AUTOMATED TICKET ASSIGNMENT PROJECT REPORT

ABSTRACT

Automated Ticket Assignment using Deep Learning Models

Project Team:

ANAND

DEVENAJAN

MAYANK

PRAMOD

RENJITH

CONTENTS

SUMMARY OF PROJECT

INTRODUCTION

One of the key activities of any IT function is to “Keep the lights on” to ensure there is no impact to the Business operations. IT leverages Incident Management process to achieve the above Objective.
Incident tickets reaching the IT Teams can be manually or automatically assigned to various groups after initial triaging by the Frontllne teams. Manually triaging and assigning to expert groups is prone to mistakes and errors. Wrong assignment may lead to longer times to resolve and noise in the system when the issues are not attended to within the SLAs.

The best alternative is to use 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.

Accurate Automated assignment also leads to higher Customer satisfaction as the ticket reaches the rightly skilled technical faster and without any risk of wrong assignments.

BUSINESS DOMAIN VALUE

Currently the incidents are created by various stakeholders (Business Users, IT Users and Monitoring Tools) within IT Service Management Tool and are assigned to Service Desk teams (L1 / L2 teams).

This team will review the incidents for right ticket categorization, priorities and then carry out initial diagnosis to see if they can resolve. Around ~54% of the incidents are resolved by L1 / L2 teams. This is reflected in the fact that “GRP_0” in the dataset has around 50% of the tickets.

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.

SOLUTION PARAMETERS:

- The Solution should be able to reduce the time and efforts L1 / L2 needs to spend time reviewing Standard Operating Procedures (SOPs) before assigning to Functional teams
- Eliminate the manual review of SOPs before ticket assignment). 15 min is being spent for SOP review for each incident, by the L1/L2 team.
- Save on the Minimum of ~1 FTE effort needed only for incident assignment to L3 teams
- Eliminate 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.

SOLUTION PROPOSED:

- Analyse the data which has the content of the tickets as well as the Group details which handled the issue.
- Build and train various Machine Learning Models to classify the tickets.

- Build a model which will accept the Incident Ticket data in the form of Text, pre-process it enable it to be fed to the model and predict the right Group (L3/Vendor) to whom the ticket should be routed
- One of the important tasks is to filter out the noise from the data provided.
- We need to pre-process the data as this data is collected from different sources and formats - it can be from emails or other similar sources. The emails may be sent by individuals to report new issues or for follow up. Some emails can also be auto generated from various tools which capture errors or monitoring situations. Typically, the data will contain the email header information along with the signature of the sender and disclaimers. The challenge is to remove all the data which does not directly deal with the issue reported or assistance sought.

OVERVIEW OF THE PROCESS OF REACHING THE SOLUTION

- Initial period the time was spent in understanding the data.
- Attempt was made to use all the information available.
- Care was taken not to eliminate any data to avoid losing valuable information/data points.
- Translation, Regex, Stopwords, Custom word replacement, etc was used to clean the data
- Multiple Machine Learning Models were run against this data. Word2Vec, TFID were fed to KNN, Random Forest & Decision tree Classifiers.
- After multiple iterations it was determined that Deep Learning Models using LSTM and Conv1D layers would give the best results.
- The Neural Network can learn the relationships between the various sentences and provide healthy accuracies. The basic limitation has been on the fact the Dataset is very small for Deep Learning Models. With 8500 records, and after clean-up of the data, we were able to get only ~10000 tokens to train the network.
- To finalize the model without too much computation requirements Pre-trained weights from Glove was used. This helped achieve F1-Score around 0.64, which validated the approach
- Later once the model structure was finalized attempt was made to train all the layers using the Glove Embedding Layers as Initial Weights. This boosted the accuracy to 0.70 using Two Dropout layers and L1 & L2 Regularization
- The model was saved and deployed as a standalone code (notebook). This allows the model to be deployed on production with a small footprint. The same can be set up as a webservice which can run on the Ticket decryption text and identify the Group to which the Ticket needs to be routed without any reference to SOP or manual review.

OVERVIEW OF THE FINAL PROCESS

- Initial period the time was spent in understanding the data.
- Attempt was made to use all the information available.
- Care was taken not to eliminate any data to avoid losing valuable information/data points.
- Translation, Regex, Stopwords, Custom word replacement, etc was used to clean the data
- Multiple Machine Learning Models were run against this data. Word2Vec, TFID were fed to KNN, Random Forest & Decision tree Classifiers.
- After multiple iterations it was determined that Deep Learning Models using LSTM and Conv1D layers would give the best results.

- To finalize the model without too much computation requirements Pre-trained weights from Glove was used. This helped achieve F1-Score around 0.64, which validated the approach
- Later once the model structure was finalized attempt was made to train all the layers using the Glove Embedding Layers as Initial Weights. This boosted the accuracy to 0.70 using Two Dropout layers and L1 & L2 Regularization
- The model was saved and deployed as a standalone code (notebook). This allows the model to be deployed on production with a small footprint. The same can be set up as a webservice which can run on the Ticket decryption text and identify the Group to which the Ticket needs to be routed without any reference to SOP or manual review.

DETAILS OF THE PROCESS FOR ARRIVING AT THE SOLUTION

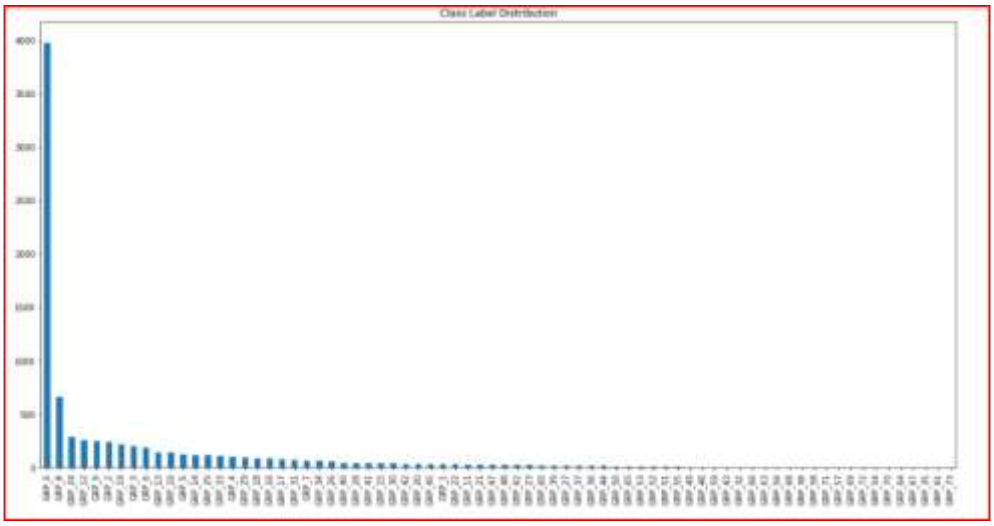
DATA PRE-PROCESSING

REVIEW OF THE DATA

- The xlsx file contained the following columns:
  - **Short description –** very short summary of the ticket
  - **Description –** Entire text of the issue reported
  - **Caller –** email of the Caller in first name last name format
  - **Assignment group –** the Group which was assigned the Ticket

IDENTIFICATION OF VARIOUS PATTERNS IN THE DATA

- We identified the null values in the Dataset.
- The data had entire emails pasted along with Disclaimer statements.
- Data was primarily in English but also had text from other languages.
- We saw that “Short Description” & “Description” contained the information of interest for us to help classify the text.
- The data had lots of names of the persons involved, signatures, server names, URLs, names of various tools, date and time of incidents, etc.
- Most of this information had to be cleaned up before it could be fed into the ML/Deep Learning models.
- The distribution of the Classes were very skewed. ~50% of the data belonged to GRP_0.



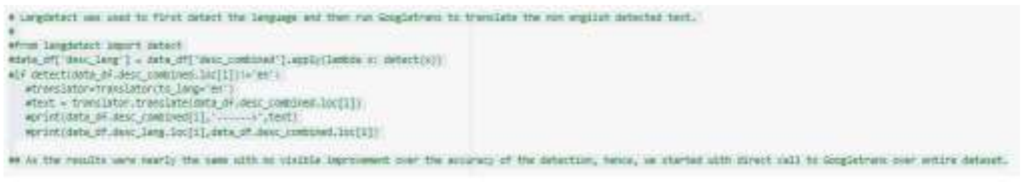
PROCESS OF DATA CLEANUP

- The columns (Short Description & Description) were seen to contain pertinent information for classification and hence we decided to combine them both
- After merging the above two columns we noticed that there were some Columns with non English characters and hence needed translation.

TRANSLATION

PROCESS

- Hence, we started with Translating the text. We tried native python libraries for translation but eventually settled on GoogleTrans. GoogleTrans was called for each row and the translated text captured in the "Translated" field. As this process is computationally heavy, we faced timeout issues. This led us to enhance the function to run for small sub-set of data without facing the time out issues. The translated data was stored into the filesystem to avoid running the function repeatedly.



```
# Langdetect was used to first detect the language and then run Googletrans to translate the non english detected text.
#
#from langdetect import detect
#data_df['desc_lang'] = data_df['desc_combined'].apply(lambda x: detect(x))
#if detect(data_df.desc_combined.loc[i])!='en':
   #translator=Translator(to_lang='en')
   #text = translator.translate(data_df.desc_combined.loc[i])
   #print(data_df.desc_combined[i],'------>',text)
   #print(data_df.desc_lang.loc[i],data_df.desc_combined.loc[i])

## As the results were nearly the same with no visible improvement over the accuracy of the detection, hence, we started with direct call to Googletrans over entire dataset.
```

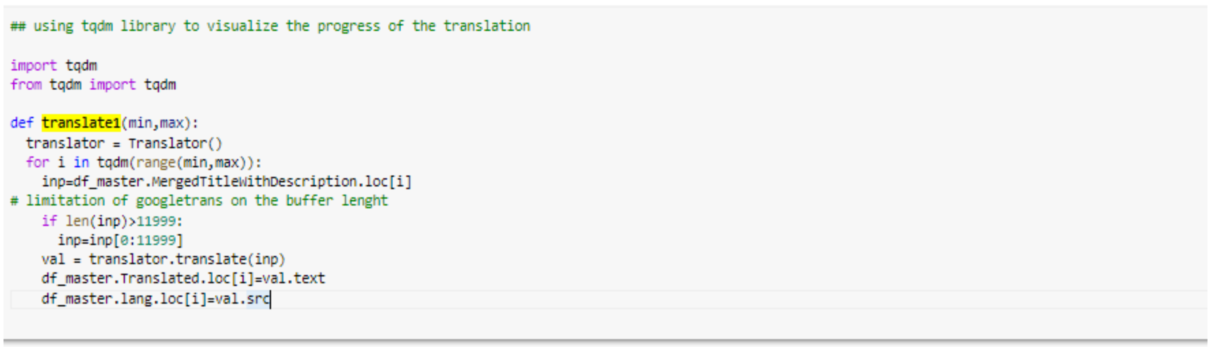
ISSUE IN TRANSLATIONS:

- After the first run of translation, we noticed that Text in Chinese language was not getting translated.
- We also noticed that some languages were wrongly detected
- On checking manually, we noticed that some text was rightly detected as belonging to a particular language but were getting translated. It was primarily because of some special characters in the text.

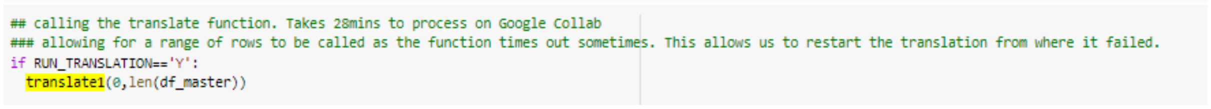
FIX APPLIED

- For such wrongly identified languages we reset them back to English which was the right language for the text
- It was noticed that there are 4 major languages in the dataset – English, German, Chinese & Portuguese.
- For Chinese language, we had to dump the data and run offline translation and load it back to the dataset
- Similarly, for German Text we had to do offline translation for some of the Records and load it back to the dataset.
- We also updated the Lang field that we had added to get some statistics around Languages in the Data for rows which were wrongly detected

Here is the Language distribution of the records



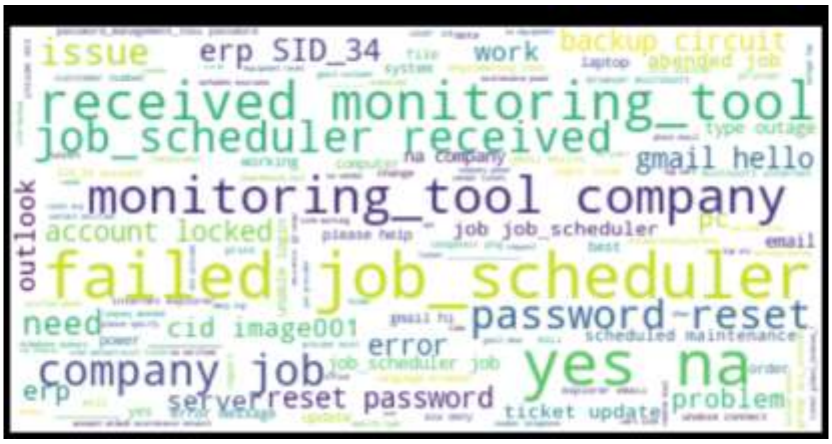
```
## using tqdm library to visualize the progress of the translation

import tqdm
from tqdm import tqdm

def translate1(min,max):
  translator = Translator()
  for i in tqdm(range(min,max)):
    inp=df_master.MergedTitleWithDescription.loc[i]
# limitation of googletrans on the buffer lenght
    if len(inp)>11999:
      inp=inp[0:11999]
    val = translator.translate(inp)
    df_master.Translated.loc[i]=val.text
    df_master.lang.loc[i]=val.src
```



```
## calling the translate function. Takes 28mins to process on Google Collab
### allowing for a range of rows to be called as the function times out sometimes. This allows us to restart the translation from where it failed.
if RUN_TRANSLATION=='Y':
  translate1(0,len(df_master))
```

Language	*% to Total*
English	95.5%
German	3.8%
Chinese	0.6%
Portuguese	0.2%

- After Translation this was the word cloud



REGEX

REMOVING OF SPECIAL CHARACTERS

- We wanted to eliminate some patterns of data while retaining some business related words which we believe will help in classifying the Ticket.
- To this end we studied the data and progressively added Regex expresssions to avoid ending up with orphaned words or bits and pieces of words which may not make much sense.

