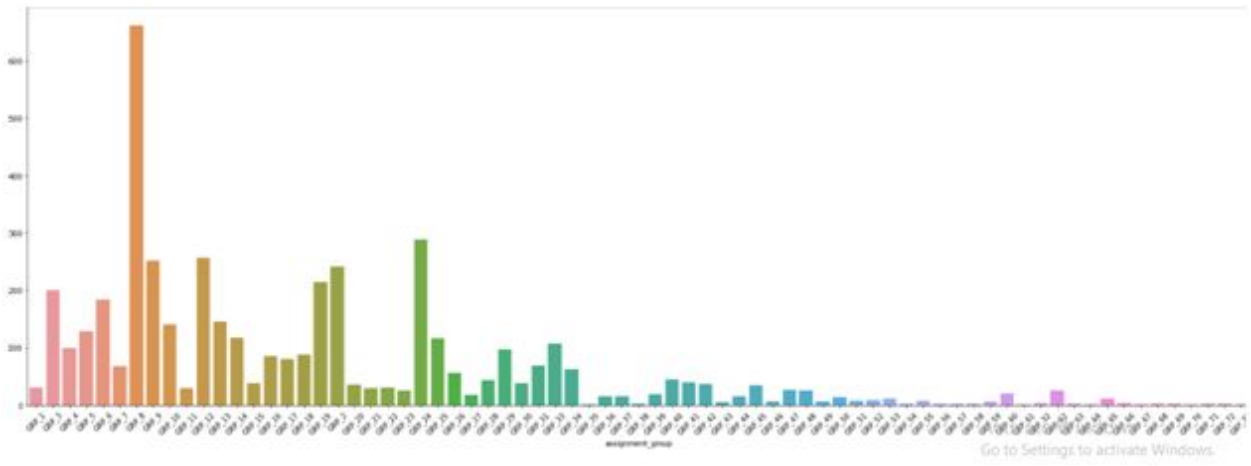
CONCLUSION

DistilBERT gives the best accuracy of the three models and is the model to select over others for the Project.

FINALLY, TRAINING THE MODEL USING PYTORCH

INITIAL DISTRIBUTION

Distribution of the count of the groups expect Group 0 which has the highest count



UPSAMPLING THE DATA

Using SMOTE the data was upsampled , to increase the count of the rows across all groups.

INSIGHTS

After upsampling all the Classes have 1500 rows

Dropping assignment group column from X and using it in Y

TOKENIZING

Tokenizing using DistilBERT

INSIGHTS

Using Pre-trained model trained in –DISTILBERT base uncased

TOKENIZING

Splitting the data into Train, Validation and Test data. The data is also Split for masks

SPLITTING THE DATA

Splitting the data into Train, Validation and Test data. The data is also Split for masks

INSIGHTS

The validation Split is 30 percent of the data

The test Split is 20 percent of the train data

CONVERTING TO TENSORS

All the train test data is converted to Tensors

INSIGHTS

Defining the model DISTILBERT and the optimizer used is ADAM

Defining the function epoch time to note the time lapsed for epochs

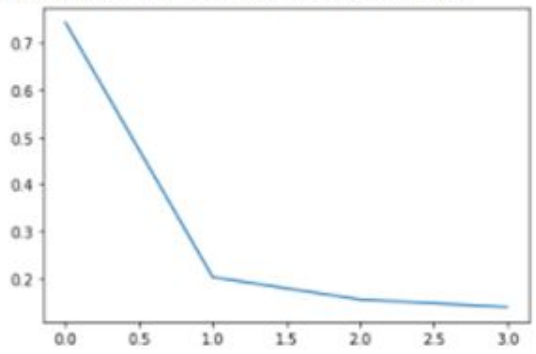
MODEL TUNING

Training the model on our pre-processed data and printing the train and validation losses after each epoch

INSIGHTS

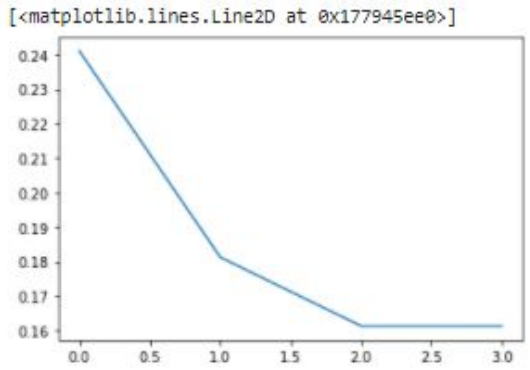
```
Train loss after itaration 1: 0.742989
Validation loss after itaration 1: 0.241045
Time: 549m 42s

Train loss after itaration 2: 0.201923
Validation loss after itaration 2: 0.181267
Time: 636m 41s

Train loss after itaration 3: 0.154447
Validation loss after itaration 3: 0.161316
Time: 722m 50s

Train loss after itaration 4: 0.138329
Validation loss after itaration 4: 0.161327
Time: 1003m 29s
```

```
[<matplotlib.lines.Line2D at 0x17b4895b0>]
```



