Automatic Ticket Assignment

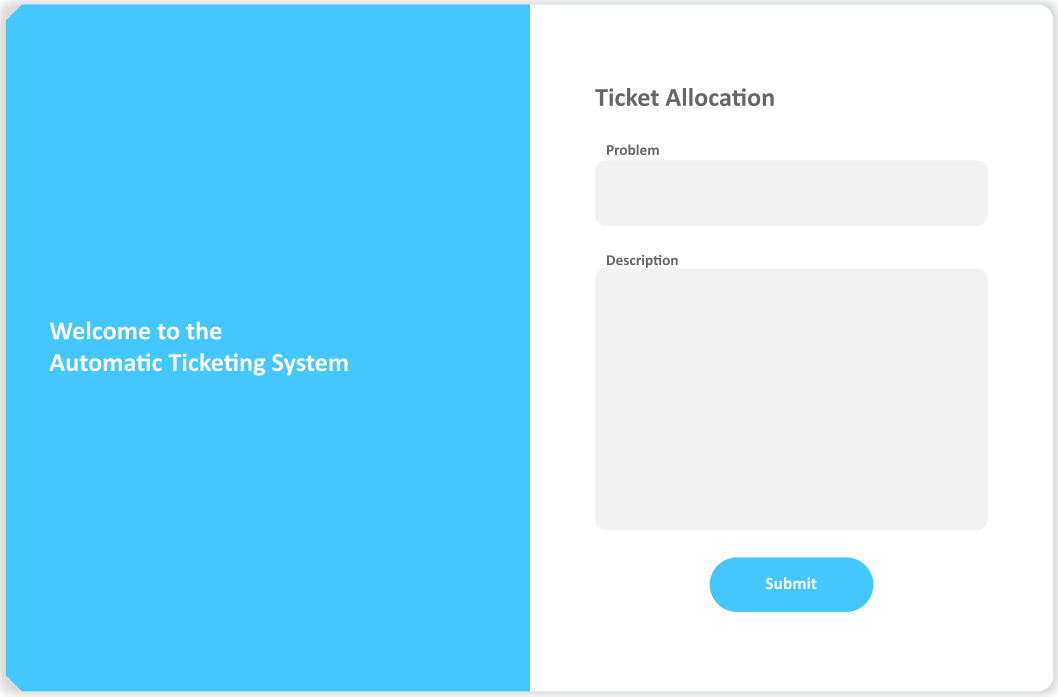
Capstone Project G10 - NLP1 Mar 21A

(Final Report)

Team

Lakshmi R Pillai
Rajesh D
Sarrah Sadikot
Sankarananda R
Tejas Ramachandharan

Automatic Ticketing System



INDEX

#	Topic	Page
1	Summary of problem statement, data and findings	1
2	Overview of the final process	5
3	Step-by-step walk through the solution	13
4	Model evaluation	29
5	Comparison to bench mark	50
6	Visualization	52
7	User Interface Development	63
8	Implication	70
9	Limitation	71
10	Closing Reflections	71

1. Summary of problem statement, data and findings

Problem:

Providing uninterrupted good service to the customer is the basic need of every successful business. That is one of the reasons why most businesses try to provide 24x7 customer interaction and support using various IT tools. A good Incident Management team handles unplanned downtimes and service quality issues with fast solutions in minimum time. This is usually achieved with ticketing systems created for Business / IT Users and End Users/ Vendors where tickets are raised for concerns and issues, sometimes integrating monitoring systems and tools to it. When an issue or support ticket appears on the help desk, it needs to be processed and assigned a tag (or category) so that it's routed to the correct team member. With time as the business grows the scale of issues and tickets also grows. With time as the business grows the scale of issues and tickets also grows. Manual classification of IT service desk tickets may result in the assignment of the tickets to the wrong resolution group. It is a time-consuming process and requires great human effort as incoming incidents are analyzed and assessed by the support teams to classify by right categorization, priorities, and take action on the requests. Human involvement makes it prone to human error and dependencies. Incorrect routing of IT service desk tickets leads to the reassignment of tickets, unnecessary resource utilization, delays the resolution time, and increase costs. It can cause user dissatisfaction, poor customer service, resulting in customer loss and affecting the business profit and value.