- Using Regex (ApplyRegEx function) we were able to eliminate email Id, URLs, Email signatures, Email header information, hostnames, instance ids, Disclaimer, etc.

CALLER NAMES IN THE TEXT

- It was noticed that that text contained names of the callers and other people.
- It was decided that we would remove that using the caller column. The Caller column was parsed to separate the first name and last name and the same was run on the combined Description text to eliminate them.
- text = [word for word in re.split(' ',text) if word not in callerlist]

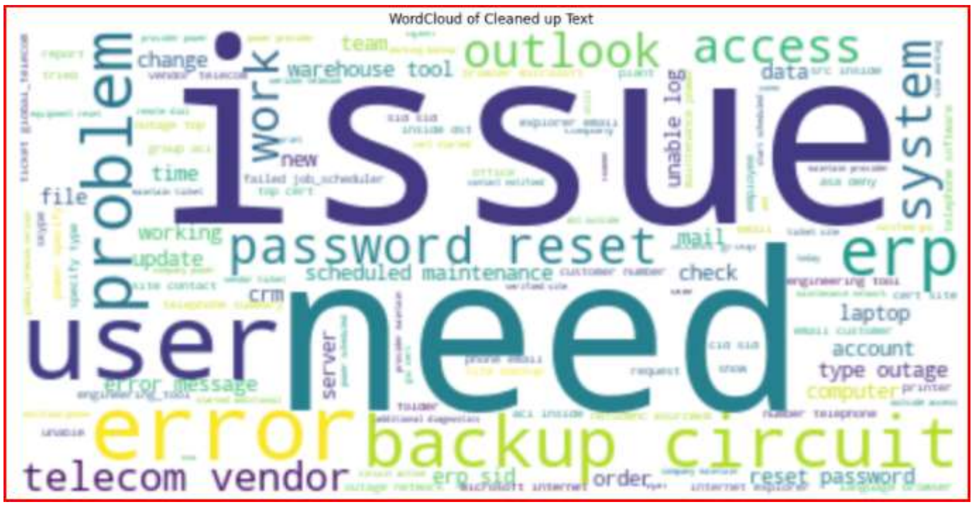
MISSPELLED WORDS

- It was noticed that there are a lot of misspelled words in the data.
- It was attempted to use Textblob to correct such words. However, the results were disappointing as the corrected words didn't bear any relationship to the rest of the text.
- Hence we used Wordnet corpus to identify non English words in the data. This was then exported to identify misspelt words and a replacement dictionary object was created & used

```
text = [replacementset[word] if word in replacementset else word for word in text.split(" ")]
```

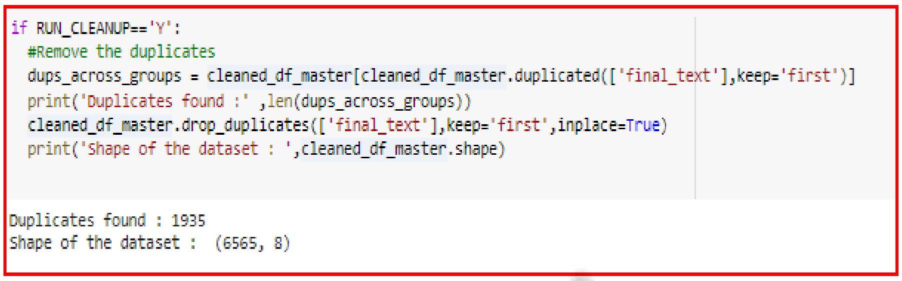
CUSTOM STOPWORDS:

- We also created a custom list of stopwords which was an enhancement on the standard nltk stopwords. We included a few words which were common in the data but not on the generic stopword list.
- text = [word for word in text.split(" ") if word not in custom_stopwords]
- Finally we used lemmatization to clean up the data
- text = [wn.lemmatize(word) for word in text]
- The word cloud was more indicative of the keywords used for Ticket assignments after the cleanup



DUPLICATION HANDLING

- Once the data was cleaned up there were quite a few duplicate records. There were same text across groups. Such duplicates end up reducing the accuracy of the classification. It was hence decided to eliminate the duplicates across groups/class.

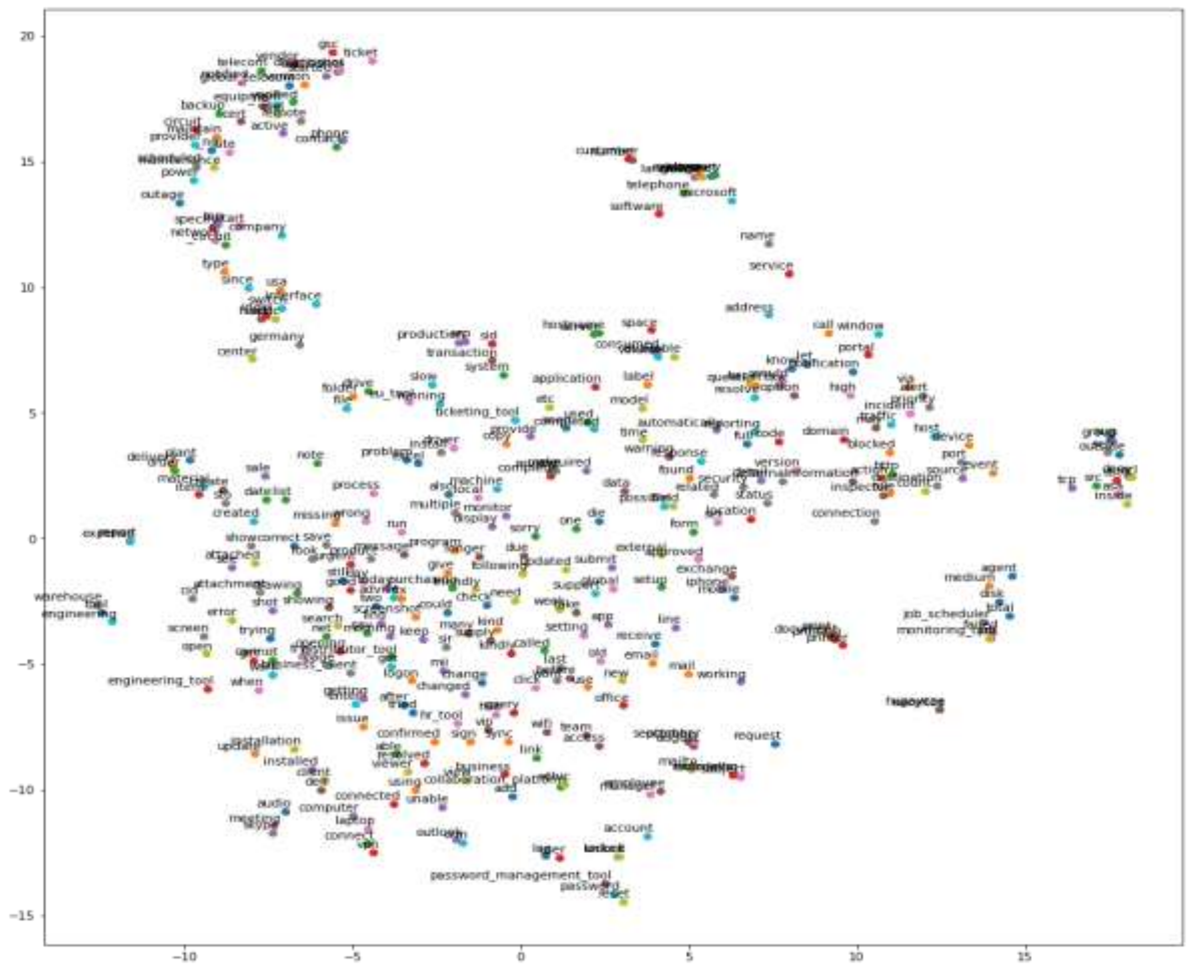


```
if RUN_CLEANUP=='Y':
  #Remove the duplicates
  dups_across_groups = cleaned_df_master[cleaned_df_master.duplicated(['final_text'],keep='first')]
  print('Duplicates found :' ,len(dups_across_groups))
  cleaned_df_master.drop_duplicates(['final_text'],keep='first',inplace=True)
  print('Shape of the dataset : ',cleaned_df_master.shape)

Duplicates found : 1935
Shape of the dataset :  (6565, 8)
```

- The Cleaned up dataframe was exported to filesystem to use for model building instead of having to run the computationally expensive clean up functions.

TSNE PLOT ON WORDVEC MODEL:

```
#### Plot of Words with minimum frequency of 75
model = Word2Vec(corpus, size=500, window=30, min_count=75, workers=4)
tsne_plot(model)
```



The visualization of tokens in 2 dimension using T-SNE helped identify the major clusters in the Dataset. This helped us take the decision to pare down the dataset to only 22 group eventually.

DATA REDUCTION, SELECTION OF DATA

- It was noticed that there were some Classes with only 1 record. There were 52 other Groups which had fewer than 50 records. It is not possible to use such sparse data to do any classification.
- The following process was used to identify the threshold for determining the appropriate number of groups to be used for training model:

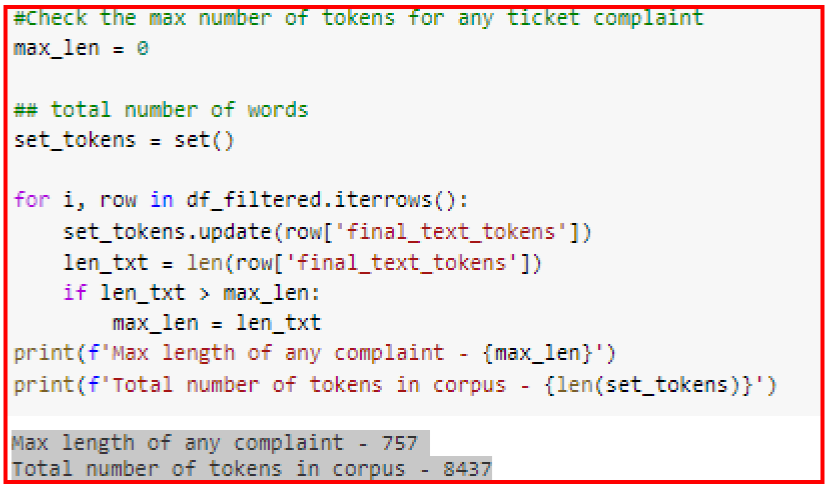
```
Threshold
================================
[1, 10, 25, 50, 75, 100]
Number of Records
================================
[6613, 6522, 6346, 5871, 5493, 4981]
Percentage Reduction
================================
[0, 1, 4, 11, 17, 25]
Number of Classes
================================
[74, 48, 37, 22, 17, 11]
```

- Restricting groups to have minimum number of samples to 50 gave us finally 22 classes for training our model. Data reduction was around 11% for the same.

MACHINE LEARNING MODELS

VECTORIZATION

- First the tokens were created. It was noticed that the maximum length of any issue text was 757 tokens while the total number of tokens were 8437.

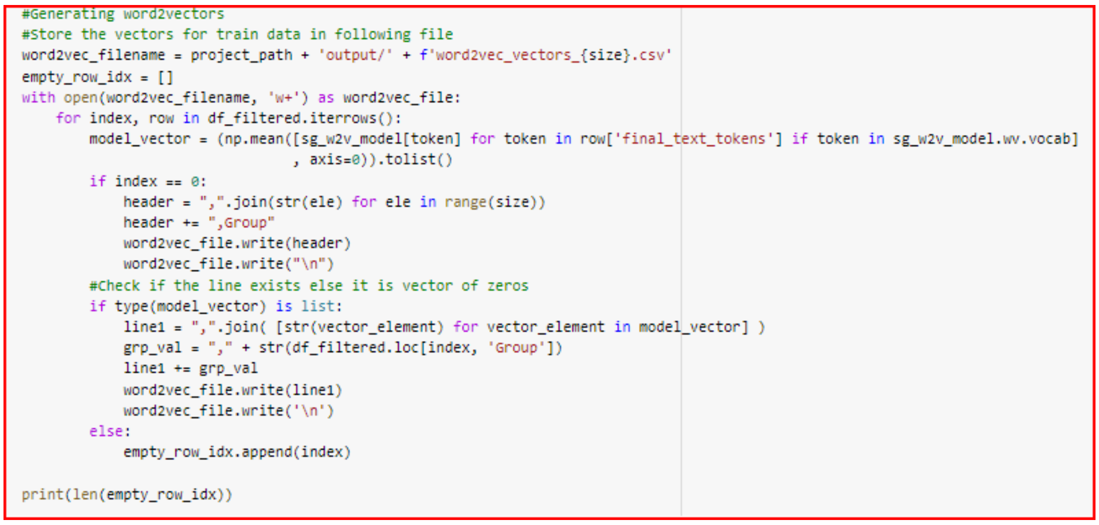


```python
#Check the max number of tokens for any ticket complaint
max_len = 0

## total number of words
set_tokens = set()

for i, row in df_filtered.iterrows():
    set_tokens.update(row['final_text_tokens'])
    len_txt = len(row['final_text_tokens'])
    if len_txt > max_len:
        max_len = len_txt
print(f'Max length of any complaint - {max_len}')
print(f'Total number of tokens in corpus - {len(set_tokens)}')
```

```
Max length of any complaint - 757
Total number of tokens in corpus - 8437
```

Gensim WORD2VEC MODEL

- Gensim Word2Vec was used to generate word2vec vectors
- Word2vec vectors of dimension 300 using skip gram model was created.



```python
#Generating word2vectors
#Store the vectors for train data in following file
word2vec_filename = project_path + 'output/' + f'word2vec_vectors_{size}.csv'
empty_row_idx = []
with open(word2vec_filename, 'w+') as word2vec_file:
    for index, row in df_filtered.iterrows():
        model_vector = (np.mean([sg_w2v_model[token] for token in row['final_text_tokens'] if token in sg_w2v_model.wv.vocab]
                                , axis=0)).tolist()
        if index == 0:
            header = ",".join(str(ele) for ele in range(size))
            header += ",Group"
            word2vec_file.write(header)
            word2vec_file.write("\n")
        #Check if the line exists else it is vector of zeros
        if type(model_vector) is list:
            line1 = ",".join( [str(vector_element) for vector_element in model_vector] )
            grp_val = "," + str(df_filtered.loc[index, 'Group'])
            line1 += grp_val
            word2vec_file.write(line1)
            word2vec_file.write('\n')
        else:
            empty_row_idx.append(index)

print(len(empty_row_idx))
```

- A dataframe was created to accumulate the Results of the Various Models that we wanted to run on the data.
- Using various Classifiers the following models were built.