Above is the train loss graph

Below is the validation loss graph



ACCURACY

Accuracy on Train Data: 94.9%

Accuracy on Val Data: 94.7%

Accuracy on Test Data: 94.05%

CLASSIFICATION REPORT

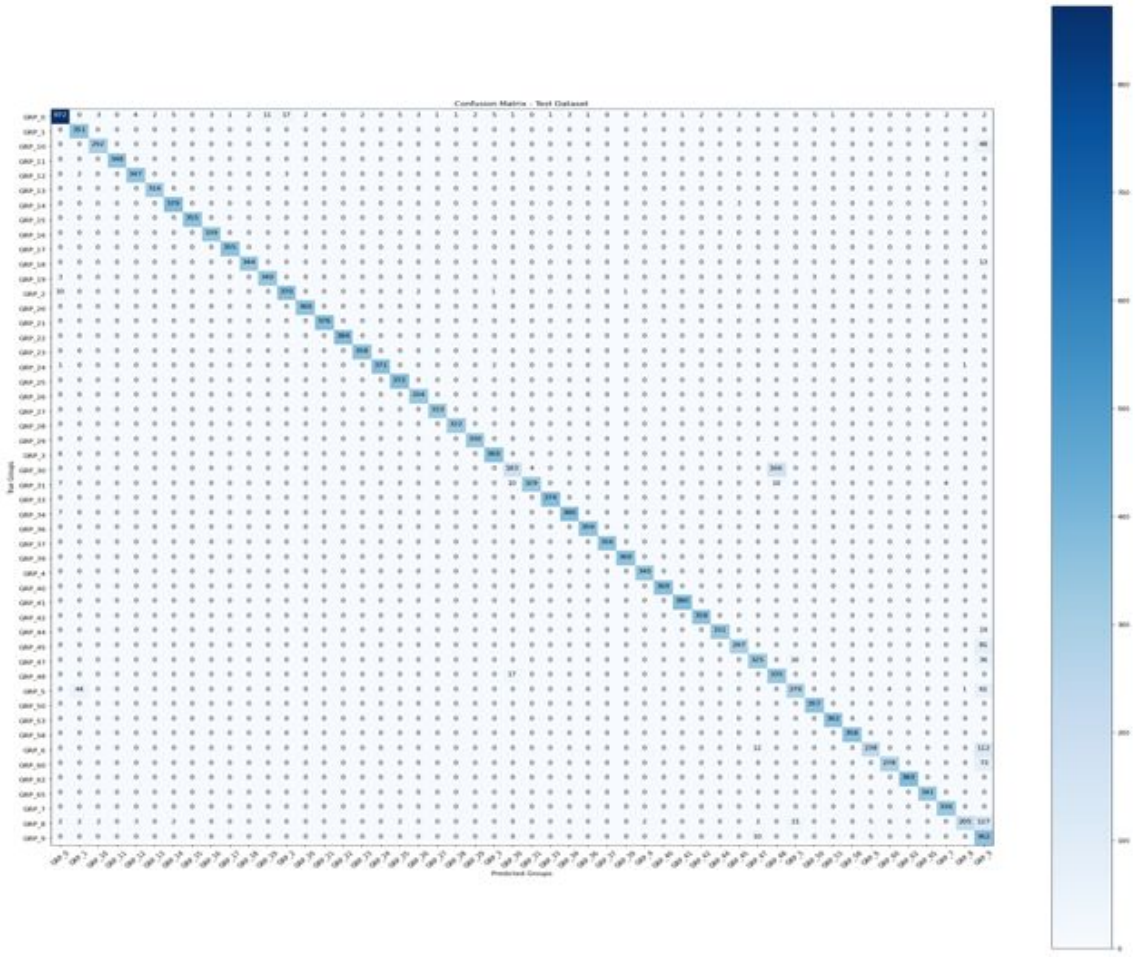
A classification report is drawn as below