Traditional Incident Management Method:

- The process of ticket handling followed by most organizations is such a way that incidents, 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).
- Support teams analyze them to classify by right categorization, priorities and take action on the requests.
- About half of the incidents (~54%) that are of lower priorities and ease to work on with quick fixes are resolved by L1 / L2 teams.
- Higher priority issues and those requiring demanding detailed diagnosis and those unable to manage by L1/L2 are escalated to Functional teams from Applications and Infrastructure (L3 teams). About ~56% of incidents are resolved by Functional / L3 teams.
- Some incidents may require Vendor involvement too.
- Standard Operating Procedures (SOPs) are to be reviewed and followed for higher priority incidents that require at least 15 minutes per incident. It is a hugely time-consuming process that requires dedicated personnel

Issues with the Traditional system:

- Manual classification systems are often complicated and cluttered, and support agents struggle to assign a category. After spending endless hours going through tickets, they'll often end up assigning the 'Other' tag to get through this tedious task faster.
- Assignment to the wrong functional group/team. Around ~25% of Incidents are wrongly assigned to functional teams.
- Long processing time.
- Issues not addressed at the right time.

Consequences:

- Poor customer service.
- Unsatisfied customers.
- Loss of customers.

Loss of business and profit `Solution:

- Guided by powerful AI techniques that can classify incidents to right functional groups can help organizations reduce the issue's resolving time and focus on more productive tasks.
- Advanced AI tools can learn, adapt, and determine actions to take, without human input.

Benefits:

- Automatic ticket classification with machine learning uses natural language processing (NLP), which helps machines process, understand, and generate human language quickly, consistently, and cost-effectively.
- Correct ticket classification helps businesses sort their unstructured data, which provides many more valuable insights than structured data and utilize that for business growth.
- As the business expands, customer queries, issues, feedback, requests also increase. Using AI techniques makes it scalable, it's also fast and highly accurate if models have been trained correctly. It can sort millions of pieces of data at a fraction of the cost of manual tagging, save time, and avoid burdening teams with tedious and repetitive tasks.

- It can work around the clock, so can respond immediately.
- Consistent criteria- Ticket classification with machine learning applies the same criteria to measure each set of data, plus a machine will never be subjective lack alertness, and rush through tickets without understanding them properly.
- It can be easily integrated with other AI tools, including chatbots and automatic data collection and reporting thus helping in extracting more insights

Objective:

The objective is to build a classifier that can classify the tickets by analyzing the text.

Data Given:

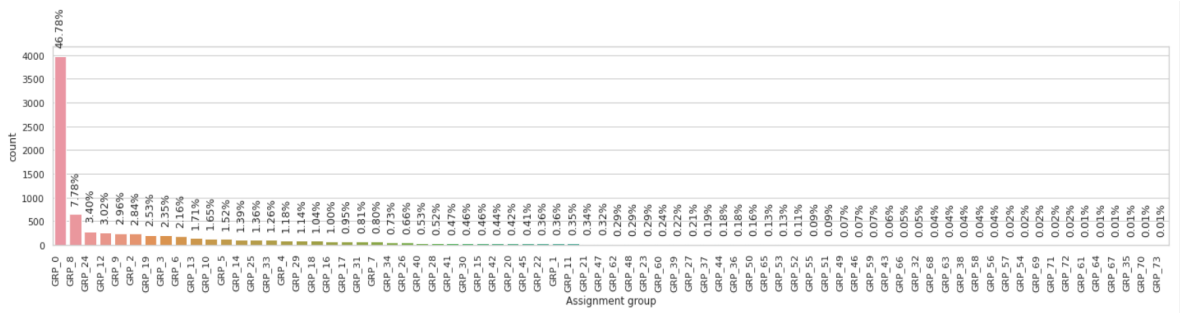
	Short description	Description	Caller	Assignment group
7526	not possible to login due to a locked account	not possible to login due to a locked account	qufjnslk bvtfrwnu	GRP_0
3709	do not have permission to ess portal. at	please reach out to me for more information if...	ytwmgpbk cpawsihk	GRP_2
3237	erp accout had been locked	_x000D_\n_x000D_\nreceived from: qyidkvap.cxnf...	qyidkvap cxnfdjpk	GRP_0
7956	unable to login to vpn and reset password	unable to login to vpn and reset password	xiaurwpz hzusljmc	GRP_0
5751	when i create a recurring appointment in mds i...	when i create a recurring appointment in mds i...	acteiqdu bferalus	GRP_40
5050	unable to launch outlook	unable to launch outlook	vjwdyanl knfsjdgz	GRP_0