CLASSIFIERS

***Classifier** Model*
Word2Vec + Decision Tree
Word2Vec + Random Forest
Word2Vec + KNN

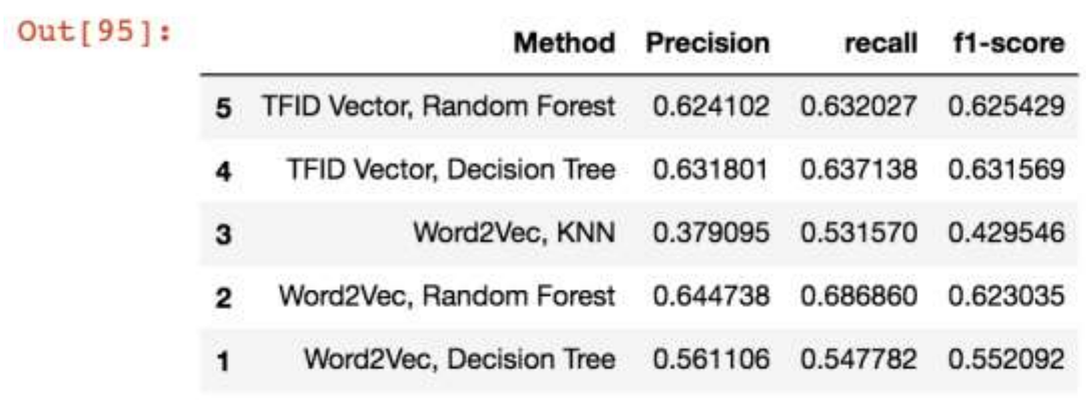
TFID Vector + Decision Tree

TFID Vector + Random Forest

- Here are the comparative accuracies of the various Models

RESULTS OF ML MODELS

Various ML models were attempted using the vectors generated from both the approaches ..



Out[95]:

	Method	Precision	recall	f1-score
5	TFID Vector, Random Forest	0.624102	0.632027	0.625429
4	TFID Vector, Decision Tree	0.631801	0.637138	0.631569
3	Word2Vec, KNN	0.379095	0.531570	0.429546
2	Word2Vec, Random Forest	0.644738	0.686860	0.623035
1	Word2Vec, Decision Tree	0.561106	0.547782	0.552092

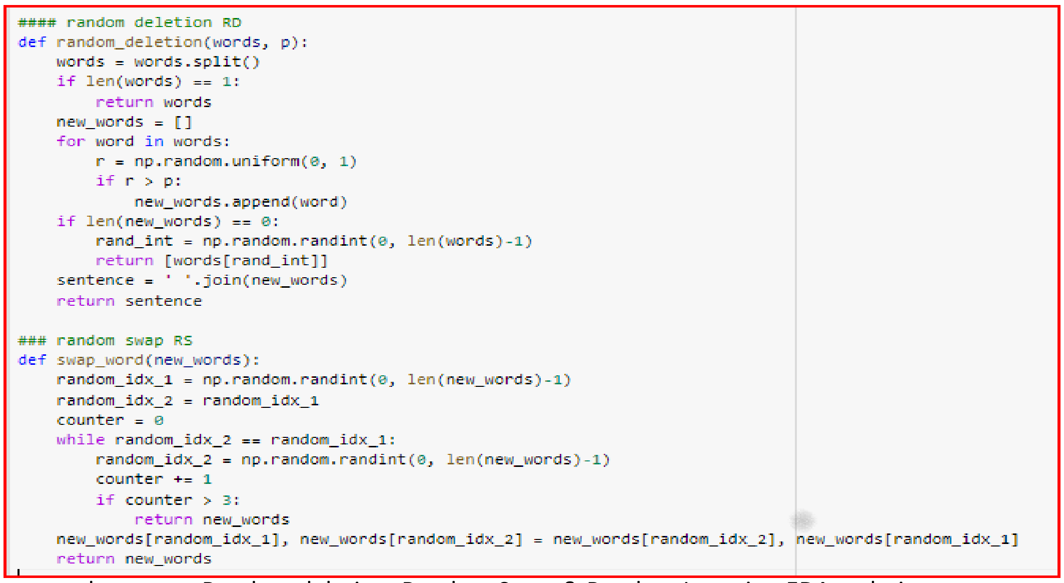
CONCLUSIONS FOR MACHINE LEARNING METHODS

- RandomForestModels, whether it is word2vec or tfid generated the better precision and recall scores. RandomForest word2vec generated pretty high recall scores, around 68%.
- F1-scores with both tfid and word2vec remained around 62 %.
- However, looking at the fact that we are looking at Ticket classification problem and a model that would need to continually feed more data over a period of time, it was decided to evaluate Deep Learning Models too.

DATA AUGMENTATION

- For the Deep Learning Models it was decided that we would need to augment the data. To do this we experimented with SMOTE & RandomOversampling methods. Both methods didn’t boost the recall scores though the training accuracy went up.
- At this point it was decided to work with “Easy Data Augmentation” algorithms
- As the Target class distribution is very imbalanced, it was decided to Augment the Data to ensure better balance between classes. The Class '0' was nearly 50% of the dataset.

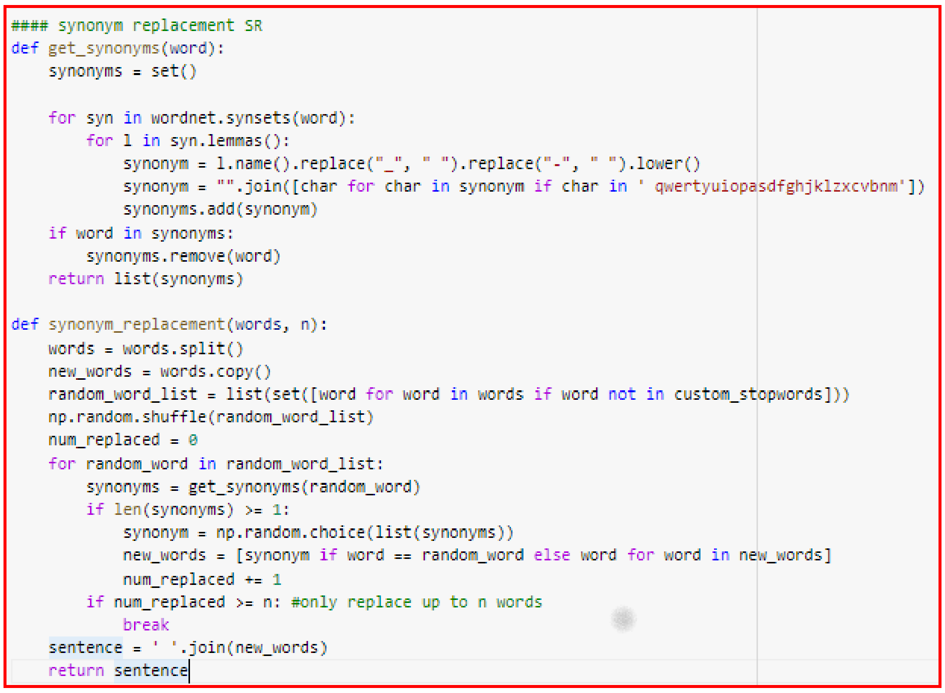
- Data split into train and test to ensure that we only create synthetic data for training set and not for testing (held out set). This ensures that data does not leak into the testing test thereby artificially boosting the model accuracies.



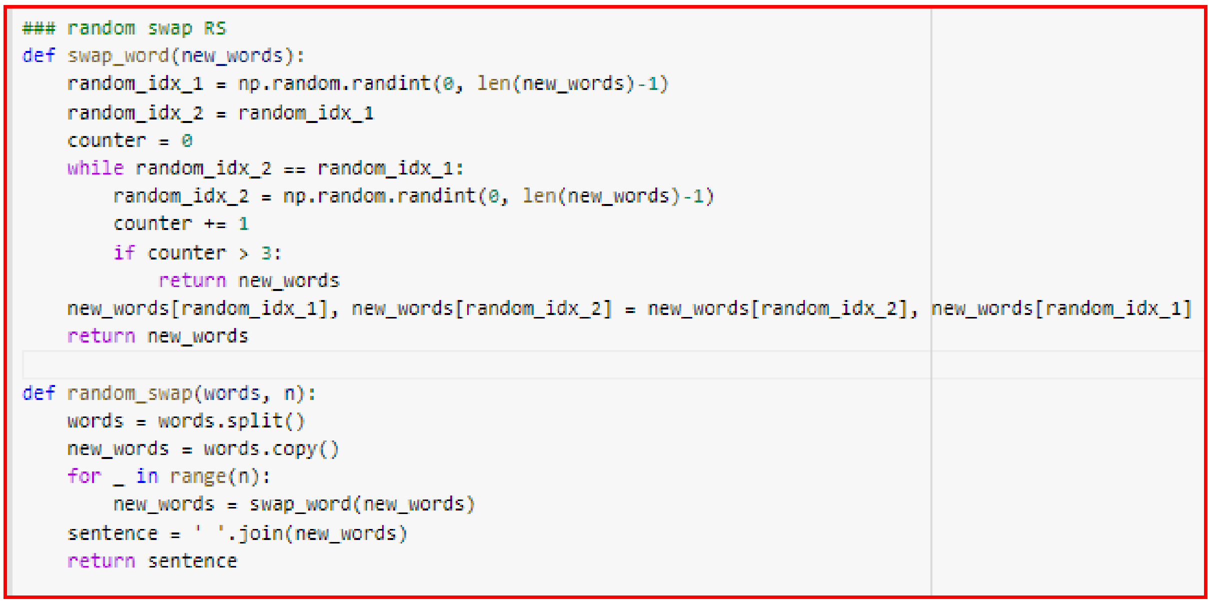
```python
#### random deletion RD
def random_deletion(words, p):
    words = words.split()
    if len(words) == 1:
        return words
    new_words = []
    for word in words:
        r = np.random.uniform(0, 1)
        if r > p:
            new_words.append(word)
    if len(new_words) == 0:
        rand_int = np.random.randint(0, len(words)-1)
        return [words[rand_int]]
    sentence = ' '.join(new_words)
    return sentence

### random swap RS
def swap_word(new_words):
    random_idx_1 = np.random.randint(0, len(new_words)-1)
    random_idx_2 = random_idx_1
    counter = 0
    while random_idx_2 == random_idx_1:
        random_idx_2 = np.random.randint(0, len(new_words)-1)
        counter += 1
        if counter > 3:
            return new_words
    new_words[random_idx_1], new_words[random_idx_2] = new_words[random_idx_2], new_words[random_idx_1]
    return new_words
```

- ○ We are using Synonym replacement, Random deletion, Random Swap & Random Insertion EDA techniques.

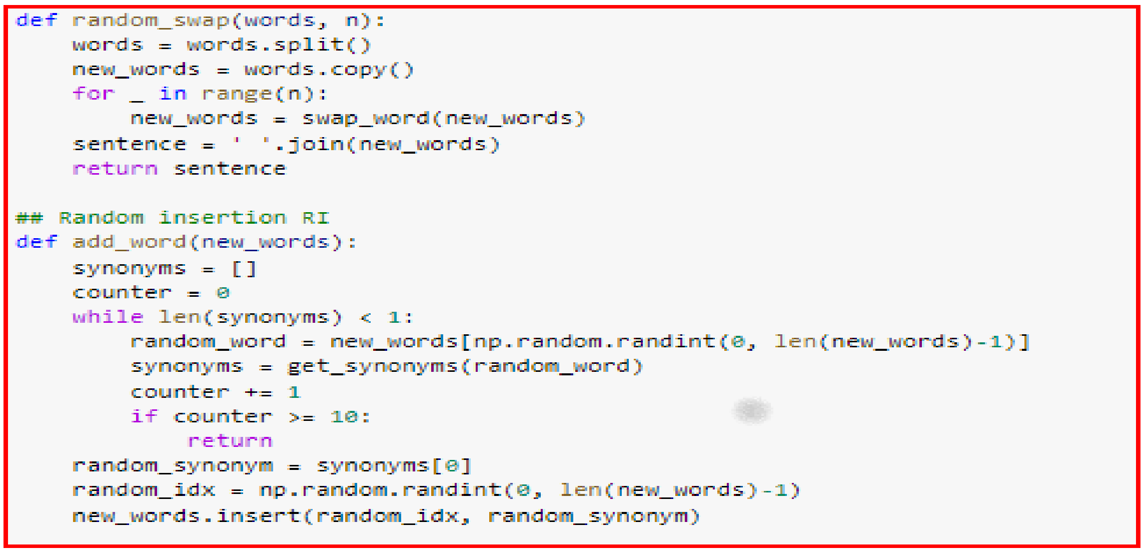
The EDA code by jasonwei20 [(Wei, 2019)] was used for this purpose.