INSIGHTS

Group 9 and Group 48 have low precision

Group 9, Group 8, Group 60, Group 6, Group 5 , Group 45, Group 30 have low recall compared to others

	precision	recall	f1-score	support
GRP_0	0.96	0.90	0.93	970
GRP_1	0.88	1.00	0.94	351
GRP_10	0.98	0.86	0.92	340
GRP_11	1.00	1.00	1.00	348
GRP_12	0.98	0.95	0.97	365
GRP_13	0.99	0.98	0.99	322
GRP_14	0.98	0.98	0.98	385
GRP_15	1.00	1.00	1.00	355
GRP_16	0.99	1.00	1.00	339
GRP_17	1.00	1.00	1.00	355
GRP_18	0.99	0.96	0.98	357
GRP_19	0.97	0.97	0.97	349
GRP_2	0.95	0.96	0.96	384
GRP_20	0.99	1.00	1.00	360
GRP_21	0.99	1.00	0.99	376
GRP_22	1.00	1.00	1.00	384
GRP_23	0.99	1.00	1.00	358
GRP_24	1.00	0.99	0.99	375
GRP_25	0.98	1.00	0.99	372
GRP_26	0.99	1.00	0.99	334
GRP_27	1.00	1.00	1.00	333
GRP_28	1.00	1.00	1.00	322
GRP_29	0.99	0.98	0.99	338
GRP_3	0.97	0.98	0.98	375
GRP_30	0.87	0.52	0.65	353
GRP_31	0.99	0.91	0.95	360
GRP_33	1.00	1.00	1.00	374



Above is the Confusion Matrix

CONCLUSION

The Data Up-sampling and Pre-processing DISTILBERT Pre-Trained Model gives the best accuracy here also on the test data with 94.05%. We have saved the model and the training as well as validation losses pickle files for further use. And will deploy the model as a web based service.

APPENDIX

STEPS TAKEN

As the whole thing was done in google colabs.

STEP 1

Mounting the files from the google drive.

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mount("/content/gdrive", force_remount=True).
```

STEP 2

Installing the packages like NLK, langdetect, xlrd , wordcloud etc

```
!pip install nltk
!pip install langdetect
!pip install xlrd==1.2.0
!pip install wordcloud
!pip install cufflinks
```

STEP 3

Importing various packages to process the data and do the necessary operations

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow
from tensorflow.keras.preprocessing.sequence import pad_sequences
import nltk
import re
from nltk.tokenize import word_tokenize
from keras.utils import to_categorical
from sklearn.model_selection import train_test_split
import string
from scipy.stats import chi2_contingency
from langdetect import detect
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MultiLabelBinarizer
import cufflinks as cf
from scipy.stats import chi2_contingency
```

STEP 4

Reading the data from the excel file :

```
[167] data = pd.read_excel("/content/gdrive/MyDrive/input_data.xlsx")
```

STEP 5

Starting with the EDA. Observing various columns and the data in the file. Looking at the first five rows

```
[168] data.head()
```

	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

STEP 6

Looking at the bottom 5 rows

```
data.tail()
```

	Short description	Description	Caller	Assignment group
8495	emails not coming in from zz mail	\r\n\r\nreceived from: avglmrts.vhqmtiua@gmail...	avglmrts vhqmtiua	GRP_29
8496	telephony_software issue	telephony_software issue	rbozivdq gmlhrtvp	GRP_0
8497	vip2: windows password reset for tifpdchb pedx...	vip2: windows password reset for tifpdchb pedx...	oybwdsgx oxyhwrfz	GRP_0
8498	machine nÃ£o estÃ¡ funcionando	i am unable to access the machine utilities to...	ufawcgob aowhxjky	GRP_62
8499	an mehreren pc`s lassen sich verschiedene prgr...	an mehreren pc`s lassen sich verschiedene prgr...	kqvbrspl jyzoklfx	GRP_49

STEP 7

The shape of the dataframe is (8500, 4)

```
data.shape
```

```
(8500, 4)
```

STEP 8

Getting basic info about the columns and their data type

```
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
```

STEP 9

Getting basic descriptive stats about the data

```
data.describe()
```

	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

STEP 10

Renaming the columns to have a common nomenclature

```
# Since Some Columns have capital names and some lowercase names, converting them to identical camelcase names
data.columns = ['short_description', 'long_description', 'caller', 'assignment_group']
```

STEP 11

Checking for null and nan values. Both were found to exist who we will replace them

```
# Checking for null values
data.isnull().values.any()
```

```
True
```

STEP 12

Replacing the na and nan values , IGNORE_TEXT

```
data['short_description'].fillna("IGNORE_TEXT", inplace = True)
data['long_description'].fillna("IGNORE_TEXT", inplace = True)
```

STEP 13

Checking for NAN or NA values again .We find that non exist now

```
# Checking for Nan Values
data.isna().values.any()
```

```
False
```

```
# Checking for null values
data.isnull().values.any()
```

```
False
```

STEP 14

Combines the two description columns into one column

```
[253] data['short_description'] = data['short_description'] + ' '
```

```
data['full_description'] = data['short_description'] + data['long_description']
```

STEP 15

Dropping the short and long description columns

```
[256] data = data.drop(['short_description', 'long_description'], axis=1)
```

STEP 16

Separating the target and feature column

```
[257] Y = data['assignment_group']
      X = data
      X.drop(['assignment_group'], axis=1)
```

	caller	full_description
0	spxjnwir pjlcoqds	login issue -verified user details.(employee# ...
1	hmjdrvpb komuaywn	outlook \r\n\r\nreceived from: hmjdrvpb.komuay...