The Data given is an excel file having the following columns. Short description

a. Description
b. Caller
c. Assignment group

Out of the above five columns “Caller” is not useful for our purpose. There are few missing values also present in the document. Also there seems to be some non-English words also present. So we need to do a preprocessing of the data before further processing and model building.

The distribution of raw data, without any preprocessing is as below.



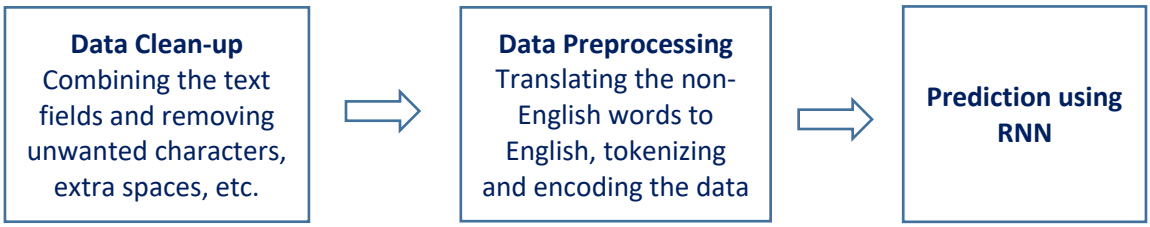
Findings and implications:

There is a huge imbalance of the data. There are a lot of Assignment Groups with <100 number of tickets so for better visualization and further processing we can group them to one. Also it will be better to merge the Description and Short description columns.

Guided by powerful AI techniques that can classify incidents to right functional groups can help organizations reduce the issue’s resolving time and focus on more productive tasks. Advanced AI tools can learn, adapt, and determine actions to take, without human input.

2. Overview of the final process

We used RNN based algorithm for assigning the IT Tickets to the relevant groups. The overall process can be represented as below:



RNN based final model

Merging the "Short description" and "Description" and removing unnecessary Columns

Created a new column "Text" by merging the "Short description" and "Description" columns; and then removing the unwanted columns - "Short description" , "Description" and "Caller"

```
dataset["Text"] = dataset["Short description"] + " " + dataset["Description"]

dataset.drop(["Short description","Description","Caller"],axis=1,inplace=True)

dataset
```

	Assignment group	Text
0	GRP_0	login issue -verified user details.(employee# ...
1	GRP_0	outlook _x000D_\n_x000D_\nreceived from: hmjdr...
2	GRP_0	cant log in to vpn _x000D_\n_x000D_\nreceived ...
3	GRP_0	unable to access hr_tool page unable to access...
4	GRP_0	skype error skype error