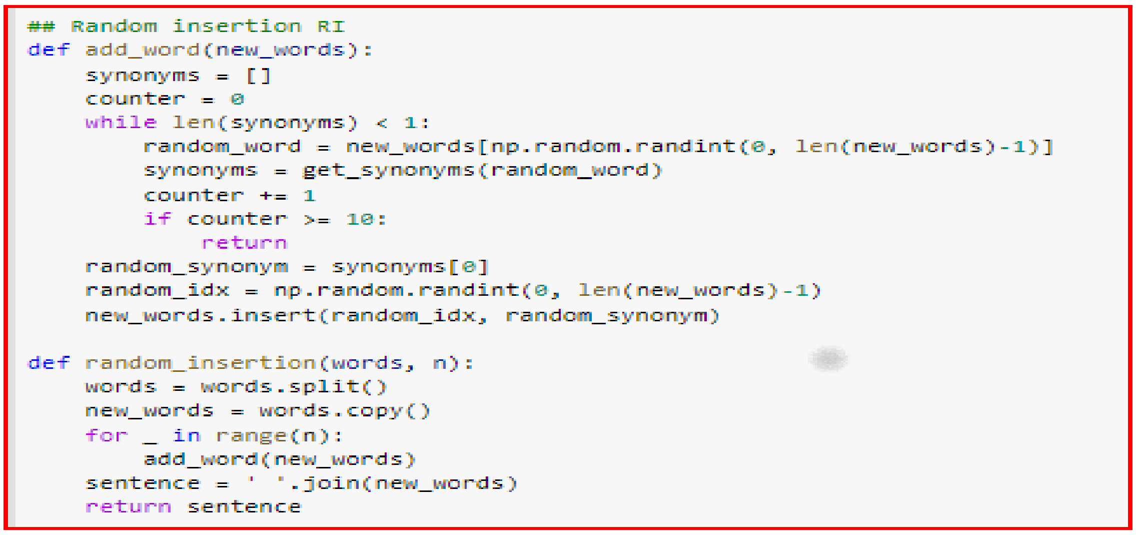
```python
#### synonym replacement SR
def get_synonyms(word):
    synonyms = set()

    for syn in wordnet.synsets(word):
        for l in syn.lemmas():
            synonym = l.name().replace("_", " ").replace("-", " ").lower()
            synonym = "".join([char for char in synonym if char in ' qwertyuiopasdfghjklzxcvbnm'])
            synonyms.add(synonym)
    if word in synonyms:
        synonyms.remove(word)
    return list(synonyms)

def synonym_replacement(words, n):
    words = words.split()
    new_words = words.copy()
    random_word_list = list(set([word for word in words if word not in custom_stopwords]))
    np.random.shuffle(random_word_list)
    num_replaced = 0
    for random_word in random_word_list:
        synonyms = get_synonyms(random_word)
        if len(synonyms) >= 1:
            synonym = np.random.choice(list(synonyms))
            new_words = [synonym if word == random_word else word for word in new_words]
            num_replaced += 1
        if num_replaced >= n: #only replace up to n words
            break
    sentence = ' '.join(new_words)
    return sentence
```



```
### random swap RS
def swap_word(new_words):
    random_idx_1 = np.random.randint(0, len(new_words)-1)
    random_idx_2 = random_idx_1
    counter = 0
    while random_idx_2 == random_idx_1:
        random_idx_2 = np.random.randint(0, len(new_words)-1)
        counter += 1
        if counter > 3:
            return new_words
    new_words[random_idx_1], new_words[random_idx_2] = new_words[random_idx_2], new_words[random_idx_1]
    return new_words

def random_swap(words, n):
    words = words.split()
    new_words = words.copy()
    for _ in range(n):
        new_words = swap_word(new_words)
    sentence = ' '.join(new_words)
    return sentence
```

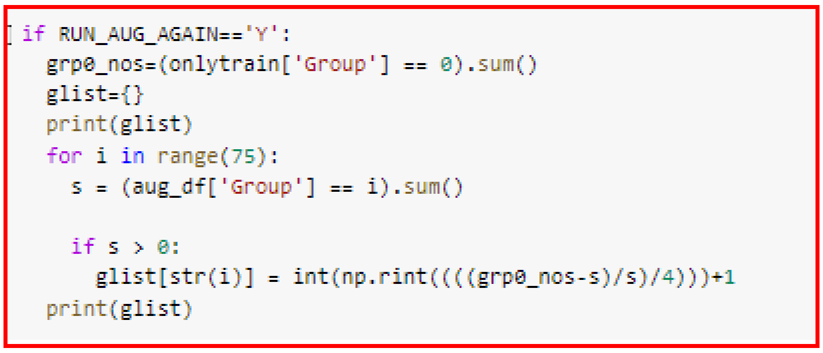


```
def random_swap(words, n):
    words = words.split()
    new_words = words.copy()
    for _ in range(n):
        new_words = swap_word(new_words)
    sentence = ' '.join(new_words)
    return sentence

## Random insertion RI
def add_word(new_words):
    synonyms = []
    counter = 0
    while len(synonyms) < 1:
        random_word = new_words[np.random.randint(0, len(new_words)-1)]
        synonyms = get_synonyms(random_word)
        counter += 1
        if counter >= 10:
            return
    random_synonym = synonyms[0]
    random_idx = np.random.randint(0, len(new_words)-1)
    new_words.insert(random_idx, random_synonym)
```



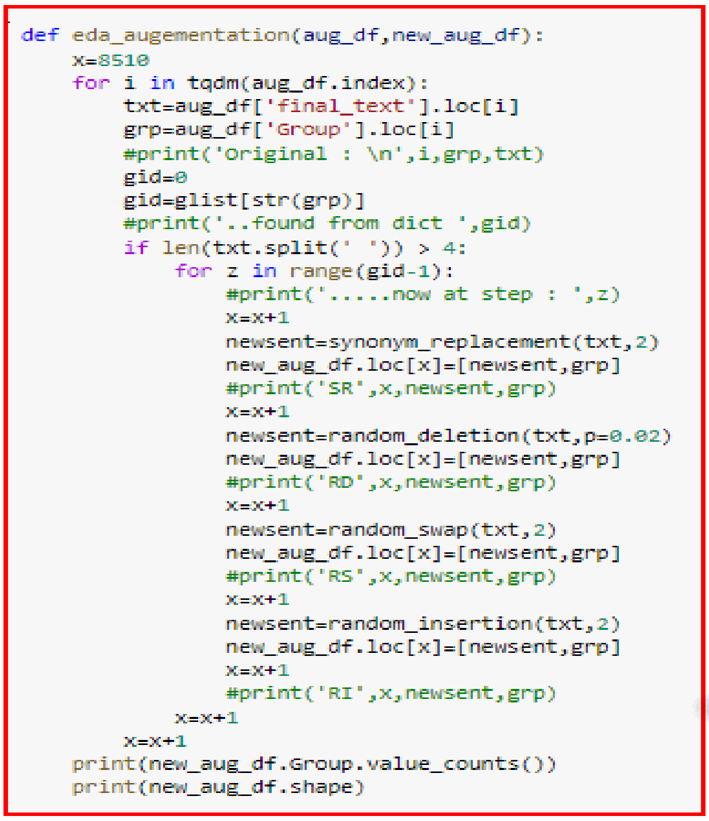
```
## Random insertion RI
def add_word(new_words):
    synonyms = []
    counter = 0
    while len(synonyms) < 1:
        random_word = new_words[np.random.randint(0, len(new_words)-1)]
        synonyms = get_synonyms(random_word)
        counter += 1
        if counter >= 10:
            return
    random_synonym = synonyms[0]
    random_idx = np.random.randint(0, len(new_words)-1)
    new_words.insert(random_idx, random_synonym)

def random_insertion(words, n):
    words = words.split()
    new_words = words.copy()
    for _ in range(n):
        add_word(new_words)
    sentence = ' '.join(new_words)
    return sentence
```

- The minority classes were identified and then boosted to close to the Majority Class (GRP_0). This was applied on the pared down dataset with 22 Classes.
- Decision was taken to first split the Train Test datasets and then only augment the Training Data set to avoid “Data Leaking” into the Test Set. 15% of the Dataset was “held out” and the same used for evaluation of the Model.
- The process used to Augment the data is as follows:
- First the amount of Augmentation to be done was determined by comparing the GRP_0 with the other groups.



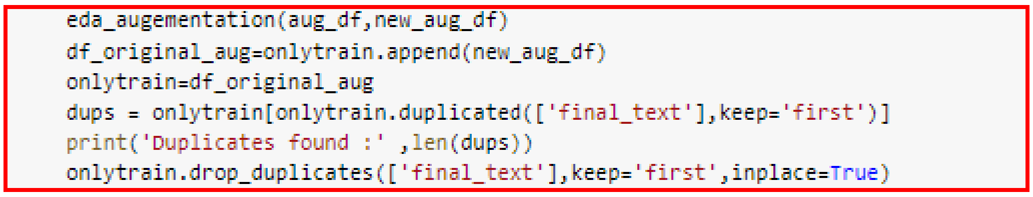
```
if RUN_AUG_AGAIN=='Y':
  grp0_nos=(onlytrain['Group'] == 0).sum()
  glist={}
  print(glist)
  for i in range(75):
    s = (aug_df['Group'] == i).sum()

    if s > 0:
      glist[str(i)] = int(np.rint((((grp0_nos-s)/s)/4)))+1
  print(glist)
```

- (No. of rows of GRP_0 – No. of Rows of specific minority class)/No. of Rows of Specific Minority Class / 4 (this is because for every record we created 4 more augmented records. One each of - Synonym replacement, Random insert, Random Delete, Random swap)



```
def eda_augementation(aug_df,new_aug_df):
    x=8510
    for i in tqdm(aug_df.index):
        txt=aug_df['final_text'].loc[i]
        grp=aug_df['Group'].loc[i]
        #print('Original : \n',i,grp,txt)
        gid=0
        gid=glist[str(grp)]
        #print('..found from dict ',gid)
        if len(txt.split(' ')) > 4:
            for z in range(gid-1):
                #print('.....now at step : ',z)
                x=x+1
                newsent=synonym_replacement(txt,2)
                new_aug_df.loc[x]=[newsent,grp]
                #print('SR',x,newsent,grp)
                x=x+1
                newsent=random_deletion(txt,p=0.02)
                new_aug_df.loc[x]=[newsent,grp]
                #print('RD',x,newsent,grp)
                x=x+1
                newsent=random_swap(txt,2)
                new_aug_df.loc[x]=[newsent,grp]
                #print('RS',x,newsent,grp)
                x=x+1
                newsent=random_insertion(txt,2)
                new_aug_df.loc[x]=[newsent,grp]
                x=x+1
                #print('RI',x,newsent,grp)
            x=x+1
        x=x+1
    print(new_aug_df.Group.value_counts())
    print(new_aug_df.shape)
```

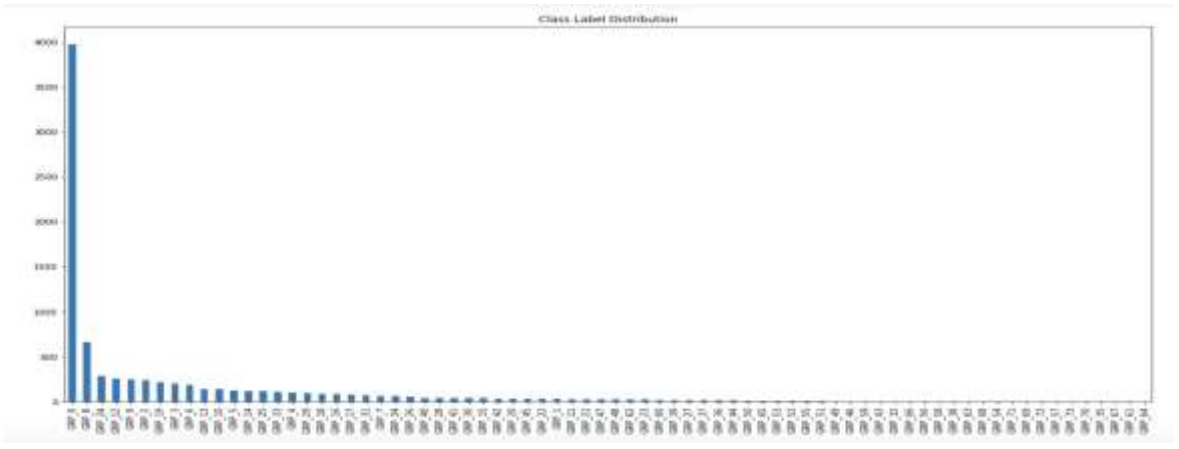
- (No. of rows of GRP_0 – No. of Rows of specific minority class)/No. of Rows of Specific Minority Class / 4 (this is because for every record we created 4 more augmented records. One each of - Synonym replacement, Random insert, Random Delete, Random swap)
- A dictionary object showing the multiplier to be applied to each Class was determined and stored in the dictionary object "glist". This was used to find out how many records for each group needs to be generated.
- This ensured that we had a better balance. One drawback of this method is that a lot of duplicates are created in the process and had to be deleted. Care was taken to keep the original record and delete the synthetic duplicates.



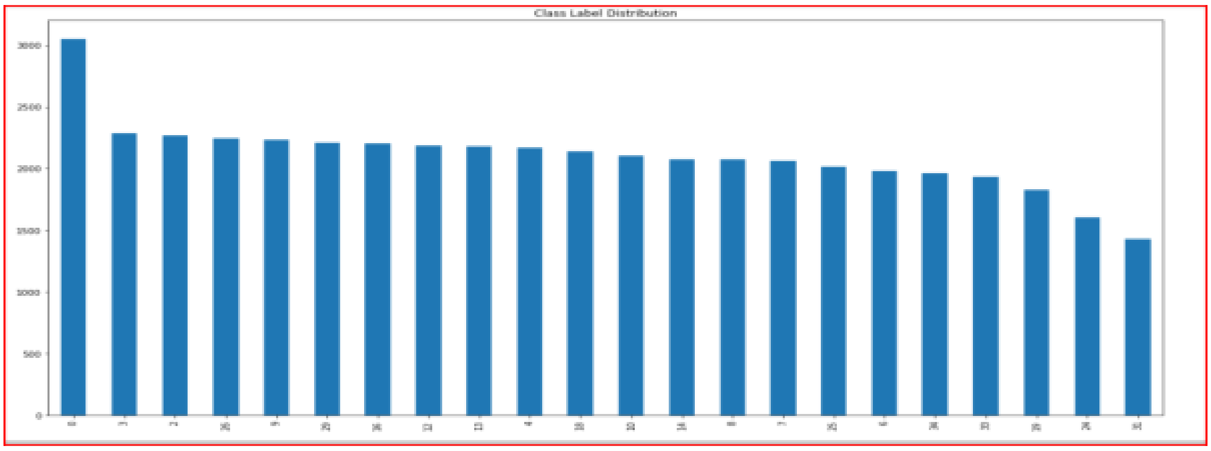
```
eda_augementation(aug_df,new_aug_df)
df_original_aug=onlytrain.append(new_aug_df)
onlytrain=df_original_aug
dups = onlytrain[onlytrain.duplicated(['final_text'],keep='first')]
print('Duplicates found :' ,len(dups))
onlytrain.drop_duplicates(['final_text'],keep='first',inplace=True)
```

