Automated Ticket Assignment:

NLP Capstone Project

Group-2 NLP-1

Mentor: Dipanshu Haldar

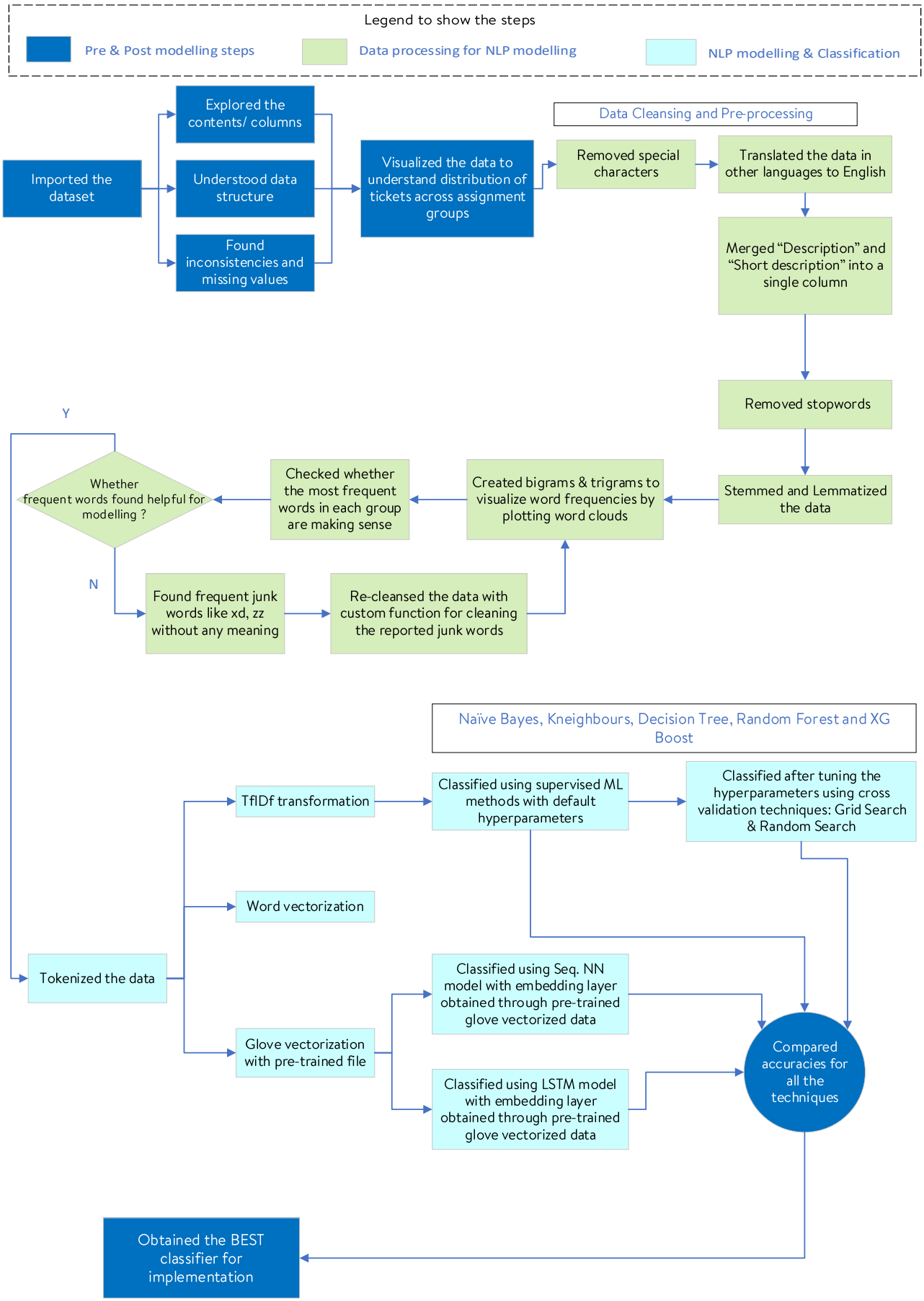
Team members:

- Pranav Kumar Singh
- Bhawna Jha
- Janki Pandya
- Ramakrishna Pothula
- Sumit

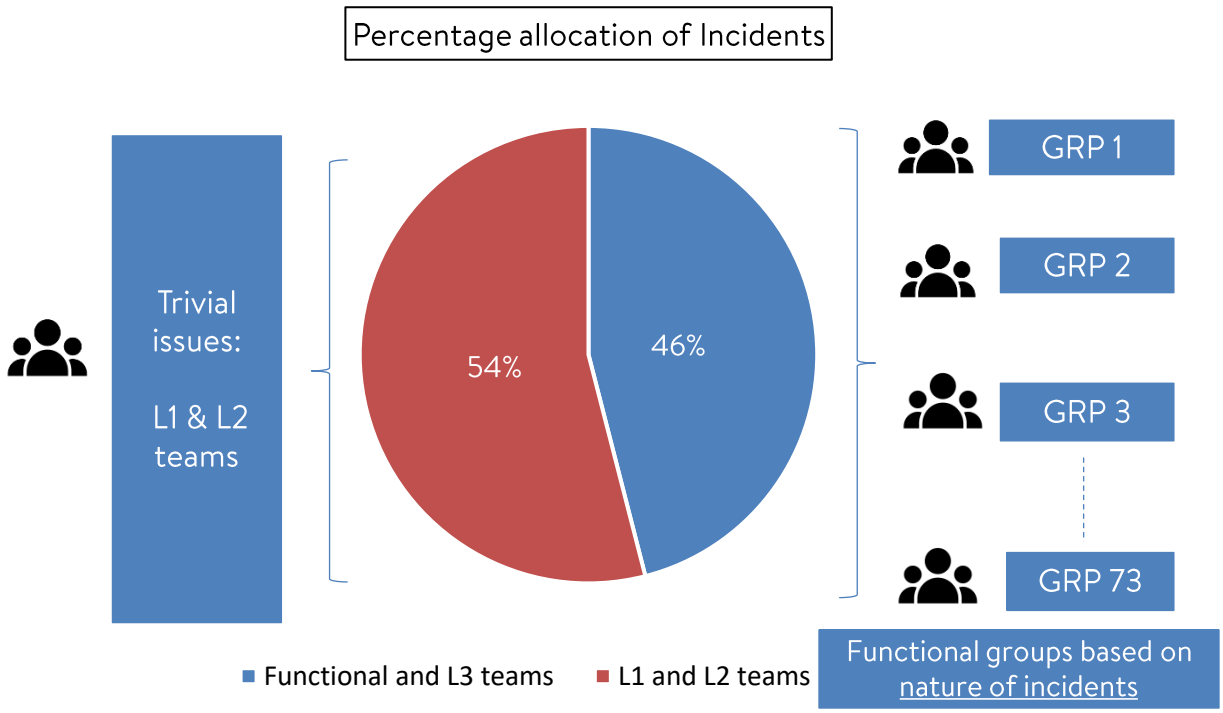
Table of Contents

Table of Contents

Summary of Project stages in Modelling & Classification



AS-IS state, users and pain-points



How are incidents handled today?

- Around ~54% of the incidents are resolved by L1 / L2 teams who act as 1st point of contact or triage for incidents
- Incase L1 / L2 is unable to resolve, they 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 ~46% of incidents are resolved by Functional / L3 teams. Incase if vendor support is needed, they will reach out for their support towards incident closure.

Pain points

- Volume of incidents: Min. ~25-30% of incidents are **manually** reviewed by L1/L2 as of today before they assign the tickets to respective functional groups
- Efforts spent
  - ~ 15 min spent for **manual ticket assignment** for each incident
  - 1 FTE needed only for assignment
  - ~25% wrong allocation of incidents to functional groups due to human error
  - Additional efforts for re-assignment

The graphs suggests the size of the problem which needs to be solved as part of the project.
The discussed discrepancies and inconsistencies finally result in poor customer experience

Data Analysis and observation

Data exploration

The dataset has been provided as an excel file with four columns as below:

- Short description: Specifying the nature of issue
- Description: The actual issue
- Caller: UserID of the caller of the issue
- Assignment group: Group to which the issue was assigned

	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
...	...	...	...	...
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

Understanding the structure of data

```
Data.shape
```

```
(8500, 4)
```

```
Data.describe(include='all')
```

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

- There are 8500 data points spread across the four columns.
- There are 74 unique assignment groups to which the issues are being assigned
- “password reset” issue seems the most common which has occurred 38 times