STEP 17

Printing the shape

```
[258] X.shape

      (8500, 3)

[259] Y.shape

      (8500,)
```

STEP 18

Removing URL from the full_description column

```
[260] def remove_URL(column):
          url = re.compile(r'(https|http)?:\/\/(\w|\.|\/|\?|\=|\&|\%)*\b')
          return url.sub(r'', column)
      X['full_description'] = X['full_description'].apply(remove_URL)
```

STEP 19

Removing HTML tags from the full_description column

```
[261] def remove_HTML(column):
          html=re.compile(r'<.*?>')
          return html.sub(r'',column)

      X['full_description'] = X['full_description'].apply(remove_HTML)
```

STEP 20

Removing Punctuations from the full_description column

```
[262] def remove_punct(column):
          table=str.maketrans('','',string.punctuation)
          return column.translate(table)
      X['full_description'] = X['full_description'].apply(remove_punct)
```

STEP 21

Removing emojis from the long_description column

```
[263] def remove_emojis(column):
          emoj = re.compile("["
              u"\U0001F600-\U0001F64F"  # emoticons
              u"\U0001F300-\U0001F5FF"  # symbols & pictographs
              u"\U0001F680-\U0001F6FF"  # transport & map symbols
              u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
              u"\U00002500-\U000028EF"  # chinese char
              u"\U00002702-\U000027B0"
              u"\U00002702-\U000027B0"
              u"\U000024C2-\U0001F251"
              u"\U0001f926-\U0001f937"
              u"\U00010000-\U0010ffff"
              u"\u2640-\u2642"
              u"\u2600-\u2B55"
              u"\u200d"
              u"\u23cf"
              u"\u23e9"
              u"\u231a"
              u"\ufe0f"  # dingbats
              u"\u3030"
                                "]+", re.UNICODE)
          return re.sub(emoj, '', column)
      X['full_description'] = X['full_description'].apply(remove_emojis)
```

STEP 22

Removing special characters from the long_description column

```
[264] def remove_Specials(column):
          special = re.compile(r'[^A-Za-z0-9 ]+')
          return special.sub(r'', column)

      X['full_description'] = X['full_description'].apply(remove_Specials)
```

STEP 23

Setting up the environmental variables and translating the text to English

```
# Setting Environment Variables
%env key_var_name=4dbd425b5751456295bc03c3c85e2485
%env endpoint_var_name=https://api.cognitive.microsofttranslator.com/
```

```
env: key_var_name=4dbd425b5751456295bc03c3c85e2485
env: endpoint_var_name=https://api.cognitive.microsofttranslator.com/
```

```
import pycurl, json
from io import BytesIO

def translateToEngCurl(text):
    key_var_name = 'key_var_name'
    if not key_var_name in os.environ:
        raise Exception('Please set/export the environment variable: {}'.format(key_var_name))
    subscription_key = os.environ['key_var_name']

    endpoint_var_name = 'endpoint_var_name'
    if not endpoint_var_name in os.environ:
        raise Exception('Please set/export the environment variable: {}'.format(endpoint_var_name))
    endpoint = os.environ['endpoint_var_name']

    # If you encounter any issues with the base_url or path, make sure
    # that you are using the latest endpoint: https://docs.microsoft.com/azure/cognitive-services/translator/reference/v3-0-translate
    path = '/translate?api-version=3.0'
    params = '&to=en'
    constructed_url = endpoint + path + params
```

```
X["full_description_en"] = ""
for index, row in X.iterrows():
    try:
        Translated = translateToEng(row["full_description"])
        row["full_description_en"] = Translated[0]['translations'][0]['text']
    except:
        print("error", index)
```

STEP 24

Storing the data in an excel file

```
X.to_csv('input_data_features.csv')
```

STEP 25

Reading from the excel file

```
df=pd.read_csv('/content/gdrive/MyDrive/input_data_features.csv')
```

```
df.head()
```

STEP 26

Looking at the info in the new dataframe

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8495 entries, 0 to 8494
Data columns (total 5 columns):
 #   Column               Non-Null Count  Dtype 
---  ------               --------------  ----- 
 0   Unnamed: 0           8495 non-null   int64 
 1   caller               8495 non-null   object
 2   assignment_group     8495 non-null   object
 3   full_description     8495 non-null   object
 4   full_description_en  8495 non-null   object
dtypes: int64(1), object(4)
memory usage: 332.0+ KB
```

STEP 27

Dropping the unnamed column

```
df = df.drop(['Unnamed: 0'], axis=1)
```

```
X=df
```

STEP 28

Separating the output column and the rest of the columns

```
Y = data['assignment_group']

X=X.drop(['assignment_group'], axis=1)
```

STEP 29

Checking for null values in the dataset .