RESULTS OF AUGMENTATION

INITIAL CLASS DISTRIBUTION:



CLASS DISTRIBUTION AFTER EDA

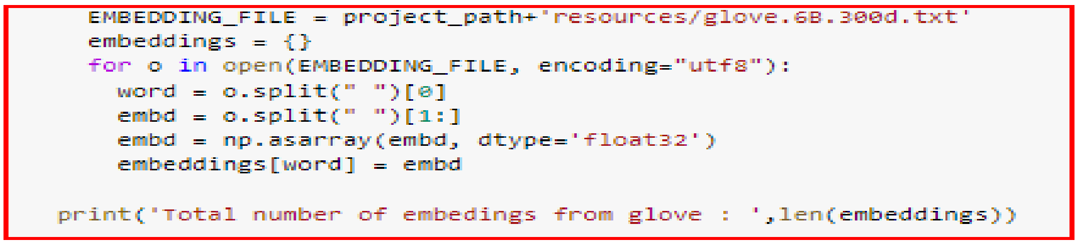


COMPARISON TO THE ORIGINAL DATA

Only Training data augmented			
Class	**Original**	**Augmented**	**% of Increase**
0	2592	2592	0%
	202	2412	1094%
3	167	2560	1433%
4	79	2528	3100%
6	61	2360	3769%
7	56	2464	4300%
8	259	2056	694%
9	66	2600	3839%
10	68	2412	3447%
12	201	2388	1088%
13	120	2380	1883%
14	99	2352	2276%
16	72	2556	3450%
18	71	2448	3348%
19	179	2004	1020%
24	193	1836	851%
25	99	2304	2227%
26	47	2632	5500%
29	81	2560	3060%
31	52	1872	3500%
33	91	2268	2392%
34	50	2392	4684%
Total	**4905**	**51976**	

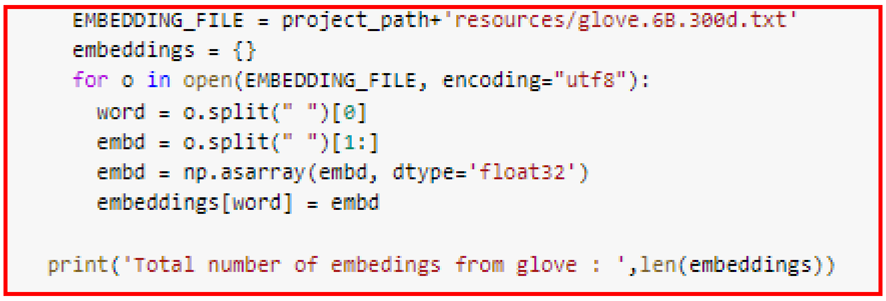
PRE-TRAINED MODELS

- It was decided to go for pre-trained models as training the model from a scratch with this less data didn't see logical.
- Glove 300 dimension embedding vector with 400000 words was used for the purpose.



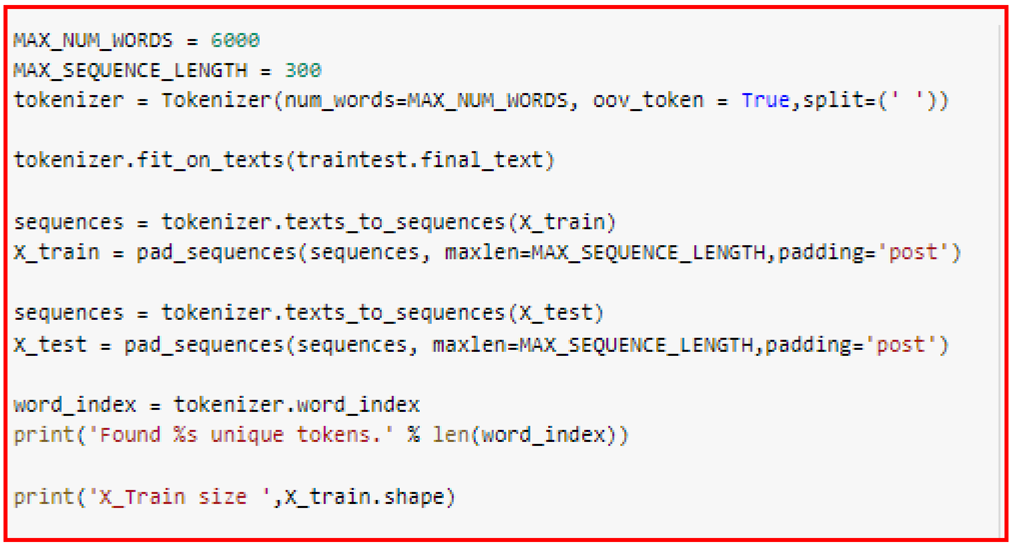
```
EMBEDDING_FILE = project_path+'resources/glove.6B.300d.txt'
embeddings = {}
for o in open(EMBEDDING_FILE, encoding="utf8"):
  word = o.split(" ")[0]
  embd = o.split(" ")[1:]
  embd = np.asarray(embd, dtype='float32')
  embeddings[word] = embd

print('Total number of embedings from glove : ',len(embeddings))
```

- LSTM based Deep Learning Model was used to train the final model on the augmented data.
- The model used the Keras Embedding layer as the first input layer.
- Glove 300 dimension embedding vectors was used for the purpose.



```
EMBEDDING_FILE = project_path+'resources/glove.6B.300d.txt'
embeddings = {}
for o in open(EMBEDDING_FILE, encoding="utf8"):
  word = o.split(" ")[0]
  embd = o.split(" ")[1:]
  embd = np.asarray(embd, dtype='float32')
  embeddings[word] = embd

print('Total number of embedings from glove : ',len(embeddings))
```

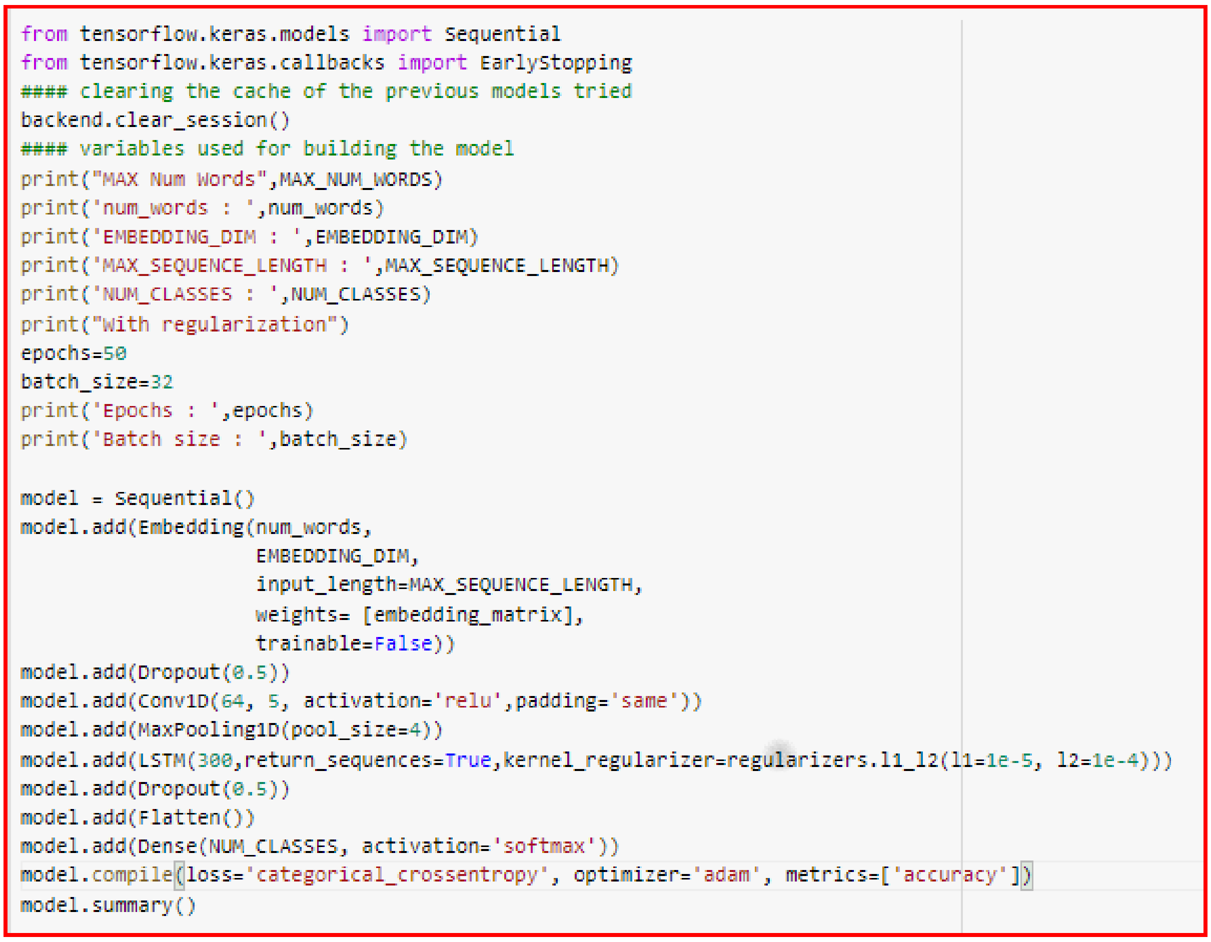
- Keras preprocessing Tokenizer was used to calculate the weights.



```
MAX_NUM_WORDS = 6000
MAX_SEQUENCE_LENGTH = 300
tokenizer = Tokenizer(num_words=MAX_NUM_WORDS, oov_token = True,split=(' '))

tokenizer.fit_on_texts(traintest.final_text)

sequences = tokenizer.texts_to_sequences(X_train)
X_train = pad_sequences(sequences, maxlen=MAX_SEQUENCE_LENGTH,padding='post')

sequences = tokenizer.texts_to_sequences(X_test)
X_test = pad_sequences(sequences, maxlen=MAX_SEQUENCE_LENGTH,padding='post')

word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))

print('X_Train size ',X_train.shape)
```

MODEL DESIGN:

- Sentence length of 300 tokens was used.
- Classification was done for 22 classes
- L1 & L2 Regularization was used.
- Experimented with Dropout after Embedding and LSTM layers was experimented and found to be yielding better results.
- Adam optimizer gave the best results.



```
from tensorflow.keras.models import Sequential
from tensorflow.keras.callbacks import EarlyStopping
#### clearing the cache of the previous models tried
backend.clear_session()
#### variables used for building the model
print("MAX Num Words",MAX_NUM_WORDS)
print('num_words : ',num_words)
print('EMBEDDING_DIM : ',EMBEDDING_DIM)
print('MAX_SEQUENCE_LENGTH : ',MAX_SEQUENCE_LENGTH)
print('NUM_CLASSES : ',NUM_CLASSES)
print("With regularization")
epochs=50
batch_size=32
print('Epochs : ',epochs)
print('Batch size : ',batch_size)

model = Sequential()
model.add(Embedding(num_words,
                    EMBEDDING_DIM,
                    input_length=MAX_SEQUENCE_LENGTH,
                    weights= [embedding_matrix],
                    trainable=False))
model.add(Dropout(0.5))
model.add(Conv1D(64, 5, activation='relu',padding='same'))
model.add(MaxPooling1D(pool_size=4))
model.add(LSTM(300,return_sequences=True,kernel_regularizer=regularizers.l1_l2(l1=1e-5, l2=1e-4)))
model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(NUM_CLASSES, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()
```

OPTIMIZER ANALYSIS

Data used	Input Dimension	Optimizer	Accuracy	Validation Accuracy	Eval Accuracy	F1-Score	Comments
Augmented	**6000**	**Adam**	**0.966**	**0.800**	**0.651**	**0.666**	**Best Results**
Augmented	6000	RMSPROP	0.966	0.800	0.651	0.653	
Augmented	6000	SGD	0.921	0.756	0.622	0.637	

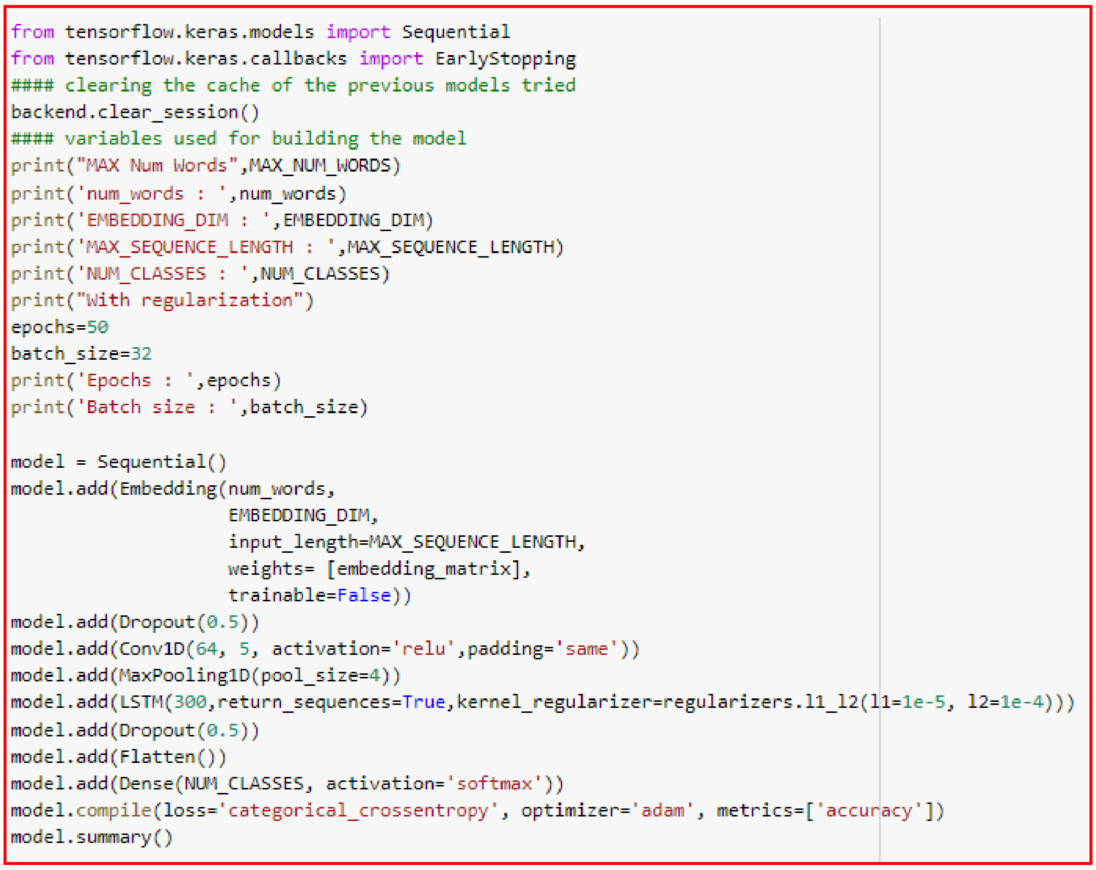
- As Adam optimizer was giving the best results, Adam optimizer was used further to tune the model.
- Further, experiments were conducted to find out the optimal number of Vectors to feed to the Network to identify the best model:

Experiments with various sizes of Input Dimension for optimal number of embedding to feed to the Network.