Finding inconsistencies in the data

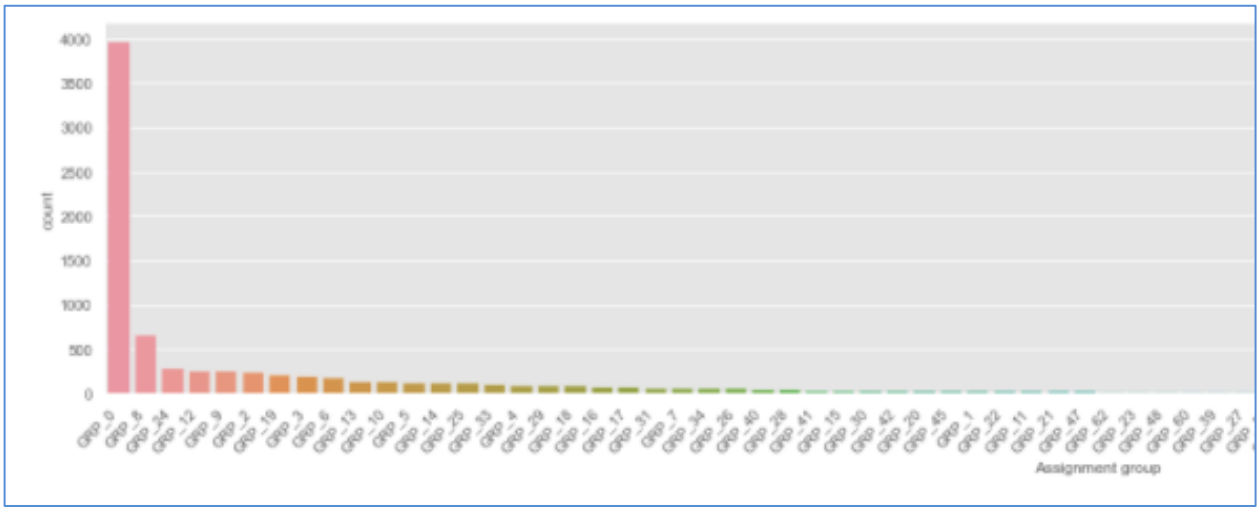
```
Data["Assignment group"].value_counts()
```

```
GRP_0     3976
GRP_8      661
GRP_24     289
GRP_12     257
GRP_9      252
           ...
GRP_70       1
GRP_67       1
GRP_35       1
GRP_73       1
GRP_64       1
Name: Assignment group, Length: 74, dtype: int64
```

- There are few assignment groups which gets good number of tickets (GRP_8, 24, 12 and 9) and there are few which hardly receive any ticket (GRP_70, 67, 35, 73 and 64)
- GRP_0 is an outlier with 3976 tickets about of 8500 i.e. ~50% of the issues

Visualizing different patterns

We will plot frequency distribution in **descending order**, across various groups which were assigned the tickets

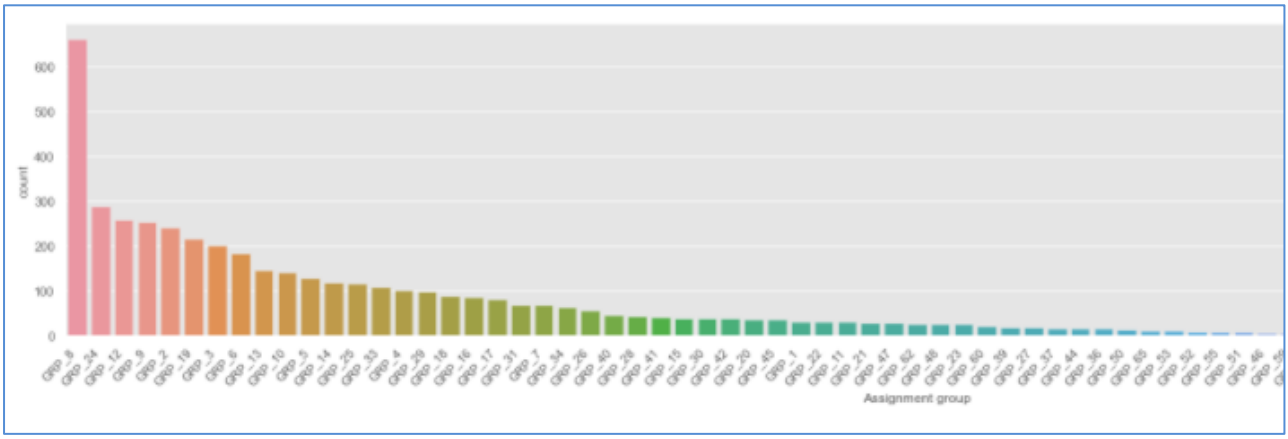


Observation:

- GRP_0 is an outlier which received~50% of the total number of tickets (~4000 tickets)
- There are around 50% of the groups which receive considerable number of tickets, as observed from the graph

Plotting the frequency distribution once more without the outlier GRP_0

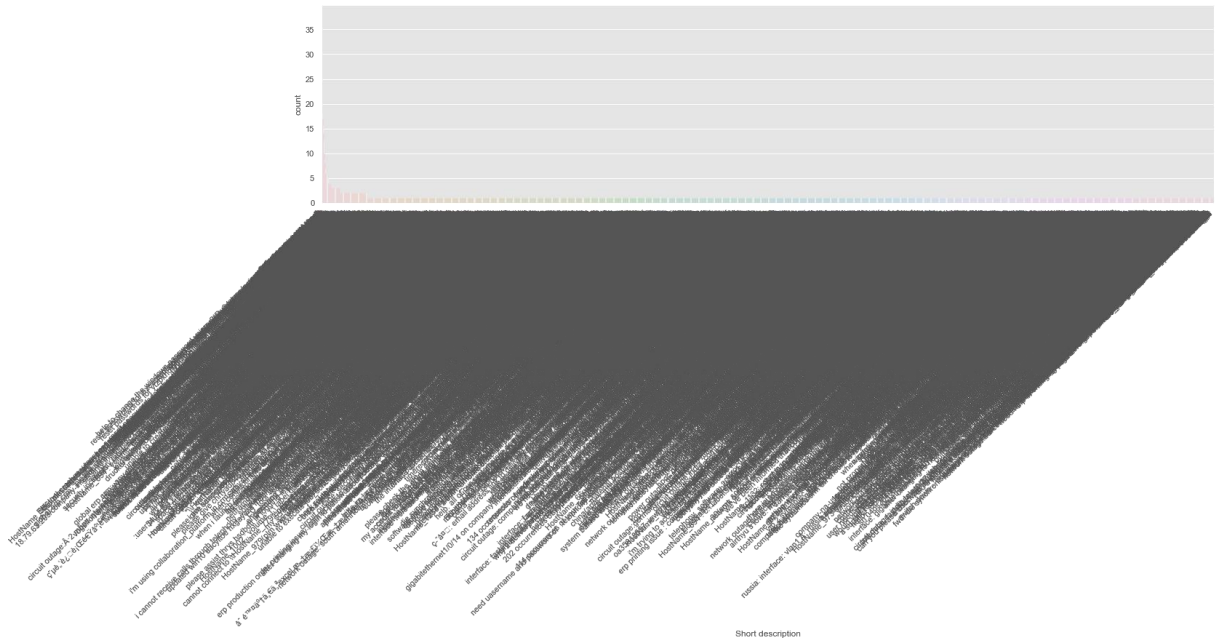
To be continued



Observation:

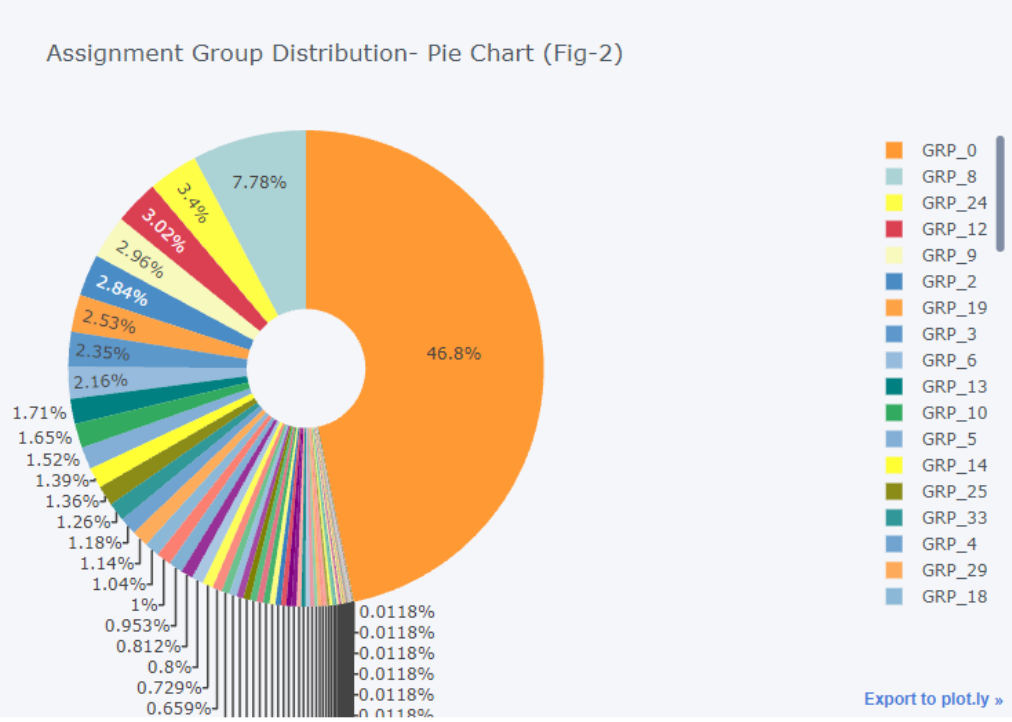
- GRP_8 has more than 600 tickets, followed by 5 groups with count of tickets between 200 - 300 range
- There are 7 groups with count of tickets between 100 - 200 range
- There are around 18 groups with very minimal count of tickets

Frequency of occurrence of issues



- We can see that the distribution is almost uniform for all the short description categories except for a few.
- The ones which were found most common were around “password reset”.