```
data.isna().values.any()
```

```
False
```

```
data.isnull().values.any()
```

```
False
```

STEP 30

Post translation of long_description column to English. Dropping columns with errors

```
Y = Y.drop([2144, 4087, 4089, 4687, 7345])
```

```
Y.shape
```

```
(8495,)
```

```
X.shape
```

```
(8495, 3)
```

STEP 31

Checking for correlation between caller and assignment group

```
def correlation(x, y):
    confusion_matrix = pd.crosstab(x,y)
    chi2 = chi2_contingency(confusion_matrix)[0]
    n = confusion_matrix.sum().sum()
    phi2 = chi2/n
    r,k = confusion_matrix.shape
    phi2corr = max(0, phi2-((k-1)*(r-1))/(n-1))
    rcorr = r-((r-1)**2)/(n-1)
    kcorr = k-((k-1)**2)/(n-1)
    return np.sqrt(phi2corr/min((kcorr-1),(rcorr-1)))

correlation(X['caller'], Y)
```

```
0.11325931104332101
```

STEP 32

As the correlation between caller and the assignment group column is low, dropping the caller column

```
X = X.drop(['caller'], axis=1)
```

```
X_emb = X
```

```
X_emb
```

	full_description	full_description_en
0	login issue verified user detailsemployee man...	login issue verified user detailsemployee man...
1	outlook received from hmjdrvpbkomuaywngmailcom...	outlook received from hmjdrvpbkomuaywngmailcom...
2	cant log in to vpn received from eylqgodmybqkw...	cant log in to vpn received from eylqgodmybqkw...
3	unable to access hrtool page unable to access ...	unable to access hrtool page unable to access ...

STEP 33

Using Reg Exp Tokenizer for tokenization and printing the tokens

```
nltk.download('punkt')
tokenizer = nltk.tokenize.RegexpTokenizer(r'\w+')
X['full_description_en'] = X['full_description_en'].apply(tokenizer.tokenize)
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
```

```
X['full_description_en']
```

```
0       [login, issue, verified, user, detailsemployee...
1       [outlook, received, from, hmjdrvpbkomuaywngmai...
2       [cant, log, in, to, vpn, received, from, eylqg...
3       [unable, to, access, hrtool, page, unable, to,...
4                           [skype, error, skype, error]
```

STEP 34

Stemming and lemmatization the data

```
def lemmatize_tokens(tokens):
    lemmatizer = nltk.stem.WordNetLemmatizer()
    tokens = [lemmatizer.lemmatize(token) for token in tokens]
    return tokens

X['full_description_en'] = X['full_description_en'].apply(lemmatize_tokens)
```

STEP 35

Removing the STOPWORDS

```
from nltk.corpus import stopwords
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))

def remove_stopwords(descriptions):
    word_tokens = descriptions
    clean_tokens = [w for w in word_tokens if not w in stop_words]

    return clean_tokens
X['full_description_en'] = X['full_description_en'].apply(remove_stopwords)
```

STEP 36

Visualizing the data in the form a WORD CLOUD

```
from wordcloud import WordCloud
from PIL import Image

comment_words = ' '
for rows in X['full_description_en']:
    for words in rows:
        comment_words = comment_words + words + ' '
def plot_cloud(wordcloud):
    # Set figure size
    plt.figure(figsize=(40, 30))
    # Display image
    plt.imshow(wordcloud)
    # No axis details
    plt.axis("off");

#mask = np.array(Image.open('twitter_mask.png'))
# Generate wordcloud
wordcloud = WordCloud(width = 3000, height = 2000, random_state=1, background_color='white', colormap='Set2', collocations=False).generate(comment_words)
# Plot
plot_cloud(wordcloud)
```

STEP 37

Making the test train split

```
X_train1, X_val, y_train1, y_val = train_test_split(X, Y, test_size=0.3, random_state=3)
```

```
X_train, X_test, y_train, y_test = train_test_split(X_train1, y_train1, test_size=0.2, random_state=3)
```

```
print(X_train.shape,y_train.shape)
print(X_val.shape,y_val.shape)
print(X_test.shape,y_test.shape)
```

```
(4756, 2) (4756,)
(2549, 2) (2549,)
(1190, 2) (1190,)
```

STEP 38

Implementing TFID vectorizer