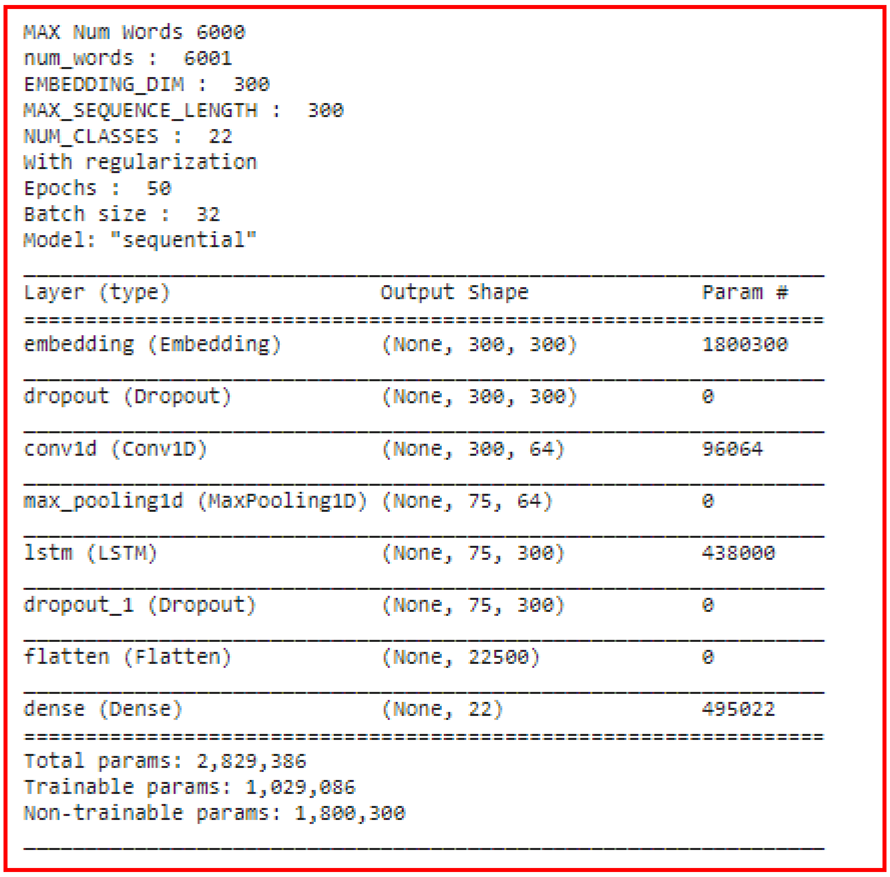
We were using Glove Embedding and only training after the 1st Layer. The results were as follows:

Data used	**Input Dimension**	**Optimizer**	**Accuracy**	**Validation Accuracy**	**Eval Accuracy**	**F1-Score**	**Comments**
Non Augmented	**5000**	**Adam**	**0.960**	**0.644**	**0.640**	**0.610**	**Baseline**
Augmented	1000	Adam	0.974	0.840	0.587	0.610	
Augmented	2000	Adam	0.979	0.813	0.641	0.644	
Augmented	3000	Adam	0.981	0.831	0.642	0.647	
Augmented	4000	Adam	0.982	0.829	0.623	0.635	
Augmented	5000	Adam	0.982	0.796	0.641	0.648	
Augmented	**6000**	**Adam**	**0.966**	**0.800**	**0.651**	**0.666**	**Best Model**
Augmented	7000	Adam	0.982	0.839	0.632	0.647	
Augmented	8000	Adam	0.981	0.832	0.636	0.645	
Augmented	9000	Adam	0.982	0.799	0.637	0.646	
Augmented	10000	Adam	0.956	0.780	0.641	0.647	
Augmented	11000	Adam	0.972	0.848	0.620	0.634	

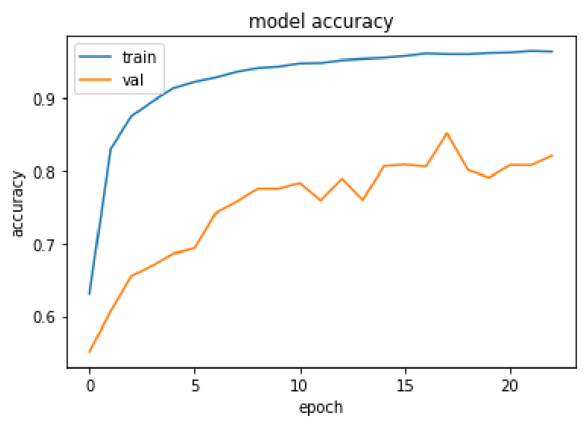
The following Model was found to be the best performing when not training the 1st layer

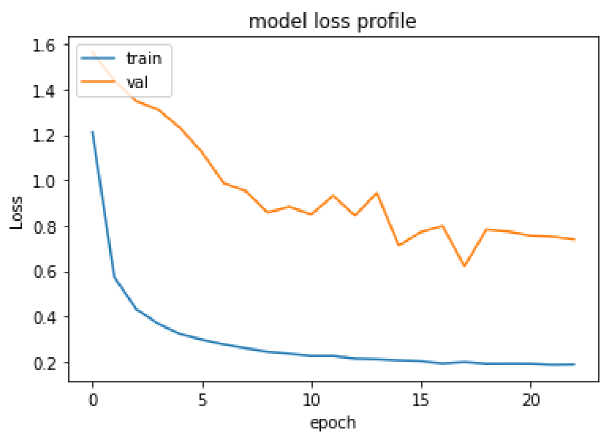


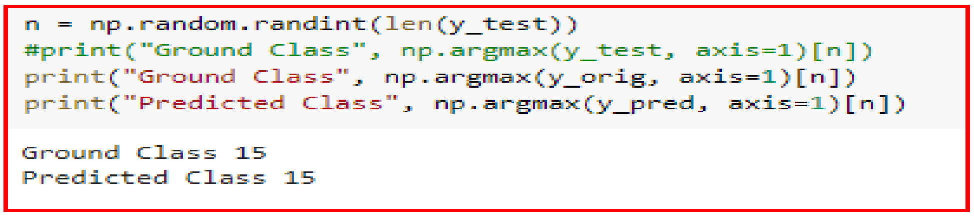
```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.callbacks import EarlyStopping
#### clearing the cache of the previous models tried
backend.clear_session()
#### variables used for building the model
print("MAX Num Words",MAX_NUM_WORDS)
print('num_words : ',num_words)
print('EMBEDDING_DIM : ',EMBEDDING_DIM)
print('MAX_SEQUENCE_LENGTH : ',MAX_SEQUENCE_LENGTH)
print('NUM_CLASSES : ',NUM_CLASSES)
print("With regularization")
epochs=50
batch_size=32
print('Epochs : ',epochs)
print('Batch size : ',batch_size)

model = Sequential()
model.add(Embedding(num_words,
                    EMBEDDING_DIM,
                    input_length=MAX_SEQUENCE_LENGTH,
                    weights= [embedding_matrix],
                    trainable=False))
model.add(Dropout(0.5))
model.add(Conv1D(64, 5, activation='relu',padding='same'))
model.add(MaxPooling1D(pool_size=4))
model.add(LSTM(300,return_sequences=True,kernel_regularizer=regularizers.l1_l2(l1=1e-5, l2=1e-4)))
model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(NUM_CLASSES, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()
```



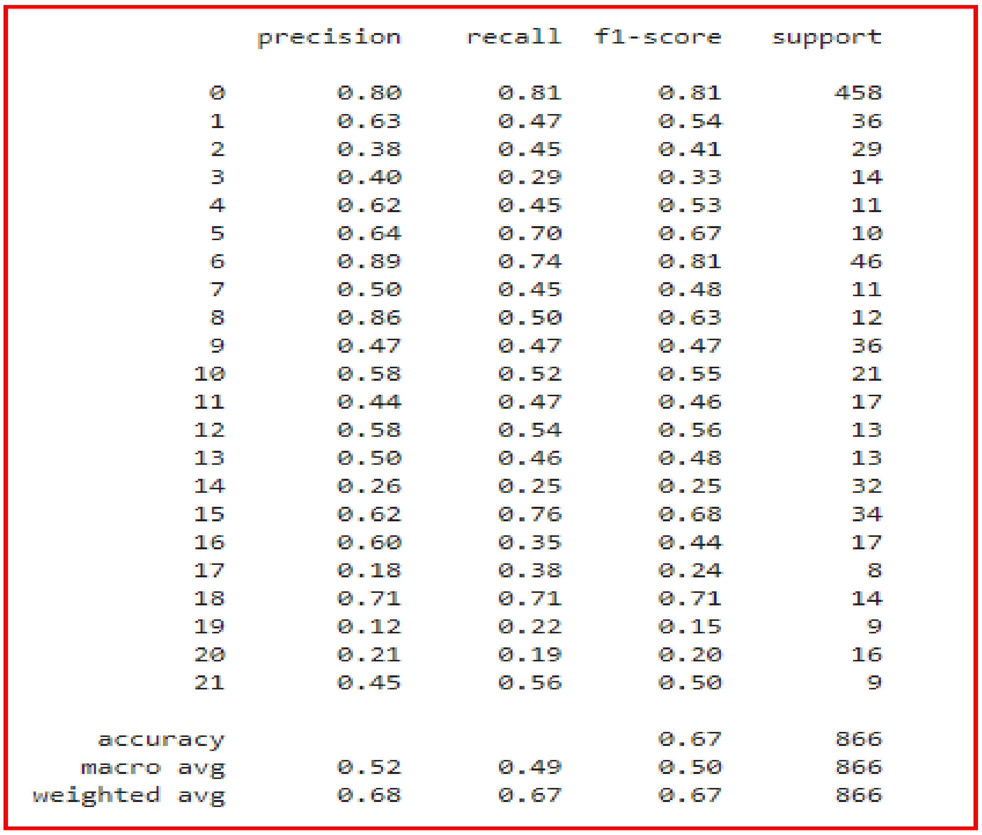
```
MAX Num Words 6000
num_words :  6001
EMBEDDING_DIM :  300
MAX_SEQUENCE_LENGTH :  300
NUM_CLASSES :  22
With regularization
Epochs :  50
Batch size :  32
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 300, 300)          1800300
_________________________________________________________________
dropout (Dropout)            (None, 300, 300)          0
_________________________________________________________________
conv1d (Conv1D)              (None, 300, 64)           96064
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 75, 64)            0
_________________________________________________________________
lstm (LSTM)                  (None, 75, 300)           438000
_________________________________________________________________
dropout_1 (Dropout)          (None, 75, 300)           0
_________________________________________________________________
flatten (Flatten)            (None, 22500)             0
_________________________________________________________________
dense (Dense)                (None, 22)                495022
=================================================================
Total params: 2,829,386
Trainable params: 1,029,086
Non-trainable params: 1,800,300
_________________________________________________________________
```



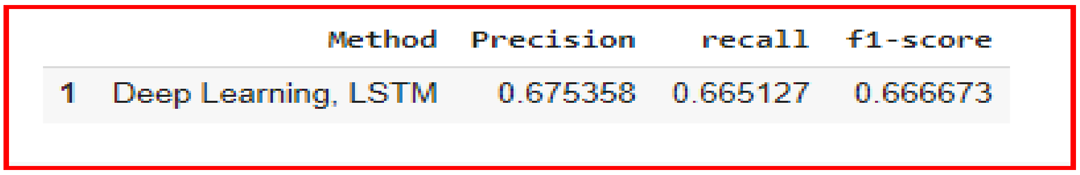




```
n = np.random.randint(len(y_test))
#print("Ground Class", np.argmax(y_test, axis=1)[n])
print("Ground Class", np.argmax(y_orig, axis=1)[n])
print("Predicted Class", np.argmax(y_pred, axis=1)[n])

Ground Class 15
Predicted Class 15
```



```
              precision    recall  f1-score   support

           0       0.80      0.81      0.81       458
           1       0.63      0.47      0.54        36
           2       0.38      0.45      0.41        29
           3       0.40      0.29      0.33        14
           4       0.62      0.45      0.53        11
           5       0.64      0.70      0.67        10
           6       0.89      0.74      0.81        46
           7       0.50      0.45      0.48        11
           8       0.86      0.50      0.63        12
           9       0.47      0.47      0.47        36
          10       0.58      0.52      0.55        21
          11       0.44      0.47      0.46        17
          12       0.58      0.54      0.56        13
          13       0.50      0.46      0.48        13
          14       0.26      0.25      0.25        32
          15       0.62      0.76      0.68        34
          16       0.60      0.35      0.44        17
          17       0.18      0.38      0.24         8
          18       0.71      0.71      0.71        14
          19       0.12      0.22      0.15         9
          20       0.21      0.19      0.20        16
          21       0.45      0.56      0.50         9

    accuracy                           0.67       866
   macro avg       0.52      0.49      0.50       866
weighted avg       0.68      0.67      0.67       866
```



	Method	Precision	recall	f1-score
1	Deep Learning, LSTM	0.675358	0.665127	0.666673

FINAL MODEL ACCURACY AND LOSS

After further reviews and discussion, it was decided to check the Model with training for all the layers. The Glove Embeddings were used as the Kernel Initialization weights and the Model was allowed to train across the entire Augmented Training dataset.