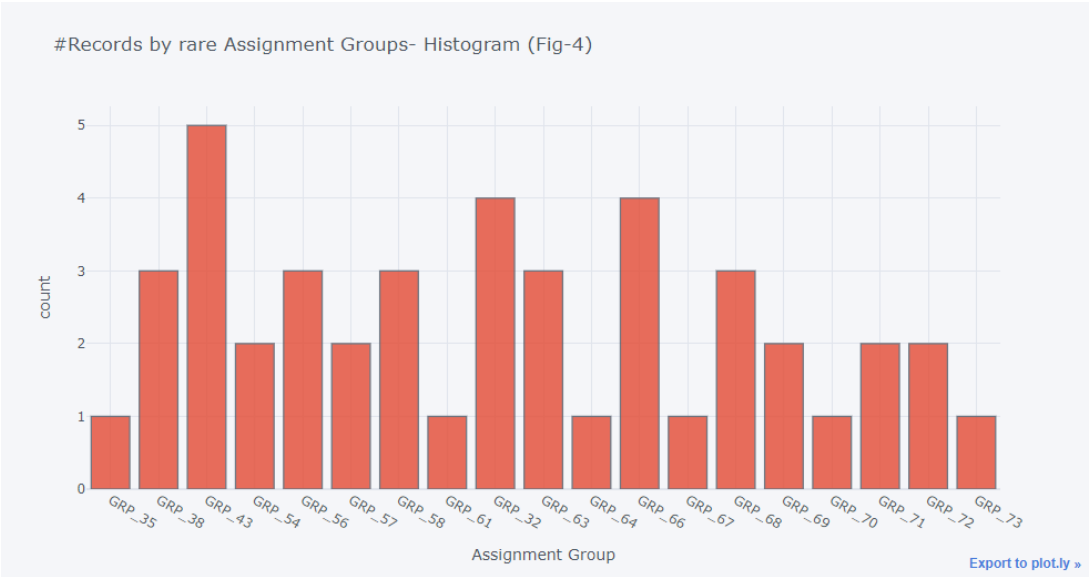
Understanding percentage distribution of tickets



Observation:

- As analyzed above, we see that GRP_0 has the highest number (46.8%) of the assigned tickets, followed by GRP_8.
- Other groups which receive good number of tickets are: GRP_24 (3.4%), GRP_12 (3.02%), GRP_9 (2.96%)

Finding groups which hardly get any ticket



Observation:

- There are 19 groups which receive less than 5 tickets.
- These groups are: GRP_35, 38, 43, 54,56, 57, 58, 61, 32, 63, 64, 66, 67, 68, 69, 70, 71, 72, 73

Pre-processing methods and results

Below are the steps that have been applied to pre-process the data:

- Dealing with missing values
- Data Translation
- Removing stop words

Dealing with missing values

For any Data Scientist, its very normal to deal with data sets having missing terms and still be able to manage and create a good predictive model out of it. Here, we have identified some missing data across two of the critical attributes – Description and Short Description. Short Description has 8 values missing whereas Description only 1 out of 8500.

```
<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
```

Therefore, we can remove one record where Description is not available while filling up missing values in Short Description using other similar description texts

Data Translation

Some entries in the dataset have been identified in other languages such as German, Spanish etc. In order to maintain the consistency in the language of data for NLP to work, we have translated the data into English using Google TranslateAPI of package deep_translator. This API accepts the text as input of any language form and translates into the language code provided in target parameter. Besides, to upsample the data from minority classes, we are translating the texts into German and then re-translate it back to English using the same API to add some more data with variations.

Code Sample:

```
from google_trans_new import google_translator
from deep_translator import GoogleTranslator
for i in range(0,len(Data)):
  print("i:",i," ",Data.iloc[i]['Description'])
  result_lang = detect(Data.iloc[i]['Description'])
 # print(result_lang)
  if result_lang == 'en':
    continue
  else:
    #print(Data.iloc[i]['Short description'])
    #print(Data.iloc[i]['Description'])
    translator = google_translator()
      Data.loc[i,'Short description']=GoogleTranslator(source=result_lang,
target='en').translate(Data.iloc[i]['Short description'])
            Data.loc[i,'Description']=GoogleTranslator(source=result_lang,
target='en').translate(Data.iloc[i]['Description'])
```

Sample Exceution:

Input ⇒ i: 8428 an mehreren pc lassen sich verschiedene prgramdntyme nicht ffnen bereich cnc

Output ==>

Translated 8428 : Different program types cannot be opened on more than one pc. cnc area

Removing stop words

Short Description and Description are the two features that are primarily used as inputs to assign the ticket automatically. Hence, it is necessary that we preprocess them by removing stop words.

In order to identify and remove stop words from given text, the 'Stopwords' list from nltk library has been used.

Input text snippet before Stopwords Removal:

```
['-verified user details.(employee# & manager name)\r\n-checked the user name in ad and reset the password.\r\n-advised the use
 '\r\n\r\nreceived from: hmjdrvpb.komuaywn@gmail.com\r\n\r\nhello team,\r\n\r\nmy meetings/skype meetings etc are not appearing
 '\r\n\r\nreceived from: eylqgodm.ybqkwiam@gmail.com\r\n\r\nhi\r\n\r\ni cannot log on to vpn\r\n\r\nbest ',
 'unable to access hr_tool page',
 'skype error ',
 'unable to log in to engineering tool and skype',
 'event: critical:HostName_221.company.com the value of mountpoint threshold for /oracle/SID_37/erpdata21/sr3psa1d_7/sr3psa1d.da
 "ticket_no1550391- employment status - new non-employee [enter user's name]",
 'unable to disable add ins on outlook',
 'ticket update on inplant_874773',
 'engineering tool says not connected and unable to submit reports',
 'hr_tool site not loading page correctly',
 'unable to login to hr_tool to sgxqsuojr xwbesorf cards',
 'user wants to reset the password',
 'unable to open payslips ',
```

Text after Stop Words Removal

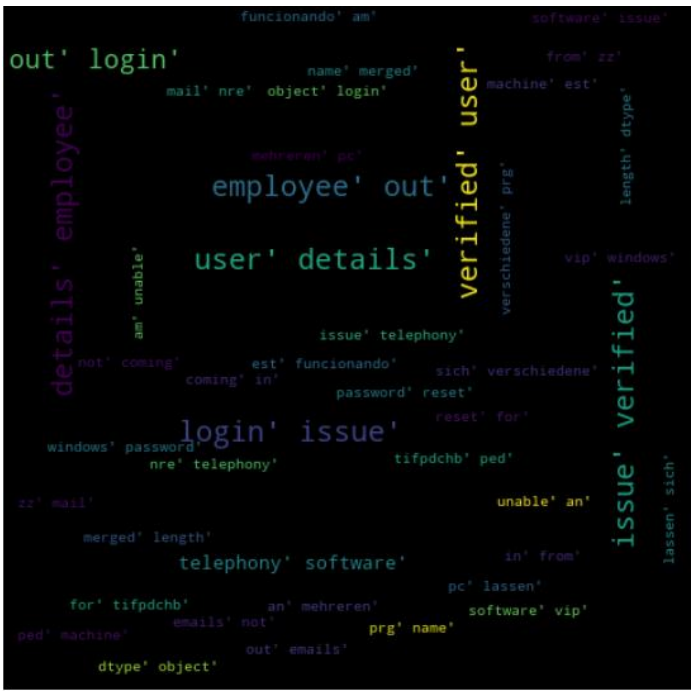
```
['login issue',
 'outlook',
 'cant log in to vpn',
 'unable to access hr_tool page',
 'skype error ',
 'unable to log in to engineering tool and skype',
 'event: critical:HostName_221.company.com the value of mountpoint threshold for /oracle/SID_37/erpdata21/ ',
 "ticket_no1550391- employment status - new non-employee [enter user's name]",
 'unable to disable add ins on outlook',
 'ticket update on inplant_874773',
 'engineering tool says not connected and unable to submit reports',
 'hr_tool site not loading page correctly',
 'unable to login to hr_tool to sgxqsuojr xwbesorf cards',
 'user wants to reset the password',
 'unable to open payslips ',
 'ticket update on inplant_874743',
 'unable to login to company vpn',
 'when undocking pc , screen will not come back',
 'erp SID_34 account locked',
 'unable to sign into vpn',
 'unable to check payslips',
 'vpn issue',
 'unable to connect to vpn',
 'user called for vendor phone number',
 'vpn not working',
```

Merging "Short description" and "Description" into one column i.e. "merged"

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

Finding Wordclouds in the given dataset using bigram and trigram models

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



Most common words for GRP_0

Most common words for GRP_8

Observation from wordclouds:

- We see that the login issues has been prevalent in most of the groups including GRP_0 and GRP_8
- The above observation suggests that there is a room for better grouping/ classification of tickets before assigning it to the right functional groups

Stemming, Tokenization & Lemmatization

Stemming and Lemmatization

For grammatical reasons, documents are going to use different forms of a word, such as organize, organizes, and organizing

The goal of both stemming and lemmatization is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form. For instance:

am, are, is → be
car, cars, car's, cars' → car

The result of this mapping of text will be something like:

the boy's cars are different colors → the boy car be differ color

We have used porter stammer here, because the most common in English is Porter stemmer. The rules contained in this algorithm are divided in five different phases numbered from 1 to 5. The purpose of these rules is to reduce the words to the root

Tokenization

Tokenization is a common task a data scientist comes across when working with text data. It consists of splitting an entire text into small units, also known as tokens. Most Natural Language Processing (NLP) projects have tokenization as the first step because it's the foundation for developing good models and helps better understand the text we have.