Removing Unwanted Characters

We tried different methods to remove the unwanted characters but the below method resulted in higher accuracy of the final model.

```
def removeString(data, regex):
    return data.str.lower().str.replace(regex.lower(), ' ')


def cleanDataset(dataset, columnsToClean, regexList):
    for column in columnsToClean:
        for regex in regexList:
            dataset[column] = removeString(dataset[column], regex)
    return dataset


def getRegexList():
    '''
    Adding regex list as per the given data set to flush off the unnecessary text

    '''
    regexList = []
    regexList += ['From:(.*)\r\n']  # from line
    regexList += ['Sent:(.*)\r\n']  # sent to line
    regexList += ['received from:(.*)\r\n']  # received data line
    regexList += ['received']  # received data line
    regexList += ['To:(.*)\r\n']  # to line
    regexList += ['CC:(.*)\r\n']  # cc line
    regexList += ['(.*)infection']  # footer
    regexList += ['\[cid:(.*)]']  # images cid
    regexList += ['https?:[^\]\n\r]+']  # https & http
    regexList += ['Subject:']
    regexList += ['[\w\d\-\_\.]+@[\w\d\-\_\.]+']  # emails are not required
    regexList += ['[0-9][\-0-90-9 ]+']  # phones are not required
    regexList += ['[0-9]']  # numbers not needed
    regexList += ['[^a-zA-z 0-9]+']  # anything that is not a letter
    regexList += ['[\r\n]']  # \r\n

    regexList += ['^[_a-z0-9-]+(\.[_a-z0-9-]+)*@[a-z0-9-]+(\.[a-z0-9-]+)*(\.[a-z]{2,4})$']
    regexList += ['[\w\d\-\_\.]+ @ [\w\d\-\_\.]+']
    regexList += ['Subject:']
    regexList += ['[^a-zA-Z]']

    regexList += [' [a-zA-Z] ']  # single letters makes no sense
    regexList += [' [a-zA-Z][a-zA-Z] ']  # two-letter words makes no sense
    regexList += ["  "]  # double spaces

    return regexList
```

```
columnsToClean = ['Text']

clean_dataset = cleanDataset(dataset, columnsToClean, getRegexList())

clean_dataset
```

	Assignment group	Text
0	GRP_0	login issue verified user details employee ma...
1	GRP_0	outlook d d from d hello team d d meetings s...
2	GRP_0	cant log to vpn d d from d d d cannot log to...
3	GRP_0	unable access tool page unable access tool page
4	GRP_0	skype error skype error

Language Translation to English

We used langdetect to identify the language and GoogleTranslate to translate non-English rows to English.

```
[ ]  def fn_lan_detect(df):
         try:
             return detect(df)
         except:
             return 'no'

     clean_dataset['language'] =clean_dataset['Text'].apply(fn_lan_detect)

     clean_dataset["language"].value_counts()
```

```
en    6169
sl     444
de     393
fr     318
af     305
da     162
no     155
sv     128
ca     104
it      72
```

```
[ ]  total = len(clean_dataset)

     for i, row in clean_dataset.iterrows():

       if (row["language"] != "en"):

         try :
           clean_dataset["Text"][i] = GoogleTranslator(source='auto', target='en').translate(row['Text'])
         except :
           clean_dataset["Text"][i]  = row["Text"]
       print(f"entrys completed : {i}/{total}",end="\r")
```

```
[ ]  clean_dataset.drop("language",axis=1,inplace=True)
     clean_dataset
```

	Assignment group	Text
0	GRP_0	login issue verified user details employee ma...
1	GRP_0	outlook d d from d hello team d d meetings s...
2	GRP_0	cant log to vpn d d from d d d cannot log to...
3	GRP_0	unable access tool page unable access tool page
4	GRP_0	skype error skype error

Grouping the classes

We grouped all the groups having <100 counts to a common group called “GRP_X”.

```
[ ] clean_dataset.drop_duplicates(inplace = True)
    clean_dataset.fillna('', inplace=True)
```

```
[ ] counts = clean_dataset["Assignment group"].value_counts()

    clean_dataset["Assignment group"] = np.where(counts[clean_dataset["Assignment group"]] < 100 , "GRP_X", clean_dataset["Assignment group"])

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

```
GRP_0     3198
GRP_X     1688
GRP_8      329
GRP_24     277
GRP_12     245
```

Oversampling

There is a huge imbalance in the data, so to reduce its effect, we did oversampling. Using resampling, we upsampled the data to match with the count of the most frequent group.