```
def vectorize():
    vectorizer = TfidfVectorizer(analyzer=lambda x: x)
    train_vectors = vectorizer.fit_transform(X_train['full_description_en'])
    val_vectors = vectorizer.fit_transform(X_val['full_description_en'])
    test_vectors = vectorizer.transform(X_test['full_description_en'])
    return train_vectors, val_vectors, test_vectors, vectorizer.vocabulary_

train_vectors, val_vectors, test_vectors, vocab = vectorize()
reversed_vocab = {i:word for word,i in vocab.items()}
```

STEP 39

Creating Glove Embedding and the a 200 dimension Glove embedding Matrix

```
max_features = 50000
maxlen = 250
embedding_size = 200
```

```
X_final = X_emb
```

STEP 40

Finding unique tokens

```
tokenizer = Tokenizer(num_words=max_features, filters='!"#$%&()*+,-./:;<=>?@[\]^_`{|}~', lower=True)
tokenizer.fit_on_texts(X_emb['full_description_en'].values)
word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))
```

```
Found 26287 unique tokens.
```

STEP 41

Printing the shape of the tensors

```
X = tokenizer.texts_to_sequences(X_emb['full_description_en'
X = pad_sequences(X, maxlen=maxlen)
print('Shape of data tensor:', X.shape)
```

```
Shape of data tensor: (8495, 250)
```

```
Y = pd.get_dummies(Y).values
print('Shape of label tensor:', Y.shape)
```

```
Shape of label tensor: (8495, 74)
```

STEP 42

Creating the embedding matrix with pretrained 200d wikipidiea based weights

```
EMBEDDING_FILE = 'glove.6B.200d.txt'

embeddings = {}
for o in open(EMBEDDING_FILE):
    word = o.split(" ")[0]
    # print(word)
    embd = o.split(" ")[1:]
    embd = np.asarray(embd, dtype='float32')
    # print(embd)
    embeddings[word] = embd

# create a weight matrix for words in training docs
embedding_matrix = np.zeros((num_words, 200))

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

STEP 43

Creating a sequential LSTM model

```
model = Sequential()

model.add(Embedding(26288, 200, weights=[embedding_matrix], input_length=X_train.shape[1]))

model.add(SpatialDropout1D(0.2))

model.add(LSTM(128, dropout=0.2, return_sequences=True))

model.add(SpatialDropout1D(0.2))

model.add(LSTM(100))

model.add(Flatten())

model.add(Dense(units=15, activation='relu'))
model.add(Dropout(0.1, input_shape=(60,)))
model.add(Dense(units=15, activation='relu'))
```

STEP 44

Executing the model summary

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 4, 200)            5257600
_________________________________________________________________
spatial_dropout1d_2 (Spatial (None, 4, 200)            0
_________________________________________________________________
lstm_2 (LSTM)                (None, 4, 128)            168448
_________________________________________________________________
spatial_dropout1d_3 (Spatial (None, 4, 128)            0
_________________________________________________________________
lstm_3 (LSTM)                (None, 100)               91600
_________________________________________________________________
flatten_1 (Flatten)          (None, 100)               0
_________________________________________________________________
dense_3 (Dense)              (None, 15)                1515
_________________________________________________________________
dropout_1 (Dropout)          (None, 15)                0
_________________________________________________________________
dense_4 (Dense)              (None, 15)                240
_________________________________________________________________
dense_5 (Dense)              (None, 74)                1184
=================================================================
Total params: 5,520,587
Trainable params: 5,520,587
Non-trainable params: 0
```

STEP 45

Using ADAM optimizer

```
optimizer = Adam(lr=1e-2, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False)
```

STEP 46

Using categorical cross entropy loss and early stopping

```
checkpoint = ModelCheckpoint("model-{loss:.2f}.h5", monitor="loss", verbose=1, save_best_only=True,
                             save_weights_only=True, mode="min")
stop = EarlyStopping(monitor="loss", patience=5, mode="min")
reduce_lr = ReduceLROnPlateau(monitor="loss", factor=0.2, patience=5, min_lr=1e-3, verbose=1, mode="min")
```

```
model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])
```

STEP 47

Fitting the model and getting the accuracy

```
history = model.fit(X_emb_train, Y_emb_train, epochs=5, batch_size=50, validation_data=(X_emb_val, Y_emb_val), callbacks=[checkpoint, stop, reduce_lr])
```

STEP 48