When you need to tokenize text written in a language other than English, you can use spaCy. This is a library for advanced natural language processing, written in Python and Cython, that supports tokenization for more than 65 languages.
Although for languages like Spanish and English, tokenization will be as simple as separating by whitespace, for non-romance languages such as Chinese and Japanese, the orthography might have no spaces to delimit "words" or "tokens." In such cases, a library like spaCy will come in handy

Vectorization of text data

The next step is to further transform the cleaned text into a form that the machine learning model can understand. This process is known as Vectorizing

```
corpus = [] #  Creating an array "corpus" to store all the processed data
Datafinal['word_count']=Datafinal['merged'].apply(lambda x: len(str(x).split(" ")))
print(Datafinal.word_count.head)
ds_count=len(Datafinal.word_count)
print(ds_count)
for i in range(ds_count):
    txt = Datafinal.iloc[i]['merged']
    txt = txt.lower()
    txt=txt.split()
    ps=PorterStemmer()
    lem=WordNetLemmatizer()
    txt = [lem.lemmatize(word) for word in txt if not word in stopwords.words('english')]
    #txt=" ".join(txt)
    corpus.append(txt)
```

Word Embeddings

As we know that many Machine Learning algorithms and almost all Deep Learning Architectures are not capable of processing strings or plain text in their raw form. In a broad sense, they require numerical numbers as inputs to perform any sort of task, such as classification, regression, clustering, etc. Also, from the huge amount of data that is present in the text format, it is imperative to extract some knowledge out of it and build any useful applications.
In short, we can say that to build any model in machine learning or deep learning, the final level data must be in numerical form because models don't understand text or image data directly as humans do.
Vectorization or word embedding is the process of converting text data to numerical vectors. Later those vectors are used to build various machine learning models. In this manner, we say this as extracting features with the help of text with an aim to build multiple natural languages, processing models, etc

6.1. Creating a column 'list_processed_text' to store all the processed texts in form of lists

```
Datafinal['list_processed_text']=''
for k in range(len(Datafinal['processed_text'])):
  new_list=[]
  print("k:",k)
  print("len:",type(Datafinal['processed_text'][k]))
  print("Data:",Datafinal['processed_text'][k])
  for i in range(len(Datafinal['processed_text'][k])):
        new_list.append([Datafinal['processed_text'][k][i]])
  #Datafinal.iloc[[k],['list_processed_text']]=new_list

  print("processed_text:",Datafinal.iloc[k]['processed_text'])
  Datafinal.at[k, 'list_processed_text'] = new_list
  print("list_processed_text:",Datafinal.iloc[k]['list_processed_text'])
```

6.2. Creating word vectors from list of processed texts

```
Datafinal['word_vec']=''
for i in range(len(Datafinal['list_processed_text'])):
    print("i:",i)
#training the corpus to generate the co occurence matrix which is used in GloVe
    corpus_glv.fit(Datafinal.iloc[i]['list_processed_text'], window=10)
#creating a Glove object which will use the matrix created in the above lines to create embeddings
#We can set the learning rate as it uses Gradient Descent and number of components
    glove = Glove(no_components=5, learning_rate=0.05)
    glove.fit(corpus_glv.matrix, epochs=30, no_threads=4, verbose=True)
    glove.add_dictionary(corpus_glv.dictionary)
    Datafinal['word_vec'][i]=glove.word_vectors
```

Glove Vectorization

GloVe allows us to take a corpus of text, and intuitively transform each word in that corpus into a position in a high-dimensional space. This means that similar words will be placed together

Using a pre-trained Glove file “glove.6B.100d” for NLP modelling

Each line of the text file contains a word, followed by N numbers. The N numbers describe the vector of the word’s position. For our file N is 100, since we are using glove.6B.100d

- We get the Loaded 400000 word vectors.
- Vocab size is 12893
- Embedded vector is 100

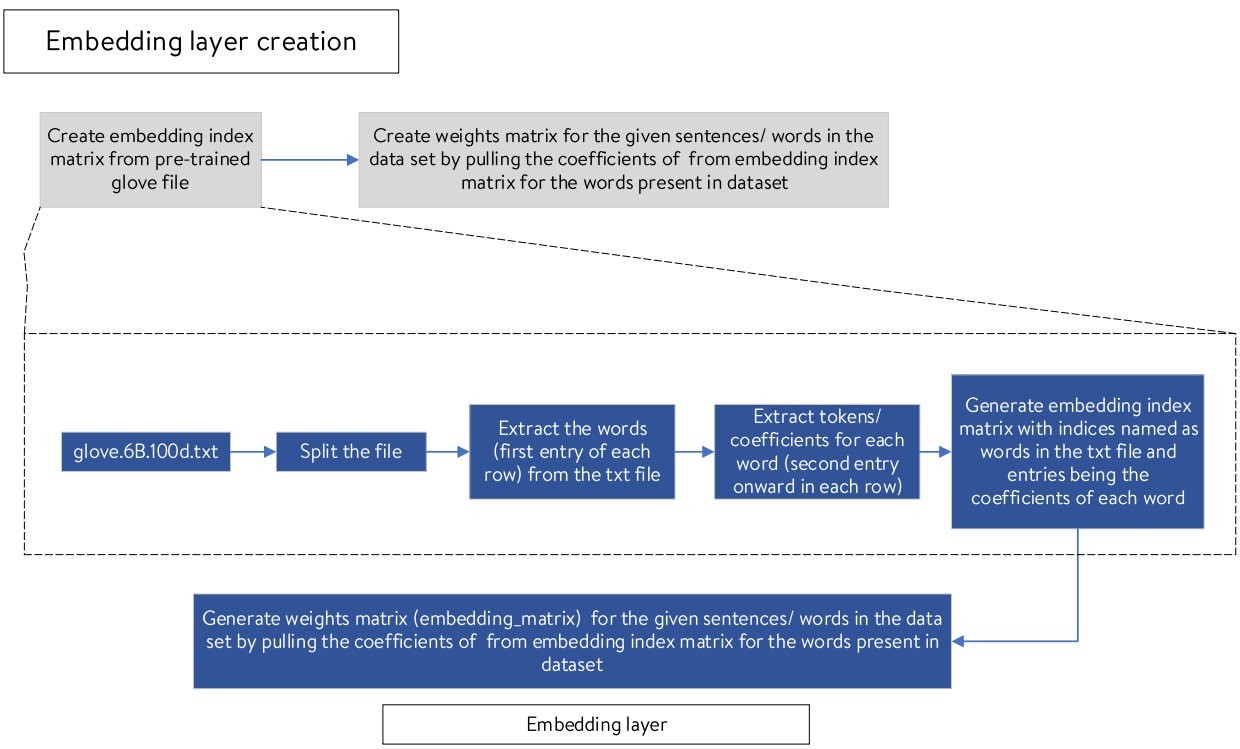
Creation of word embedding using pre-trained file "glove.6B.100d.txt"

```
from numpy import array
from numpy import asarray
embeddings_index = dict()
f = open(r'C:\Users\p0s041d\Documents\Program\Learning\AIML\AIML\AIML\Projects\Capstone\glove.6B.100d.txt', encoding="utf8")
for line in f:
    values = line.split()
    word = values[0]
    coefs = asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()
print('Loaded %s word vectors.' % len(embeddings_index))
```

```
Loaded 400000 word vectors.
```

Embedding layer creation using Glove text file

We created weights matrix from the pre-trained Glove text file and finally created embedding layer for neural network modelling as below:



NN modelling for classification of tickets

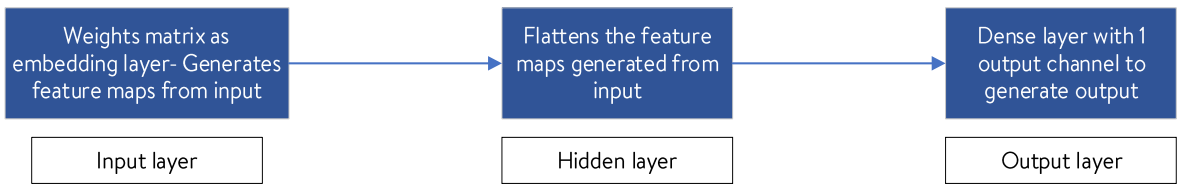
To classify the tickets, we applied NN models both with word embeddings generated through pre-trained glove file

Model Architecture

We built NN model with 3 layers:

a) **Input layer**: Made from word embeddings generated with weights defined as per pre-trained glove text file: glove.6B.100d.txt
b) **Intermediate hidden layer:** The input has been flattened in this layer
c) **Output layer:** A dense layer with 1 output channel to classify the ticket into one of the groups