METRICS EVALUATED

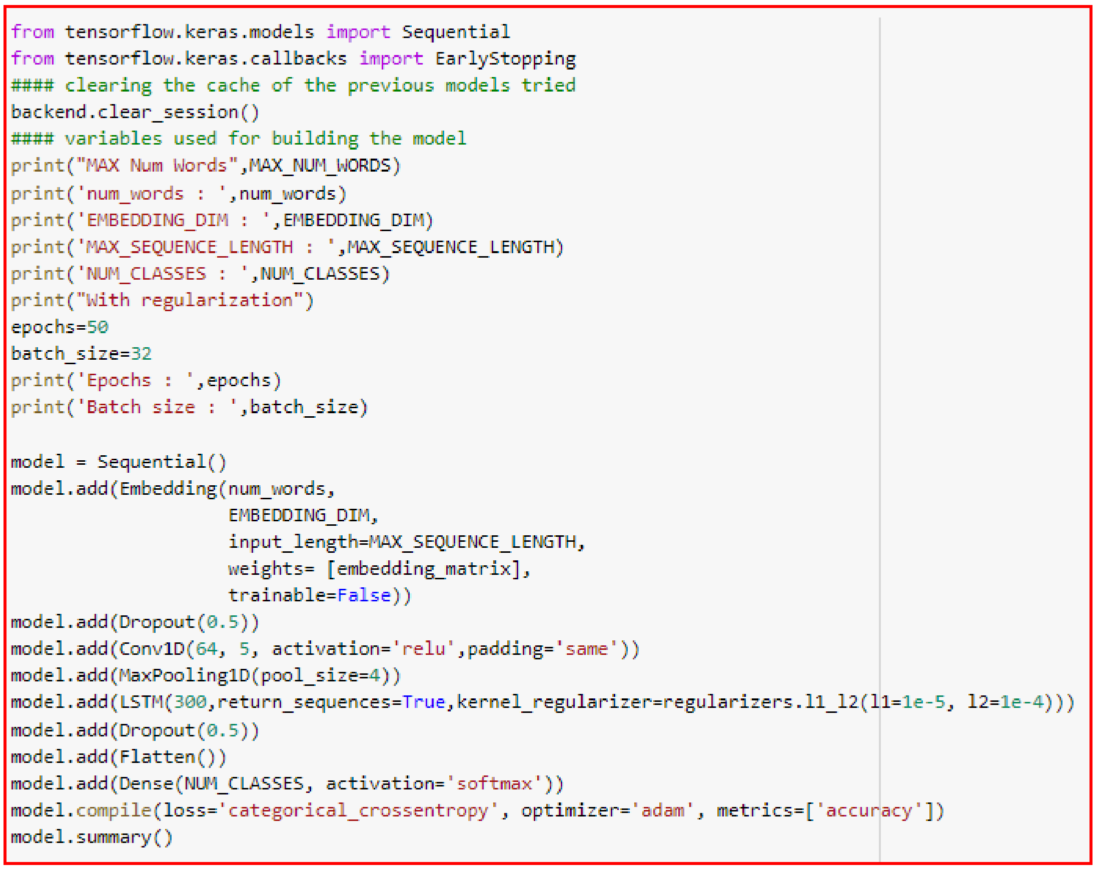
- F1-score is the best indicator for Multiclass classifier. All decisions were based on how the model was performing with respect to F1-Score across all the Classes (Weighted Average

MODEL STRUCTURE:

- Embedding Layer
  - with input of 10000 vectors
  - Input Dimension (300,300)
  - Weights – Glove Embedding
  - Trainable = True
- Dropout 50%
- Conv1D layer size (64, 5) with "same" size padding, using ReLU as the activation
- MaxPooling Layer with Pool size of 4
- LSTM Layer (input of 300 size vector, Return_sequences = True, L1 & L2 kernel regularizers,
- Dropout of 50%
- Flattten layer to prepare the inputs for the last layer Fully connected Dense Layer
- Dense Layer with 22 neurons as output with Softmax Activation

- Model was compiled with the following parameters:

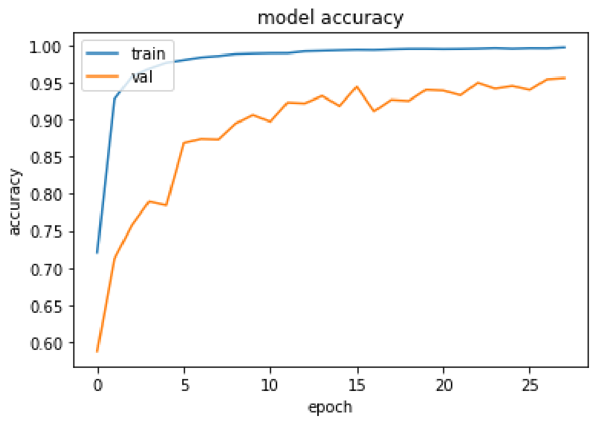
  - Loss Function : Categorical_CrossEntropy as it was a Multi-class classification problem
  - Optimizer – Adam with default values
  - Metrics – Accuracy
  - Epochs = 50
  - Early stopping was set to monitor Validation Loss, with minimum Delta at 0.0001 and patience of 3
  - Shuffle was set to True to remove any bias introduced due to the Augmentation where the synthetic records will be bunched together

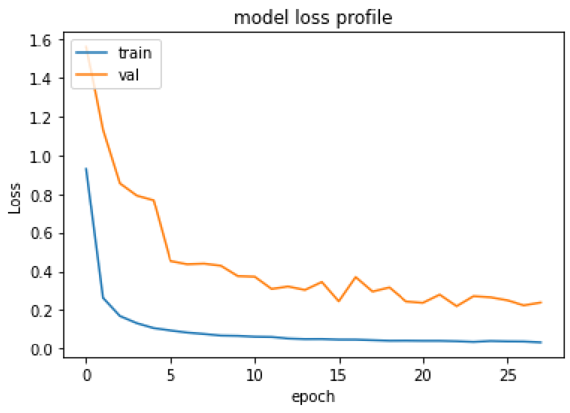


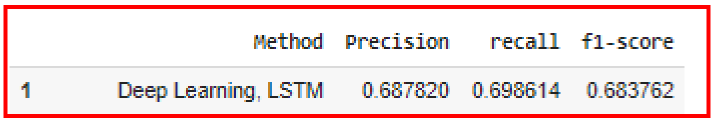
```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.callbacks import EarlyStopping
#### clearing the cache of the previous models tried
backend.clear_session()
#### variables used for building the model
print("MAX Num Words",MAX_NUM_WORDS)
print('num_words : ',num_words)
print('EMBEDDING_DIM : ',EMBEDDING_DIM)
print('MAX_SEQUENCE_LENGTH : ',MAX_SEQUENCE_LENGTH)
print('NUM_CLASSES : ',NUM_CLASSES)
print("With regularization")
epochs=50
batch_size=32
print('Epochs : ',epochs)
print('Batch size : ',batch_size)

model = Sequential()
model.add(Embedding(num_words,
                    EMBEDDING_DIM,
                    input_length=MAX_SEQUENCE_LENGTH,
                    weights= [embedding_matrix],
                    trainable=False))
model.add(Dropout(0.5))
model.add(Conv1D(64, 5, activation='relu',padding='same'))
model.add(MaxPooling1D(pool_size=4))
model.add(LSTM(300,return_sequences=True,kernel_regularizer=regularizers.l1_l2(l1=1e-5, l2=1e-4)))
model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(NUM_CLASSES, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()
```

RESULTS

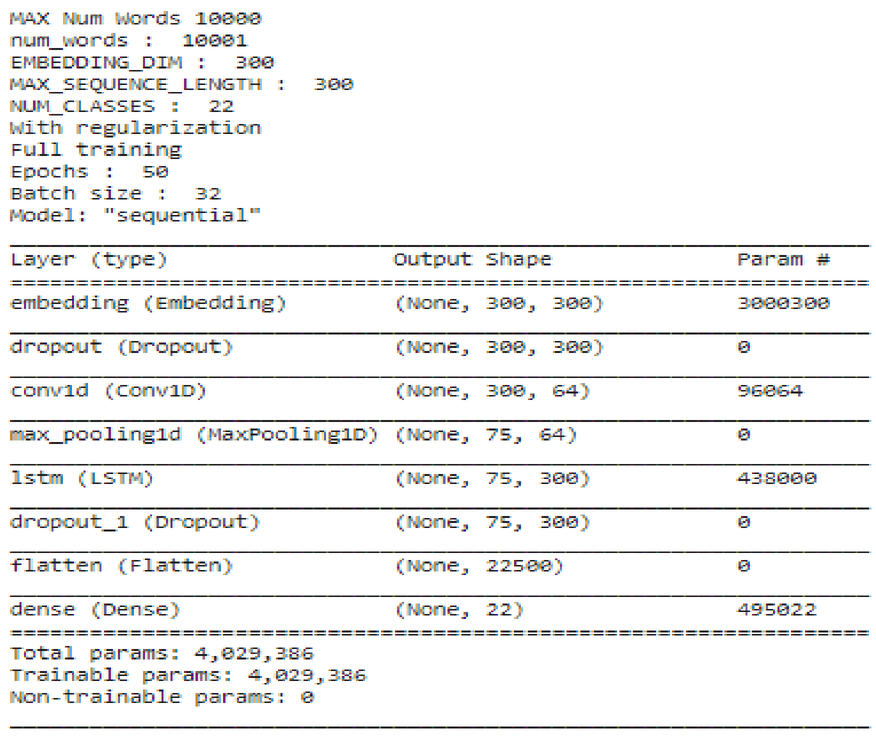
- **With 10000 embeddings, we reached ~68.4% F1 Score, with Validation Accuracy of 0.9557 & Training Accuracy of 0.9968**







	Method	Precision	recall	f1-score
1	Deep Learning, LSTM	0.687820	0.698614	0.683762

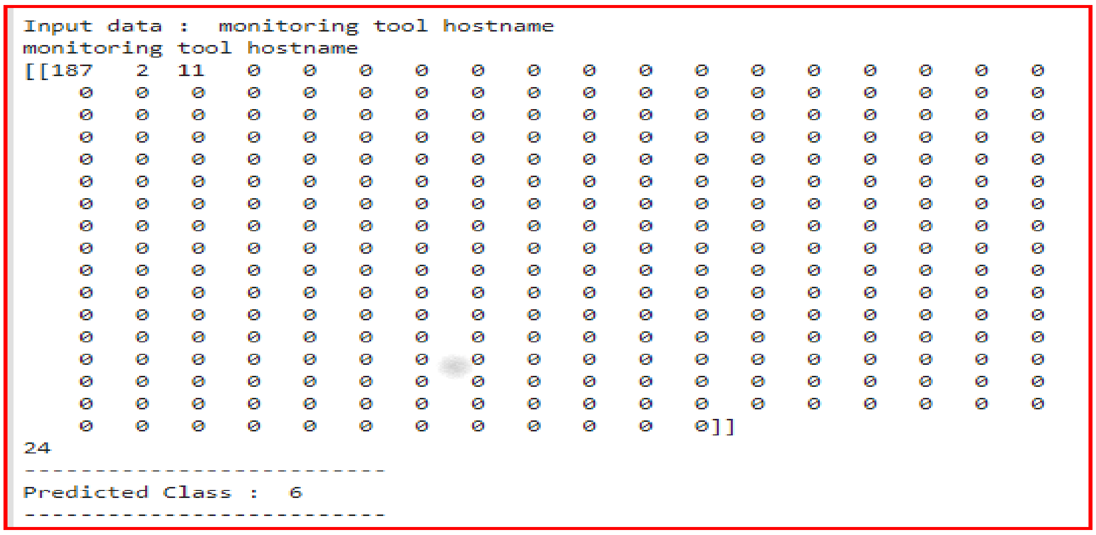

```
MAX Num Words 10000
num_words :  10001
EMBEDDING_DIM :  300
MAX_SEQUENCE_LENGTH :  300
NUM_CLASSES :  22
With regularization
Full training
Epochs :  50
Batch size :  32
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 300, 300)          3000300
_________________________________________________________________
dropout (Dropout)            (None, 300, 300)          0
_________________________________________________________________
conv1d (Conv1D)              (None, 300, 64)           96064
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 75, 64)            0
_________________________________________________________________
lstm (LSTM)                  (None, 75, 300)           438000
_________________________________________________________________
dropout_1 (Dropout)          (None, 75, 300)           0
_________________________________________________________________
flatten (Flatten)            (None, 22500)             0
_________________________________________________________________
dense (Dense)                (None, 22)                495022
=================================================================
Total params: 4,029,386
Trainable params: 4,029,386
Non-trainable params: 0
_________________________________________________________________
```

PLANS TO USE THE BEST MODEL IN PRODUCTION:

- A notebook has been created which can be used standalone to feed any text to it and it will end up with the classification.
- It needs some resources (Tokenizer object used for training, Saved Model, Stopwords & Callerlist pickle objects.