```
Epoch 1/5
WARNING:tensorflow:Model was constructed with shape (None, 4) for input KerasTensor(type_spec=TensorSpec(shape=(None, 4), dtype=tf.float32, name='en
WARNING:tensorflow:Model was constructed with shape (None, 4) for input KerasTensor(type_spec=TensorSpec(shape=(None, 4), dtype=tf.float32, name='en
96/96 [==============================] - ETA: 0s - loss: 2.9002 - accuracy: 0.4030WARNING:tensorflow:Model was constructed with shape (None, 4) for
96/96 [==============================] - 40s 393ms/step - loss: 2.8972 - accuracy: 0.4034 - val_loss: 2.2864 - val_accuracy: 0.4959

Epoch 00001: loss improved from inf to 2.61120, saving model to model-2.61.h5
Epoch 2/5
96/96 [==============================] - 35s 365ms/step - loss: 2.1986 - accuracy: 0.5266 - val_

Epoch 00002: loss improved from 2.61120 to 2.10092, saving model to model-2.10.h5
Epoch 3/5
96/96 [==============================] - 35s 360ms/step - loss: 1.8635 - accuracy: 0.5380 - val_

Epoch 00003: loss improved from 2.10092 to 1.85207, saving model to model-1.85.h5
Epoch 4/5
96/96 [==============================] - 35s 364ms/step - loss: 1.6861 - accuracy: 0.5709 - val_

Epoch 00004: loss improved from 1.85207 to 1.67692, saving model to model-1.68.h5
Epoch 5/5
96/96 [==============================] - 35s 363ms/step - loss: 1.5349 - accuracy: 0.5784 - val_
```


CONCLUSION

As seen using GLOVE Embedding the accuracy was 57 percent. We are trying ELMO and Word 2 Vec embeddings but the accuracy is less.

UPSAMPLING & DILBERT IMPLEMENTATION IN PYTORCH

```
#up-sampling all groups to have 1500 samples
from imblearn.pipeline import make_pipeline
from imblearn.over_sampling import ADASYN, SMOTE, RandomOverSampler
from sklearn.utils import resample
# Treat the imbalnce in the itTicketDF dataset by resampling to 1500.This is for us to try creating a single model which use the whole dataset & verify the performance
de_resampled_new = df_latest[0:0]

maxNo = 1500
for grp in df_latest['assignment_group'].unique():
    de_Grp = df_latest[df_latest['assignment_group'] == grp]
    resampled = resample(de_Grp, replace=True, n_samples=int(maxNo), random_state=123)
    de_resampled_new = de_resampled_new.append(resampled)

descending_order = de_resampled_new['assignment_group'].value_counts().sort_values(ascending=False).index
plt.subplots(figsize=(22,5))
#add code to rotate the labels
ax=sns.countplot(x='assignment_group', data=de_resampled_new)
ax.set_xticklabels(ax.get_xticklabels(), rotation=40, ha="right")
plt.tight_layout()
plt.show()
```

TOKENIZATION-DISTILBERT

```
from transformers import DistilBertTokenizer

tokenizer = DistilBertTokenizer.from_pretrained('distilbert-base-uncased', do_lower_case=True)
```

```
text_ids = list()
for sent in X:
    encoded_sent = tokenizer.encode(sent,
                                    add_special_tokens=True,
                                    max_length = 100)
    encoded_sent.extend([0]* (100 - len(encoded_sent)))

    text_ids.append(encoded_sent)

text_ids[0]
```

ATTENTION MASKS

```
text_ids_lengths = [len(text_ids[i]) for i in range(len(text_ids))]
print(min(text_ids_lengths))
print(max(text_ids_lengths))
```

```
100
100
```

```
att_masks = []
for ids in text_ids:
    masks = [int(id > 0) for id in ids]
    att_masks.append(masks)

att_masks[0]
```

CREATING TRAINING TEST VALIDATION DATA

```
train_x, test_val_x, train_y, test_val_y = train_test_split(text_ids, Y, random_state=111, test_size=0.3)
train_m, test_val_m = train_test_split(att_masks, random_state=111, test_size=0.3)

test_x, val_x, test_y, val_y = train_test_split(test_val_x, test_val_y, random_state=111, test_size=0.2)
test_m, val_m = train_test_split(test_val_m, random_state=111, test_size=0.2)
```

```
import torch

train_x = torch.tensor(train_x)
test_x = torch.tensor(test_x)
val_x = torch.tensor(val_x)
train_y = torch.tensor(train_y)
test_y = torch.tensor(test_y)
val_y = torch.tensor(val_y)
train_m = torch.tensor(train_m)
test_m = torch.tensor(test_m)
val_m = torch.tensor(val_m)
```

DEFINING THE MODEL

```
from transformers import DistilBertForSequenceClassification, AdamW, DistilBertConfig

num_labels = len(set(Y))

model = DistilBertForSequenceClassification.from_pretrained('distilbert-base-uncased', num_labels=num_labels,
                                                            output_attentions=False, output_hidden_states=False)
```