Defining the Model




```
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Embedding
# define model
model = Sequential()
e = Embedding(vocab_size, 100, weights=[embedding_matrix], input_length=4, trainable=False)
model.add(e)
model.add(Flatten())
model.add(Dense(1, activation='sigmoid'))
# compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
# summarize the model
print(model.summary())
```

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_7 (Embedding)      (None, 4, 100)            1285200
_________________________________________________________________
flatten_7 (Flatten)          (None, 400)               0
_________________________________________________________________
dense_7 (Dense)              (None, 1)                 401
=================================================================
Total params: 1,285,601
Trainable params: 401
Non-trainable params: 1,285,200
_________________________________________________________________
None
```

Label encoding the output before passing to the model

To pass the “Assignment groups” data while fitting the model, we converted it into a **categorical label encoded variable** called “target” as below:

Creating a target categorical column as a counterpart of "Assignment group" column

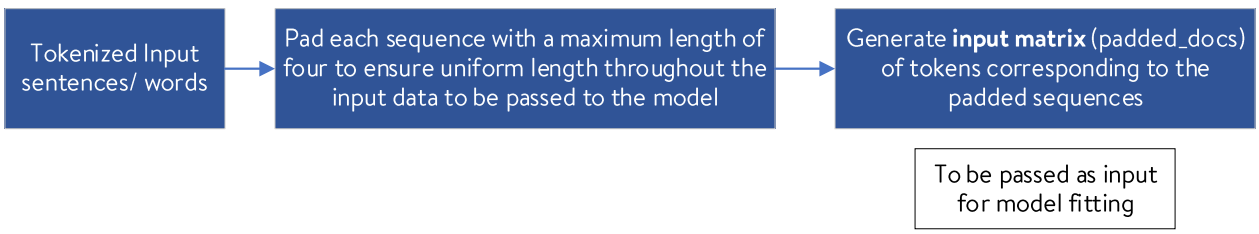
```
Datafinal['target'] = Datafinal['Assignment group'].astype('category').cat.codes
Datafinal.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8448 entries, 0 to 8447
Data columns (total 10 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Short description    8448 non-null   object
 1   Description          8448 non-null   object
 2   Caller               8448 non-null   object
 3   Assignment group     8448 non-null   object
 4   merged               8448 non-null   object
 5   word_count           8448 non-null   int64
 6   processed_text       8448 non-null   object
 7   list_processed_text  8448 non-null   object
 8   word_vec             8448 non-null   object
 9   target               8448 non-null   int8
dtypes: int64(1), int8(1), object(8)
memory usage: 602.4+ KB
```

Label encoding the target column

```
from sklearn import preprocessing
encoder = preprocessing.LabelEncoder()
# encoding train labels
encoder.fit(Datafinal['target'])
Datafinal['target'] = encoder.transform(Datafinal['target'])
```

Padding the input before passing to the model

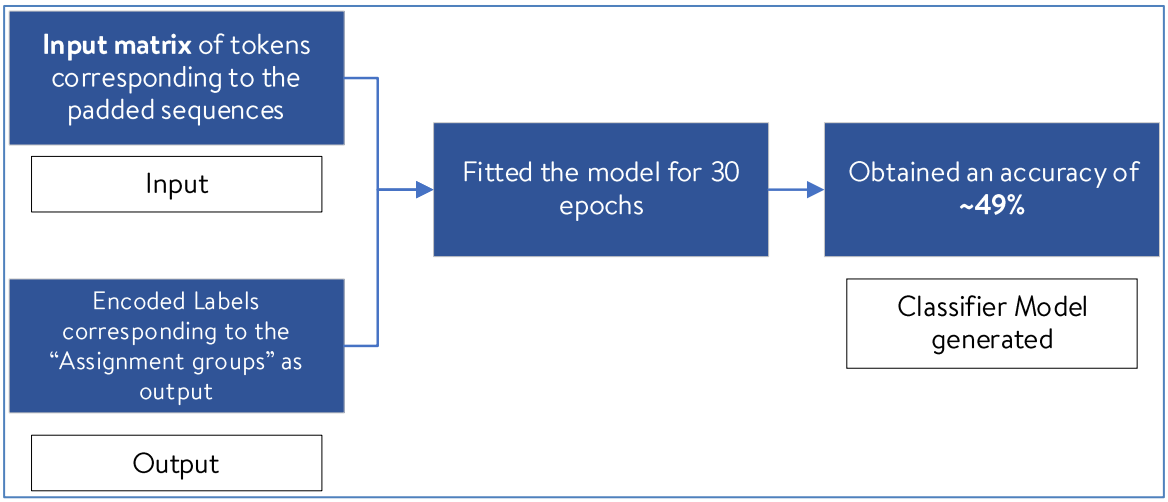


To pass the tokenized input data, we need to first ensure that all the tokenized vectors are of same lengths. For this we padded the sequences using “post” padding and kept the maximum length = 4. This resulted into the input matrix (padded_docs) of shape: 8448 rows * 4 columns which later was passed as input to the model

Fitting the model

To fit the model, we passed both the padded input matrix (padded_docs) and the label encoded output data “target” (assignment groups label data)

Detailed explanation in next page



We used 30 epochs for training the NN model. The model resulted into 49% accuracy

```
In [487]: # fit the model
          model.fit(padded_docs, Datafinal['target'], epochs=30, verbose=1)
Epoch 22/30
264/264 [==============================] - 0s 1ms/step - loss: 0.0000e+00 - accuracy: 0.0049
Epoch 23/30
264/264 [==============================] - 0s 2ms/step - loss: 0.0000e+00 - accuracy: 0.0049
Epoch 24/30
264/264 [==============================] - 0s 1ms/step - loss: 0.0000e+00 - accuracy: 0.0049
Epoch 25/30
264/264 [==============================] - 0s 1ms/step - loss: 0.0000e+00 - accuracy: 0.0049
Epoch 26/30
264/264 [==============================] - 0s 1ms/step - loss: 0.0000e+00 - accuracy: 0.0049
Epoch 27/30
264/264 [==============================] - 0s 1ms/step - loss: 0.0000e+00 - accuracy: 0.0049
Epoch 28/30
264/264 [==============================] - 0s 1ms/step - loss: 0.0000e+00 - accuracy: 0.0049
Epoch 29/30
264/264 [==============================] - 0s 1ms/step - loss: 0.0000e+00 - accuracy: 0.0049
Epoch 30/30
264/264 [==============================] - 0s 1ms/step - loss: 0.0000e+00 - accuracy: 0.0049
Out[487]: <keras.callbacks.History at 0x15f0b1983a0>
```

Use of Supervised Learning Classification techniques

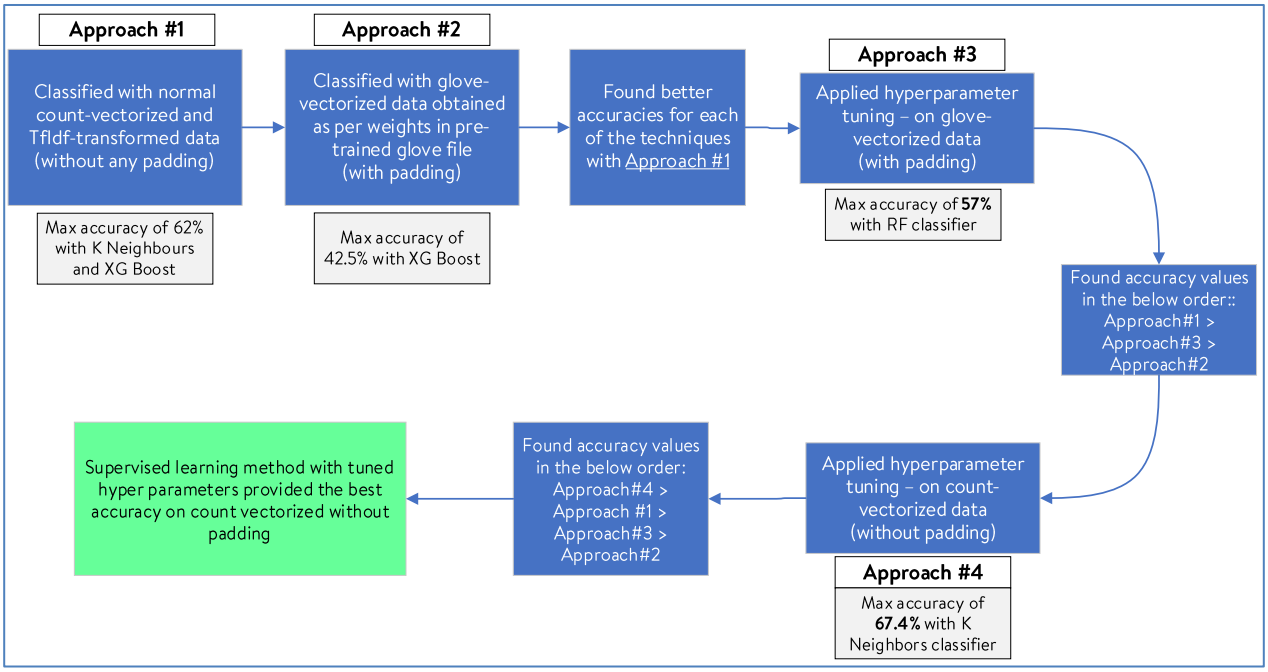
We applied 5 supervised classification techniques as below:

- Multinomial Naïve Bayes
- K Neighbors
- Decision Tree
- Random Forest
- XG Boost

Approach #1: We started with applying the 5 techniques on normal count vectorized data (without any padding) which gave the maximum accuracy as 62% through both K-Neighbors and XG Boost technique.

Approach #2: Later, we applied the same techniques on glove-vectorized data obtained through weights from pre-trained glove file.
We found better accuracies through Approach#1

Approach #3 and #4: We later applied, the techniques with hyperparameter tuning on both count vectorized and glove vectorized data as used in Approach#1 and #2



As shown in above figure, we found maximum accuracy of 47% for approach#3 obtained from Random Forest classifier.

Below are the detailed results obtained from each of the classification techniques:

Accuracies with Default hyperparameters

Techniques	**Approach#1: Count** vectorized data without padding	**Approach#2: Glove** vectorized data without padding
Multinomial NB	53%	0.03 %
K-Neighbors	62 %	41 %
Decision Tree	56 %	27 %
Random Forest	61 %	42 %
XG Boost	62 %	42.5 %

Supervised Learning Classification works better with simple count vectorized data

With **Count** vectorized data

Approach#3: With Tuned hyperparameters from GridSearch CV -

Techniques	On padded tokenized data with max length = **28** words per row (padded_docs3)	On padded tokenized data with max length = **1261** words per row (padded_docs4)	On padded tokenized data with max length = **18** words per row (padded_docs4)
K-Neighbors	-- %	-- %	-- %
Decision Tree	50 %	50 %	-- %
Random Forest	57 %	-- %	-- %
XG Boost	-- %	-- %	-- %

With **Count** vectorized data
With Tuned hyperparameters from Random Search CV
On padded tokenized data with max length = **28** words per row (padded_docs3)
-- %
48 %
56 %
-- %

Sample code snippets for classification techniques

Approach #1 – K Neighbors Classifier Technique

8.3 KNeighborsClassifier Classifier -- Using Default hyperparamters