```
[ ] maxOthers =dataset["Assignment group"].value_counts().max()

    dataset_resampled = dataset[0:0]
    for grp in dataset['Assignment group'].unique():
        ticket_dataset_group = dataset[dataset['Assignment group'] == grp]
        resampled = resample(ticket_dataset_group, replace=True, n_samples=int(maxOthers), random_state=123)
        dataset_resampled = dataset_resampled.append(resampled)


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

```
GRP_0     3198
GRP_X     3198
GRP_3     3198
GRP_8     3198
GRP_12    3198
```

Encoding the Sentences and Labels

We used keras Tokenizer to tokenize the rows and Label Encoder to encode the groups.

```
[ ] maxlen = 300
    numWords=9000
    epochs = 10
    batch_size = 100
```

```
[ ] tokenizer = Tokenizer(num_words=numWords, oov_token="<unk>", filters='!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n')
    tokenizer.fit_on_texts(dataset_resampled['Text'])

    train_seqs = tokenizer.texts_to_sequences(dataset_resampled['Text'])
```

We padded the sequences to a length of 300

```
encoder = LabelEncoder()

X = pad_sequences(train_seqs, padding='post',maxlen=maxlen)

Y = dataset_resampled["Assignment group"].astype("category")
Y = encoder.fit_transform(Y)
```

Test Train Split

We did Test Train Split with test_size=0.20 and random_state =42

```
x_train, x_test, y_train, y_test = train_test_split(X,Y, test_size = 0.20, random_state = 42)
print(x_train.shape, y_train.shape )
print(x_test.shape,  y_test.shape  )
```

```
(30700, 300) (30700,)
(7676, 300) (7676,)
```