```
100%|██████████| 267967963/267967963 [13:41<00:00, 326100.67B/s]
```

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(device)

model = model.to(device)
```

FINE TUNING THE MODEL

```
train_losses = []
val_losses = []
num_mb_train = len(train_dataloader)
num_mb_val = len(val_dataloader)

if num_mb_val == 0:
    num_mb_val = 1

for n in range(num_epochs):
    train_loss = 0
    val_loss = 0
    start_time = time.time()

    for k, (mb_x, mb_m, mb_y) in enumerate(train_dataloader):
        optimizer.zero_grad()
        model.train()

        mb_x = mb_x.to(device)
        mb_m = mb_m.to(device)
        mb_y = mb_y.to(device)

        outputs = model(mb_x, attention_mask=mb_m, labels=mb_y)

        loss = outputs[0]
        #loss = model_loss(outputs[1], mb_y)
        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
        optimizer.step()
        scheduler.step()

        train_loss += loss.data / num_mb_train

    print ("\nTrain loss after itaration %i: %f" % (n+1, train_loss))
    train_losses.append(train_loss.cpu())
```

```
        optimizer.zero_grad()
        model.train()

        mb_x = mb_x.to(device)
        mb_m = mb_m.to(device)
        mb_y = mb_y.to(device)

        outputs = model(mb_x, attention_mask=mb_m, labels=mb_y)

        loss = outputs[0]
        #loss = model_loss(outputs[1], mb_y)
        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
        optimizer.step()
        scheduler.step()

        train_loss += loss.data / num_mb_train

    print ("\nTrain loss after itaration %i: %f" % (n+1, train_loss))
    train_losses.append(train_loss.cpu())

    with torch.no_grad():
        model.eval()

        for k, (mb_x, mb_m, mb_y) in enumerate(val_dataloader):
            mb_x = mb_x.to(device)
            mb_m = mb_m.to(device)
            mb_y = mb_y.to(device)

            outputs = model(mb_x, attention_mask=mb_m, labels=mb_y)

            loss = outputs[0]
            #loss = model_loss(outputs[1], mb_y)

            val_loss += loss.data / num_mb_val

        print ("Validation loss after itaration %i: %f" % (n+1, val_loss))
        val_losses.append(val_loss.cpu())
```

ACCURACY OF THE TRAIN DATA

```
outputs = check_accuracy(trainAcc_data, trainAcc_sampler, trainAcc_dataloader)
_, predicted_values = torch.max(outputs, 1)
predicted_values = predicted_values.numpy()
true_values = train_y.numpy()
```

```
train_accuracy = np.sum(predicted_values == true_values) / len(true_values)
print ("Train Accuracy:", train_accuracy)
```

ACCURACY OF THE VALIDATION DATA

```
valAcc_data = TensorDataset(val_x, val_m)
valAcc_sampler = SequentialSampler(valAcc_data)
valAcc_dataloader = DataLoader(valAcc_data, sampler=valAcc_sampler, batch_size=batch_size)
```

```
outputs = check_accuracy(valAcc_data, valAcc_sampler, valAcc_dataloader)
_, predicted_values = torch.max(outputs, 1)
predicted_values = predicted_values.numpy()
true_values = val_y.numpy()
```

ACCURACY OF THE TEST DATA

```
outputs = check_accuracy(test_data, test_sampler, test_dataloader)
_, predicted_values = torch.max(outputs, 1)
predicted_values = predicted_values.numpy()
true_values = test_y.numpy()
```

```
test_accuracy = np.sum(predicted_values == true_values) / len(true_values)
print ("Test Accuracy:", test_accuracy)
```

```
Test Accuracy: 0.9405722276003011
```

CLASSIFICATION REPORT

```
from sklearn.metrics import classification_report

print(classification_report(true_values, predicted_values, target_names=[str(l) for l in label_values]))
```

```
              precision    recall  f1-score   support

       GRP_0       0.96      0.90      0.93       970
       GRP_1       0.88      1.00      0.94       351
      GRP_10       0.98      0.86      0.92       340
```

CONFUSION MATRIX

```
from sklearn.metrics import classification_report, confusion_matrix

cm_test = confusion_matrix(true_values, predicted_values)

plt.figure(figsize=(24,24))
plot_confusion_matrix(cm_test, classes=label_values, title='Confusion Matrix - Test Dataset')
```

```
Confusion matrix, without normalization
[[872   0   3 ...   2   0   2]
 [  0 351   0 ...   0   0   0]
 [  0   0 292 ...   0   0  48]
 ...
 [  0   0   0 ... 336   0   0]
 [  2   2   2 ...   0 205 127]
 [  0   0   0 ...   0   0 362]]
```