```
nb2 = Pipeline([('vect', CountVectorizer()),
                ('tfidf', TfidfTransformer()),
                ('clf', KNeighborsClassifier()),
               ])
nb2.fit(X_tr, y_tr)
y_pred2 = nb2.predict(X_te)

predictions2 = nb2.predict_proba(X_te)

print('accuracy %s' % accuracy_score(y_pred2, y_te))
print('f1 score %s' % f1_score(y_pred2, y_te,average='weighted'))
#print ("Logloss: %0.3f " % multiclass_logloss(y_te,predictions))

print(classification_report(y_te, y_pred2))

print(confusion_matrix(y_te,y_pred2))
```

```
accuracy 0.6207100591715976
f1 score 0.6802681589950202
              precision    recall  f1-score   support

       GRP_0       0.63      0.97      0.76       750
       GRP_1       1.00      0.25      0.40         4
```

K Neighbors and XG Boost gave maximum accuracy of **62**% with count vectorized data with Default hyperparameters

Approach #2 – XG Boost Technique

11.5 XG Boost Classifier -- Using Default Hyperparameters

```
pip install xgboost
```

```
Requirement already satisfied: xgboost in c:\users\p0s041d\anaconda3\lib\site-packages (1.5.0)
Requirement already satisfied: scipy in c:\users\p0s041d\anaconda3\lib\site-packages (from xgboost) (1.5.2)
Requirement already satisfied: numpy in c:\users\p0s041d\anaconda3\lib\site-packages (from xgboost) (1.21.3)
Note: you may need to restart the kernel to use updated packages.

WARNING: You are using pip version 21.2.4; however, version 21.3.1 is available.
You should consider upgrading via the 'C:\Users\p0s041d\Anaconda3\python.exe -m pip install --upgrade pip' comman
```

```
from xgboost import XGBClassifier
nb10 = Pipeline([
                 ('clf', XGBClassifier()),
                ])
nb10.fit(X_train, y_tr)

y_pred10 = nb10.predict(X_test)

predictions = nb10.predict_proba(X_test)

print('accuracy %s' % accuracy_score(y_pred10, y_te))
print('f1 score %s' % f1_score(y_pred10, y_te,average='weighted'))

print(classification_report(y_te, y_pred10))

print(confusion_matrix(y_te,y_pred10))
```

```
[17:05:37] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/learner.cc:1115: Starting in
3.0, the default evaluation metric used with the objective 'multi:softprob' was changed from 'merror' to 'mloglos
tly set eval_metric if you'd like to restore the old behavior.
accuracy 0.4248520710059171
f1 score 0.5772223016152224
```

XG Boost gave maximum accuracy of **42.5**% with glove vectorized data with Default hyperparameters

Approach #3 – Hyperparameter tuning on glove vectorized data

11.4 Random Forest Classifier -- -- Using Default Hyperparameters

```
from sklearn.ensemble import RandomForestClassifier
nb9 = Pipeline([
                ('clf', RandomForestClassifier()),
               ])
nb9.fit(X_train, y_tr)

y_pred9 = nb9.predict(X_test)

predictions = nb9.predict_proba(X_test)

print('accuracy %s' % accuracy_score(y_pred9, y_te))
print('f1 score %s' % f1_score(y_pred9, y_te,average='weighted'))

print(classification_report(y_te, y_pred9))

print(confusion_matrix(y_te,y_pred9))
```

```
accuracy 0.4236686390532544
f1 score 0.5757931811777965
```

RF classifier gave maximum accuracy of **42.3**% with glove vectorized data with Default hyperparameters

Approach #4 – Hyperparameter tuning on count vectorized data without padding

12.3: Applying the tuning for KNN

```
parameters_NNH=dict(n_neighbors=range(1,10),leaf_size=range(10,50), weights = ['uniform', 'distance'])
NNH = KNeighborsClassifier()
clf_NNH = GridSearchCV(NNH, parameters_NNH, cv=5)
clf_NNH.fit(X_tr_countvec_wo_padd, y_tr_countvec_wo_padd)
scores = clf_NNH.cv_results_['mean_test_score']
nnh_comb=[]
score_list=[]
[ nnh_comb.append([i,j]) for i in parameters_NNH['n_neighbors'] for j in parameters_NNH['leaf_size'] ]
for score, n_nnh in zip(scores, np.array(nnh_comb)):
    score_list.append([n_nnh,score])
    print(n_nnh,":",score)
```

K Neighbors classifier gave maximum accuracy of **67.4**% with count vectorized data with tuned hyperparameters

```
acc_df
```

	Model_Name	acc_score	f1_score
0	MultinomialNB	0.667192	0.714364
1	KNN	0.674290	0.736438

As observed the results for all the supervised classification techniques, we found K-Neighbors giving maximum accuracy of **67.4** % with hyperparameter tuning.

LSTM Technique

We applied LSTM technique to check whether the tickets classification would happen better by capturing the context of the sentences

Model Architecture

We built the model with 3 layers:

a) **Input layer**: Made from word embeddings generated with weights defined as per pre-trained glove text file: glove.6B.100d.txt
b) **Intermediate hidden layer:** Long Short-Term Memory layer (LSTM) with 100 inputs. Activation function being tanh and recurrent activation method being sigmoid function
c) **Output layer:** A dense layer with 74 output channel to classify the ticket into one of the groups

We applied the technique on count-vectorized data after padding it with length of **4 words, 28 words and 1261 words** (maximum length among the row entries obtained from tickets' description. Below are the results obtained:

Count vectorized data	
LSTM model based classifiers	
On padded tokenized data with max length = **28** words per row	57 %
On padded tokenized data with max length = **1261** words per row	47 %

LSTM technique gave maximum accuracy with count vectorized data padded for maximum length of 28 words per row of **57%**

Code snippets

Embedding layer creation

```
# The maximum number of words to be used. (most frequent)
MAX_NB_WORDS = 50000
# Max number of words in each complaint.
MAX_SEQUENCE_LENGTH = 250
# This is fixed.
EMBEDDING_DIM = 100
tokenizer = Tokenizer(num_words=MAX_NB_WORDS, filters='!"#$%&()*+,-./:;<=>?@[\]^_`{|}~', lower=True)
tokenizer.fit_on_texts(Datafinal['Assignment group'].values)
word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))
```

Model fitting

10.6. Building the LSTM model

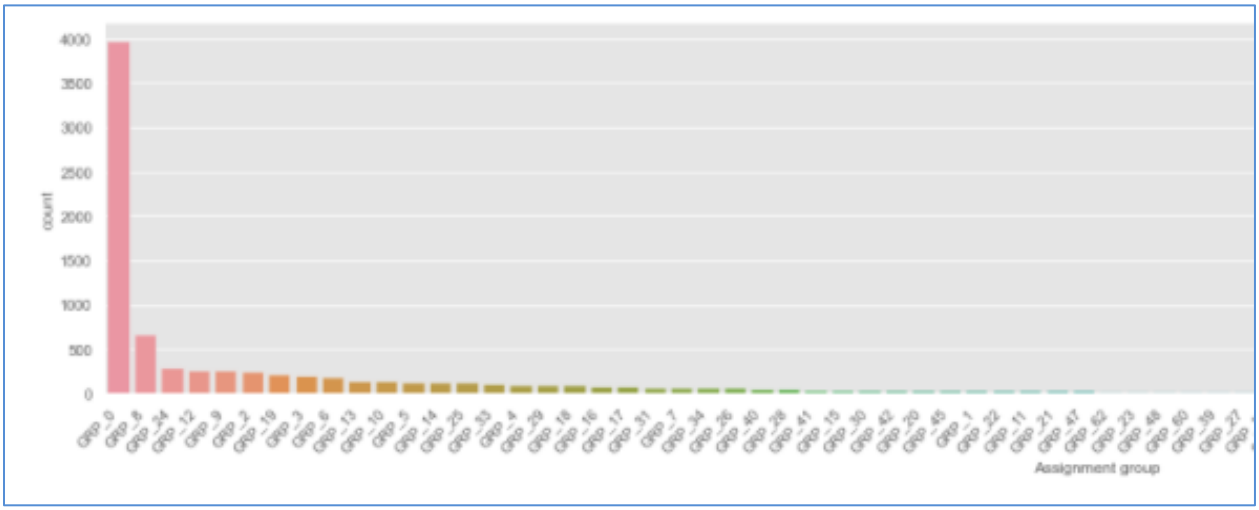
```
from keras.layers import Input, Dropout, Flatten, Dense, Embedding, LSTM, GRU
from keras.callbacks import EarlyStopping, ModelCheckpoint
model = Sequential()
model.add(Embedding(MAX_NB_WORDS, EMBEDDING_DIM, input_length=X.shape[1]))
model.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(74, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