PREDICTION NOTEBOOK

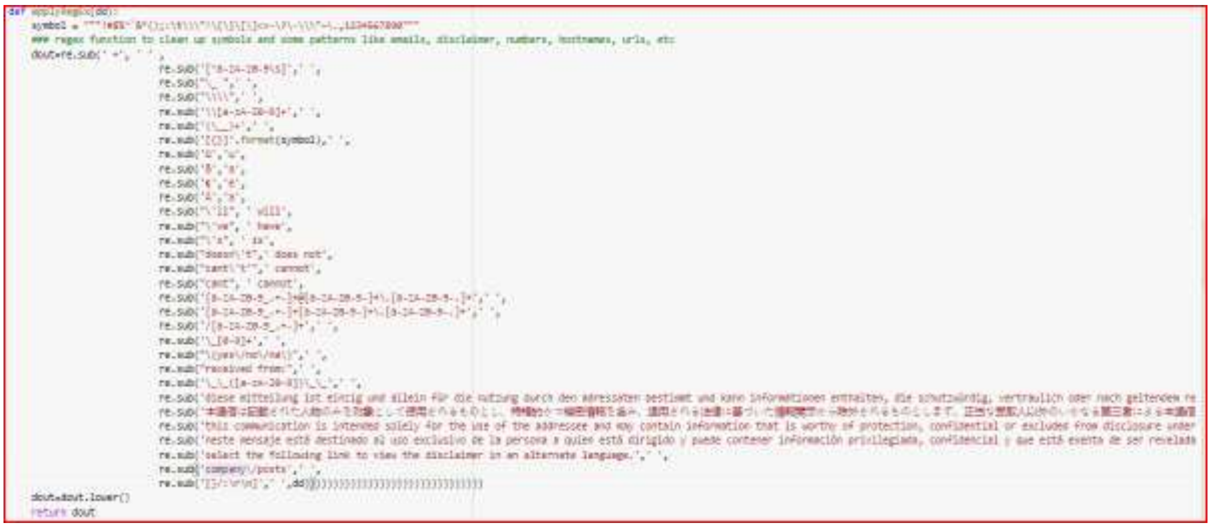
PREDICTION


```
Input data :  monitoring tool hostname
monitoring tool hostname
[[187    2   11    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0]]
24
---------------------------
Predicted Class :  6
---------------------------
```

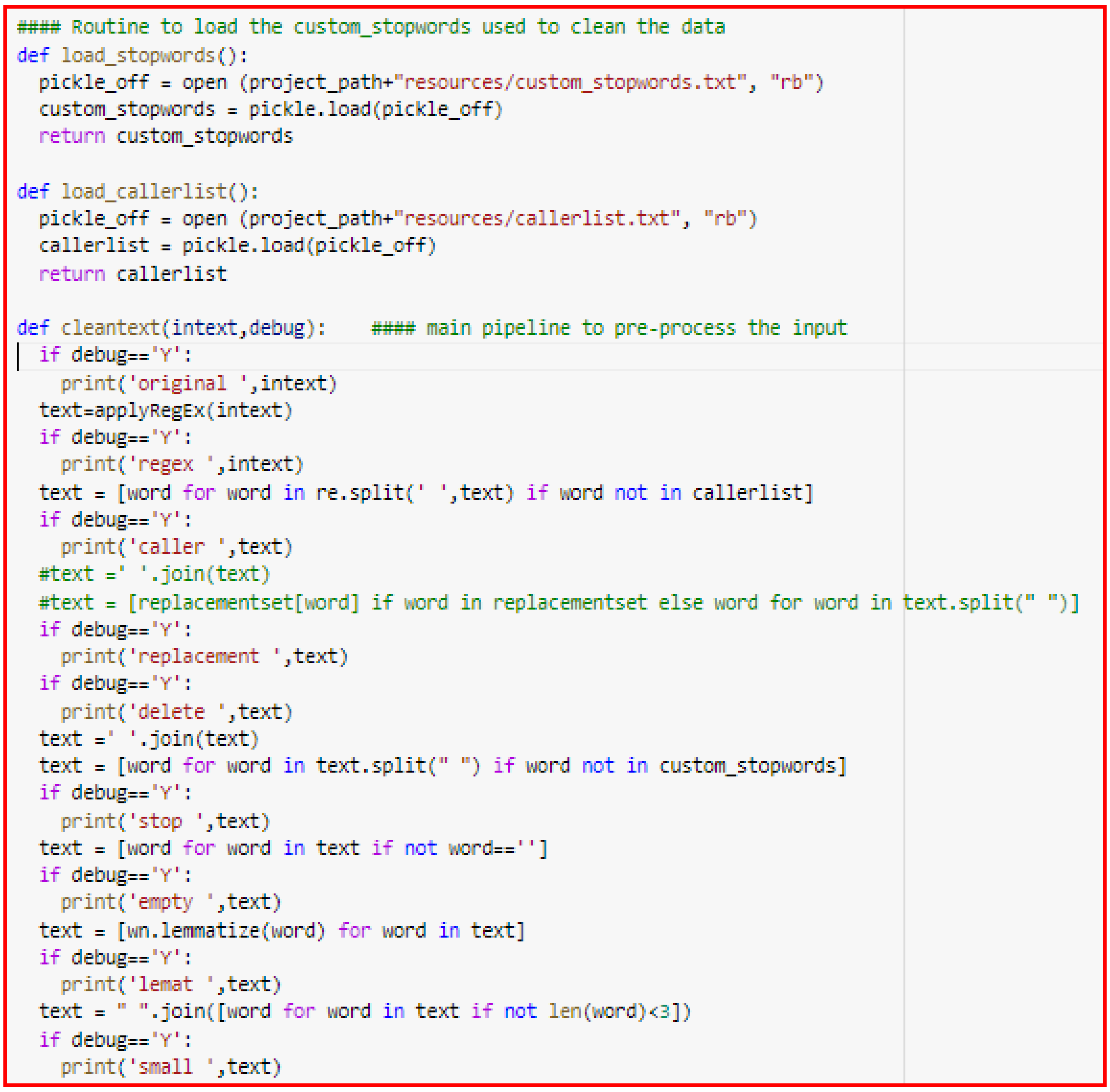
DETAILS OF THE PREDICTION CODE

The code used in the Notebook is a follows:

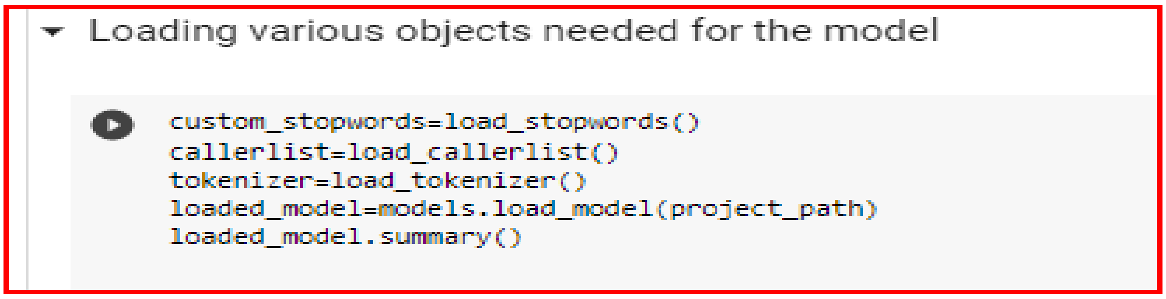
```
### Getting the saved model with architecture and weights
def get_model():
  loaded_model=models.load_model(project_path)
  return loaded_model

### loading the Tokenizer used for training the model
### used to create tokens from the string/text passed into the model
def load_tokenizer():
  # loading tokenizer
  with open(project_path+'/resources/tokenizer.pickle', 'rb') as handle:
    tokenizer = pickle.load(handle)
  return tokenizer

### Routine to translate the input
def translate(text):
  translator = Translator()
  val = translator.translate(text)
  return val
```



```
def applyRegex(dd):
    symbol = """!#$%^&*();:\t\\\"!\{\}\[\]<>-\?\-\\\"~+.,1234567890"""
    ### regex function to clean up symbols and some patterns like emails, disclaimer, numbers, hostnames, urls, etc
    dout=re.sub(' +', ' ',
                    re.sub('[^0-9A-Z0-9\s]',' ',
                    re.sub("\_",' ',
                    re.sub("\\\\",' ',
                    re.sub('\\[a-zA-Z0-9]+',' ',
                    re.sub('(\__)+',' ',
                    re.sub('[{}]'.format(symbol),' ',
                    re.sub('ü','u',
                    re.sub('ß','s',
                    re.sub('€','e',
                    re.sub('ä','a',
                    re.sub("\'ll", ' will',
                    re.sub("\'ve", ' have',
                    re.sub("\'s", ' is',
                    re.sub("doesn\'t",' does not',
                    re.sub("cant\'t",' cannot',
                    re.sub("cant", ' cannot',
                    re.sub('[0-9A-Z0-9_.+-]+@[0-9A-Z0-9-]+\.[0-9A-Z0-9-.]+',' ',
                    re.sub('[0-9A-Z0-9_.+-]+[0-9A-Z0-9-]+\.[0-9A-Z0-9-.]+',' ',
                    re.sub('/[0-9A-Z0-9_.+-]+',' ',
                    re.sub('\_[0-9]+',' ',
                    re.sub("\(yes\/no\/na\)",' ',
                    re.sub("received from:",' ',
                    re.sub('\_\_([a-zA-Z0-9])\_\_',' ',
                    re.sub('diese mitteilung ist einzig und allein für die nutzung durch den adressaten bestimmt und kann informationen enthalten, die schutzwürdig, vertraulich oder nach geltendem re
                    re.sub('本通信は記載された人物のみを対象として使用されるものとし、機密的かつ秘匿情報を含み、適用される法律に基づいた開示義務から除外されるものとします。正当な受取人以外のいかなる第三者による本通信
                    re.sub('this communication is intended solely for the use of the addressee and may contain information that is worthy of protection, confidential or excluded from disclosure under
                    re.sub('este mensaje está destinado al uso exclusivo de la persona a quien está dirigido y puede contener información privilegiada, confidencial y que está exenta de ser revelada
                    re.sub('select the following link to view the disclaimer in an alternate language.',' ',
                    re.sub('company\/posts',' ',
                    re.sub('[:/-\r\n]',' ',dd))))))))))))))))))))))))))))))))
    dout=dout.lower()
    return dout
```

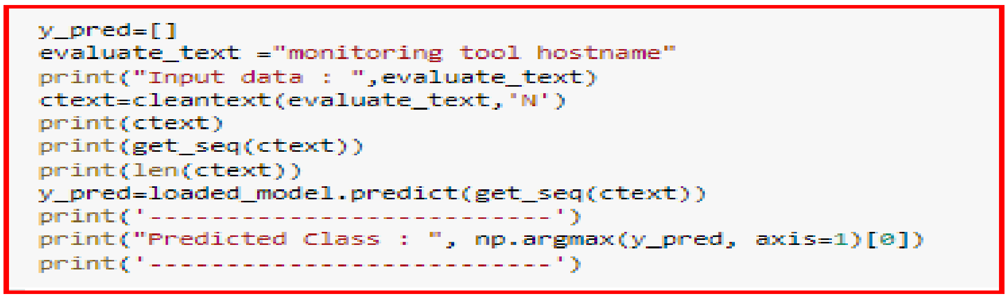


```
#### Routine to load the custom_stopwords used to clean the data
def load_stopwords():
  pickle_off = open (project_path+"resources/custom_stopwords.txt", "rb")
  custom_stopwords = pickle.load(pickle_off)
  return custom_stopwords

def load_callerlist():
  pickle_off = open (project_path+"resources/callerlist.txt", "rb")
  callerlist = pickle.load(pickle_off)
  return callerlist

def cleantext(intext,debug):    #### main pipeline to pre-process the input
  if debug=='Y':
    print('original ',intext)
  text=applyRegEx(intext)
  if debug=='Y':
    print('regex ',intext)
  text = [word for word in re.split(' ',text) if word not in callerlist]
  if debug=='Y':
    print('caller ',text)
  #text =' '.join(text)
  #text = [replacementset[word] if word in replacementset else word for word in text.split(" ")]
  if debug=='Y':
    print('replacement ',text)
  if debug=='Y':
    print('delete ',text)
  text =' '.join(text)
  text = [word for word in text.split(" ") if word not in custom_stopwords]
  if debug=='Y':
    print('stop ',text)
  text = [word for word in text if not word=='']
  if debug=='Y':
    print('empty ',text)
  text = [wn.lemmatize(word) for word in text]
  if debug=='Y':
    print('lemat ',text)
  text = " ".join([word for word in text if not len(word)<3])
  if debug=='Y':
    print('small ',text)
```



Loading various objects needed for the model

```
custom_stopwords=load_stopwords()
callerlist=load_callerlist()
tokenizer=load_tokenizer()
loaded_model=models.load_model(project_path)
loaded_model.summary()
```



```
y_pred=[]
evaluate_text ="monitoring tool hostname"
print("Input data : ",evaluate_text)
ctext=cleantext(evaluate_text,'N')
print(ctext)
print(get_seq(ctext))
print(len(ctext))
y_pred=loaded_model.predict(get_seq(ctext))
print('---------------------------')
print("Predicted Class : ", np.argmax(y_pred, axis=1)[0])
print('---------------------------')
```

LIMITATIONS:

- Overall the F1 Scores are not so high due to the fact that there was huge imbalance in the Class distribution.
- Move balanced data will help improve the accuracy.
- Most of the classes had the same information/text in them, this leads to wrong classification.
- We could not identify any specific patterns unique for any class which could have improved the accuracy

IMPLICATIONS

- Overall this is a good model to further enhance with more Data
- This Model can be used to Predict the Group to which the Ticket should be routed. The model is able to predict the Class efficiently and can be deployed such that the ticket / problem text can be sent to this model directly.
- Data is processed and fed into the saved model.
- This model will eventually reduce the dependency on the L1/L2 Teams. Based on the Class identified, the Tickets can be automately routed by downstream systems to L3 or Vendors. This will save the organization

CLOSING REMARKS

Overall this project has enabled us to build an efficient model which can be used for Automatic Ticket Assignment. This will reduce the manual mistakes and time & efforts in identifying the team to which the tickets should be routed. This will bring in saving in resources and manpower to the Organization. This model can be trained repeatedly to continually improve its effectiveness. Over period of time as more data is feed into the model for training the accuracies will also improve.

LIST OF RESOURCES/OUTPUT: FIND IT ON GITHUB

NOTEBOOKS:

- Capstone_Project_final.ipynb – Complete notebook
- Capstone_final_submit_production_v3.ipynb – only for prediction

OUTPUT

- train_review_tfidf_grp.csv -Tfid Vectors

- word2vec24_300.model – word2vec 300 dimension model
- word2vec_vectors_300.csv - word2vec 300 dimension vectors
- lstm.h5 – final model weights

RESOURCES

- tokenizer.pickle – tokenizer object for prediction notebook
- custom_stopwords.txt – stopwords used
- replacement_cn_text.csv – Chinese offline translations
- replacement_de_text.csv – German offline translations
- todeleteset.txt – List of words to delete from the dataset
- toreplaceset.pickle – replacement for misspelt and other merged words
- callerlist.txt – list of email ids which appear in the data

DATA

- input_data.xlsx – original xlsx file with the data
- traintest_aug.csv – augmented train+test data
- onlytrain_aug.csv – augmented train set
- onlytest_aug.csv – original hold out set
- eda_final_dataset.csv – dataset after translation and clean up
- translated_dataset.csv – dataset after translation

Find the notebook and resources on GITHUB