epochs = 30 # Keeping the epochs count as 5 as accuracy starts decreasign after 5 epochs
batch_size = 64

history = model.fit(X_train, Y_train, epochs=epochs, batch_size=batch_size,validation_split=0.1,callbacks=[EarlyStopping(monitor
```

Applying conditional step-by-step classifiers to handle data imbalance

As observed before, we achieved a maximum accuracy of 67% with supervised learning techniques with tuned hyperparameters.
Such accuracy may not be helpful in the industry. Hence, we should first handle the data imbalance what we observed before. Putting the earlier observed histogram again for reference:



Classifier-1 – To classify whether GRP_0 or any other group

We created one more column in data frame to store “GRP_0_vs_others” classification

```
def GRP_0_vs_others (row):
    if row['Assignment group']!= 'GRP_0':
        return 'Others'
    return 'GRP_0'
```

```
Datafinal.apply(lambda row: GRP_0_vs_others(row), axis=1)
```

```
0        GRP_0
1        GRP_0
2        GRP_0
3        GRP_0
4        GRP_0
         ...
8443    Others
8444     GRP_0
8445     GRP_0
8446    Others
8447    Others
Length: 8448, dtype: object
```

Building Classifier-1

We used count vectorized data and chose XG Boost classifier (with default hyperparameters) as classifier-1, as it gave the maximum accuracy of 83%

13.4.3. Creating the XG Boost classifier

```
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, f1_score, classification_report

XGB2 = XGBClassifier()
XGB2.fit(X_tr_step_1, y_tr_step_1)
y_pred = XGB2.predict(X_test_step_1)

predictions2 = XGB2.predict_proba(X_test_step_1)

print('accuracy %s' % accuracy_score(y_pred, y_test_step_1))
print('f1 score %s' % f1_score(y_pred, y_test_step_1,average='weighted'))
#print ("logloss: %0.3f " % multiclass_logloss(y_te,predictions))

print(classification_report(y_test_step_1, y_pred))

print(confusion_matrix(y_test_step_1, y_pred))
```

```
accuracy 0.8421052631578947
f1 score 0.8428599590434046
```

Classifier-2 – To classify tickets for rest of the groups except GRP_0

We created another dataframe “DatafinalOthers” to keep all the tickets except GRP_0

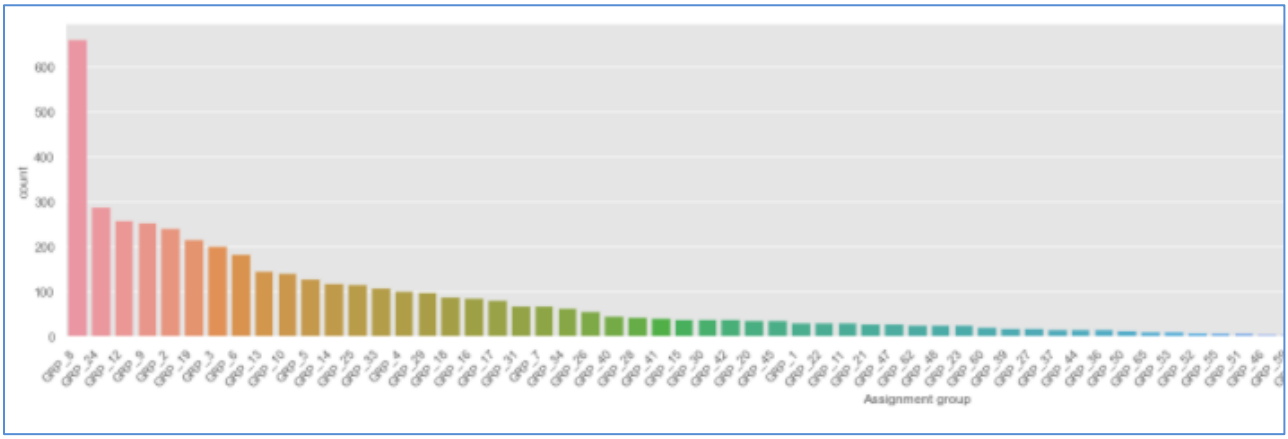
```
#Create Dataset for 'others' i.e all groups which is not part of GRP_0
DatafinalOthers= Datafinal[Datafinal['Assignment group'] != 'GRP_0']
```

```
DatafinalOthers
```

	Short description	Description	Assignment group
6	event critical hostname company com the value ...	event critical hostname company com the value ...	GRP_1
17	when undocking pc screen will not come back	when undocking pc screen will not come back	GRP_3
32	duplication of network address	gentles have two devices that are trying to sh...	GRP_4
43	please reroute jobs on printer to printer issu...	hi the printer printer is not working and need...	GRP_5
47	job job failed in job scheduler at	job job failed in job scheduler at	GRP_6
...	...	...	...
8431	erp fi ob two accounts to be added	i am sorry have another two accounts that need...	GRP_10
8432	tablet needs reimaged due to multiple	tablet needs reimaged due to multiple	GRP_3

Handling imbalance in the dataset for Classifier-2

As we saw before, there was imbalance in the data even after removal of GRP_0:



To handle the imbalance, we re-sampled the data using resample function as below:

13.5.2. Resampling the data ¶

```
maxOthers = DatafinalOthers['Assignment group'].value_counts().max()
maxOthers
```

```
661
```

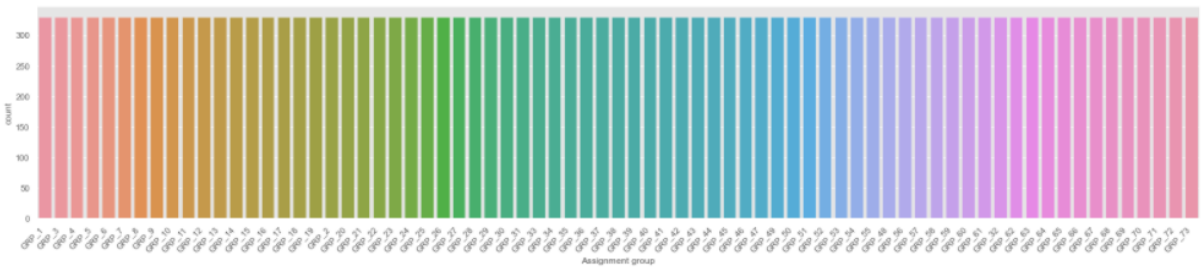
```
from sklearn.utils import resample
DatafinalOthers_resampled = DatafinalOthers[0:0]
for grp in DatafinalOthers['Assignment group'].unique():
    temp = DatafinalOthers[DatafinalOthers['Assignment group'] == grp]
    resampled = resample(temp, replace=True, n_samples=int(maxOthers/2), random_state=123)
    DatafinalOthers_resampled = DatafinalOthers_resampled.append(resampled)
```

On re-checking the data after re-sampling, we found balanced dataset as below:

13.5.3. Checking the distribution after resampling

```
plt.style.use('ggplot') # Using gg plot for plotting based on descending order
%matplotlib inline

descending_order = DatafinalOthers_resampled['Assignment group'].value_counts().sort_values(ascending=False).index
plt.subplots(figsize=(22,5))
#added code for x label rotate
ay=sns.countplot(x='Assignment group', data=DatafinalOthers_resampled,order=descending_order)
ay.set_xticklabels(ay.get_xticklabels(), rotation=45, ha="right")
plt.tight_layout()
plt.show()
```



We can see above that all the groups are distributed uniformly now

Building Classifier-2 on re-sampled data

We used count vectorized data and chose KNN classifier (with tuned hyperparameters) as classifier-2, which gave the maximum accuracy of 93%

13.5.6.1. Applying the tuned hyperparameters as obtained in section 12.3 for KNN classifier

```
KN3_final = KNeighborsClassifier(n_neighbors=7,leaf_size=10, weights = 'distance')
KN3_final.fit(X_tr_step_2, y_tr_step_2)
```

```
KNeighborsClassifier(leaf_size=10, n_neighbors=7, weights='distance')
```

```
from sklearn import metrics
print(KN3_final.score(X_tr_step_2, y_tr_step_2))
y_pred = KN3_final.predict(X_test_step_2)
#acc_df=pd.DataFrame(columns=['Model_Name','acc_score','f1_score'])
#acc_df=acc_df.append({'Model_Name':'KNN_with_resampled_data','acc_score':accuracy_sc
cm=metrics.confusion_matrix(y_test_step_2, y_pred)

#check_Acc(NNH_final)
```

```
0.9547274858370776
```

```
print('accuracy %s' % accuracy_score(y_pred, y_test_step_2))
print('f1 score %s' % f1_score(y_pred, y_test_step_2,average='weighted'))
```

```
accuracy 0.9322080796900941
f1 score 0.9293711184351205
```

Custom function for condition-based classification

We applied custom function “condition_based_classifier” with above 2 classifiers.

- First classifier determines whether GRP_0 or "Others".
- Second classifier executes only when first classifier results into "Others" and then finally classifies it to any of the other assignment groups (GRP_1, GRP_2 etc.)

13.6. Applying a function to apply the classifier 2 conditionally based on result of classifier 1

```
def condition_based_classifier(X):
    # Step:1 - Apply Classifier1 on X and find Y
    y_predicted = []
    y_predicted = XGB2.predict(X) # Applying classifier 1 overall
    for i in range(len(y_predicted)):
        if y_predicted[i] != 'GRP_0':
    # Step:2 - For all X which are classified as "Others", apply Classifier2
            y_predicted[i]=KN3_final.predict(X[i])[0] # Applying classifier 2 wherev
    # Step:3 - Finally generate the classification results
    return y_predicted
```

```
y_pred2=[]
y_pred2=condition_based_classifier(X_test_final)
```

Overall accuracy with condition-based custom classifier

We found the overall accuracy of 77% with the custom condition-based classifier

```
y_test_final = np.array(y_test_final) # Converting y_test_final into array
```

```
print('accuracy %s' % accuracy_score(y_pred2, y_test_final))  # Comparing
print('f1 score %s' % f1_score(y_pred2, y_test_final,average='weighted'))
```

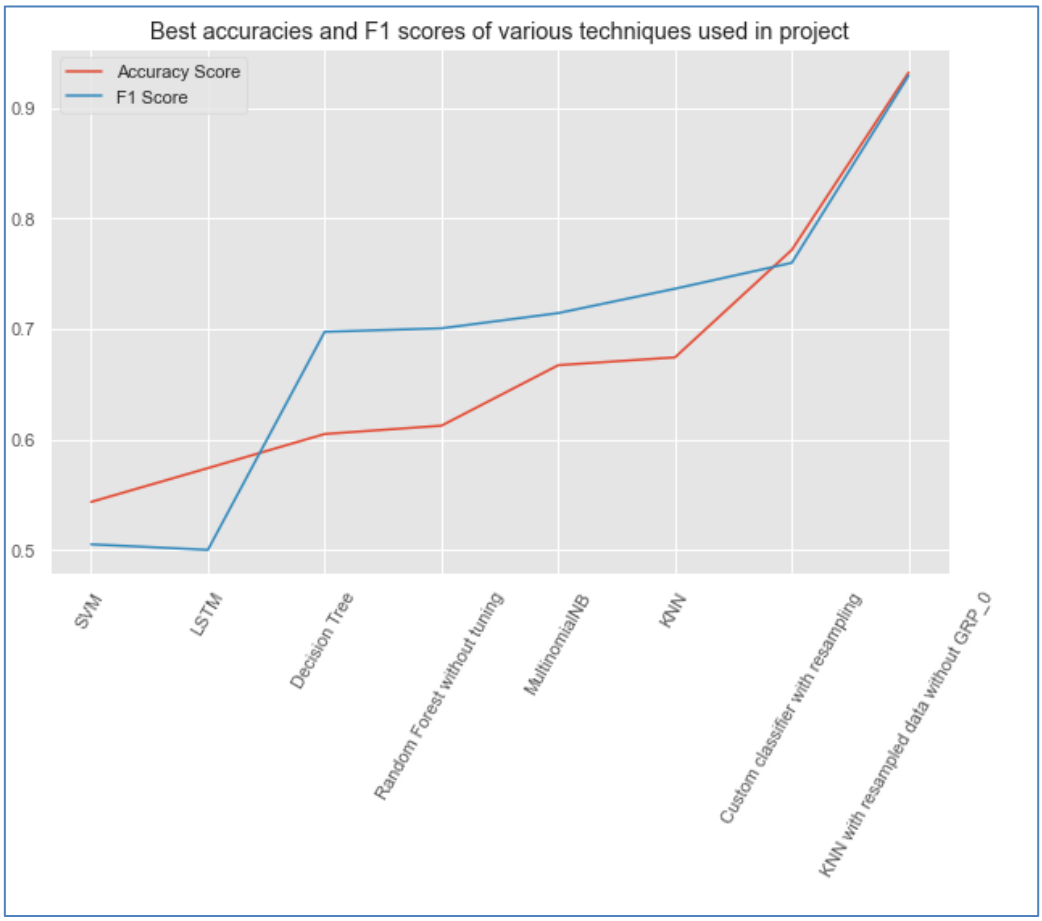
```
accuracy 0.7715976331360946
f1 score 0.7598849475279427
```

Comparison of accuracies from various techniques

Comparison Results and classifier with the best result

The maximum accuracy was found with Classifier-2 (K Neighbors for classification of tickets except GRP_0) of 93%. The second-best accuracy was obtained with custom classifier comprising of classifier-1 (XG Boost for classification of tickets into GRP_0 vs "Others") and classifier-2

Below is graph for comparison of accuracies and F-1 scores



Conclusion

Project Recommendations

As clearly visible GRP_0 is the cause of lesser accuracy. GRP_0 seems to be like a miscellaneous group/ triage group which receives almost every category of tickets. In a practical scenario, the company would have created this group to assign all the tickets which lack some fields to determine the assignment group of the ticket. So, all those "un-identified" tickets might get assigned to GRP_0 which could have a good manpower to handle the issues manually.

We recommend the use of either condition-based classifier or classifier-2 in isolation. Below are the 2 options for implementation for the incident's classification in the company:

- **Option-1 (77% accuracy)**: Fully automated classification: This would comprise of condition-based classifier to handle all the tickets (including GRP_0). However, the company would need to compromise on the accuracy with existing solution
- **Option-2 (93% accuracy)**: Hybrid solution: This would comprise of just Classifier-2. The overall solution would require some manual handling of tickets with lesser probabilities of classification (mostly GRP_0) and automated classification of rest of the tickets using classifier-2

Future Scope of existing solution improvement

We recommend finding ways for further logically grouping GRP_0 tickets based on business domain knowledge (for e.g. GRP_0_A, GRP_0_B, GRP_0_c etc.). This could be thought by first finding possibility of clusters within GRP_0 tickets. If there are clear clusters visible, IT team needs to determine the best way to classify them.

After doing further classification of GRP_0 tickets, the company can achieve a fully automated tickets classifier with better accuracy than current combined custom classifier with 77% accuracy

Insights

- The Supervised Learning classifiers gave best results on count vectorized data rather than Glove vectorized data. Hence, context-based word vectorization technique are recommended only for Neural network models and not supervised learning classifiers
- To handle imbalanced data as given in the project:
    - We should first check the existence of any outlier in the data. In case of outliers, we should build a separate classifier for classifying between the outliers and the common categories
    - For rest common categories, if there is still imbalance, we should re-sample the data before applying the classification techniques
    - This would result in a condition-based classification to first identify whether data belong to an outlier group and then do the second classification only if the data doesn't belong to the outlier group
- Padding of data is not recommended for Supervised Learning classifiers unlike what we do usually in neural network modelling. Its better to apply feature reduction techniques like PCA while applying supervised classification.
- NLP models-based classification and LSTM models might not work best with skewed data as provided to us (GRP_0 being the outlier in the dataset) where supervised learning techniques work better.

- In case the input dataset has varied row lengths, LSTM model gives good results with optimum padding of data which should be close to the median value of row lengths. Since, in the given case, median length was 18, we applied a close padding of 28 words

Data Quality and Recommendations

- The data had values which were masked and hence weren't conveying any meaning for e.g. "xd'", "zz" etc.
- Data quality could have been better. Given data had lot of junk characters like: mail sub headers ("received from:", "from:", "to", "cc:", "https:")
- In an ideal solution, the user interface meant for raising tickets should be build such that it should accept only valid text values. Good quality data received through such a solution should eliminate the need of cleansing the data and should eventually improve the results

To address these shortcomings, we had to define custom function to selectively remove these junk values from the dataset

Data Veracity

Given data looked genuine with "Caller IDs" provided for each ticket. In industry practice, if there is a possibility of malicious tickets, we can build a solution to first verify the registered Caller IDs and then process the concerned tickets

Implementation Suggestions

The proposed classifier should be kept updated with by periodically monitoring the performance/ accuracy and accordingly refitting the models with updated data.

----------End of the report ----------