Model building

Final model we used is below

```
glove_file = '/content/drive/MyDrive/Colab Notebooks/capstone project/glove.6B.50d.txt'

embeddings_glove = {}
for o in open(glove_file):
    word = o.split(" ")[0]

    embd = o.split(" ")[1:]
    embd = np.asarray(embd, dtype='float32')

    embeddings_glove[word] = embd
```

```
embedding_matrix = np.zeros((numWords+1, 50))

for i,word in tokenizer.index_word.items():
  if i<numWords+1:
    embedding_vector = embeddings_glove.get(word)
    if embedding_vector is not None:
      embedding_matrix[i] = embedding_vector
```

```
embed = Embedding(numWords+1,output_dim=50,input_length=maxlen,weights=[embedding_matrix], trainable=True)

model=Sequential()
model.add(Input(shape=(maxlen,),dtype=tf.int64))
model.add(embed)
model.add(Conv1D(100,10,activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(Dropout(0.3))
model.add(Conv1D(100,10,activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(Bidirectional(LSTM(128)))
model.add(Dropout(0.3))
model.add(Dense(100,activation='relu'))
model.add(Dense(len((pd.Series(y_train)).unique()),activation='softmax'))
model.compile(loss='sparse_categorical_crossentropy',optimizer="adam",metrics=['accuracy'])

model.summary()


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

model_history = model.fit(x_train,y_train,batch_size=batch_size, epochs=epochs, callbacks=[checkpoint,reduceLoss], validation_data=(x_test, y_test))
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 300, 50)           450050

 conv1d (Conv1D)             (None, 291, 100)          50100

 max_pooling1d (MaxPooling1D  (None, 145, 100)         0
 )

 dropout (Dropout)           (None, 145, 100)          0

 conv1d_1 (Conv1D)           (None, 136, 100)          100100

 max_pooling1d_1 (MaxPooling  (None, 68, 100)          0
 1D)

 bidirectional (Bidirectiona  (None, 256)              234496
 l)

 dropout_1 (Dropout)         (None, 256)               0

 dense (Dense)               (None, 100)               25700

 dense_1 (Dense)             (None, 12)                1212

=================================================================
Total params: 861,658
Trainable params: 861,658
Non-trainable params: 0
```

```
Epoch 1/10
307/307 [==============================] - ETA: 0s - loss: 1.3357 - accuracy: 0.5484
Epoch 1: val_accuracy improved from -inf to 0.83872, saving model to model-001-0.838718.h5
307/307 [==============================] - 34s 69ms/step - loss: 1.3357 - accuracy: 0.5484 - val_loss: 0.4949 - val_accuracy: 0.8387 - lr: 0.0010
Epoch 2/10
307/307 [==============================] - ETA: 0s - loss: 0.3712 - accuracy: 0.8747
Epoch 2: val_accuracy improved from 0.83872 to 0.92092, saving model to model-002-0.920922.h5
307/307 [==============================] - 20s 65ms/step - loss: 0.3712 - accuracy: 0.8747 - val_loss: 0.2309 - val_accuracy: 0.9209 - lr: 0.0010
Epoch 3/10
307/307 [==============================] - ETA: 0s - loss: 0.2039 - accuracy: 0.9301
Epoch 3: val_accuracy improved from 0.92092 to 0.94203, saving model to model-003-0.942027.h5
307/307 [==============================] - 20s 65ms/step - loss: 0.2039 - accuracy: 0.9301 - val_loss: 0.1781 - val_accuracy: 0.9420 - lr: 0.0010
Epoch 4/10
307/307 [==============================] - ETA: 0s - loss: 0.1409 - accuracy: 0.9508
Epoch 4: val_accuracy improved from 0.94203 to 0.95922, saving model to model-004-0.959224.h5
307/307 [==============================] - 20s 65ms/step - loss: 0.1409 - accuracy: 0.9508 - val_loss: 0.1144 - val_accuracy: 0.9592 - lr: 0.0010
Epoch 5/10
307/307 [==============================] - ETA: 0s - loss: 0.1063 - accuracy: 0.9633
Epoch 5: val_accuracy improved from 0.95922 to 0.95935, saving model to model-005-0.959354.h5
307/307 [==============================] - 20s 65ms/step - loss: 0.1063 - accuracy: 0.9633 - val_loss: 0.1243 - val_accuracy: 0.9594 - lr: 0.0010
Epoch 6/10
307/307 [==============================] - ETA: 0s - loss: 0.0855 - accuracy: 0.9706
Epoch 6: val_accuracy improved from 0.95935 to 0.96704, saving model to model-006-0.967040.h5
307/307 [==============================] - 20s 65ms/step - loss: 0.0855 - accuracy: 0.9706 - val_loss: 0.1045 - val_accuracy: 0.9670 - lr: 0.0010
Epoch 7/10
307/307 [==============================] - ETA: 0s - loss: 0.0673 - accuracy: 0.9778
Epoch 7: val_accuracy improved from 0.96704 to 0.96756, saving model to model-007-0.967561.h5
307/307 [==============================] - 20s 65ms/step - loss: 0.0673 - accuracy: 0.9778 - val_loss: 0.1066 - val_accuracy: 0.9676 - lr: 0.0010
Epoch 8/10
307/307 [==============================] - ETA: 0s - loss: 0.0655 - accuracy: 0.9777
Epoch 8: val_accuracy improved from 0.96756 to 0.97199, saving model to model-008-0.971991.h5
307/307 [==============================] - 20s 65ms/step - loss: 0.0655 - accuracy: 0.9777 - val_loss: 0.0882 - val_accuracy: 0.9720 - lr: 0.0010
Epoch 9/10
307/307 [==============================] - ETA: 0s - loss: 0.0530 - accuracy: 0.9821
Epoch 9: val_accuracy improved from 0.97199 to 0.97551, saving model to model-009-0.975508.h5
307/307 [==============================] - 20s 65ms/step - loss: 0.0530 - accuracy: 0.9821 - val_loss: 0.0856 - val_accuracy: 0.9755 - lr: 0.0010
Epoch 10/10
307/307 [==============================] - ETA: 0s - loss: 0.0528 - accuracy: 0.9819
Epoch 10: val_accuracy improved from 0.97551 to 0.97681, saving model to model-010-0.976811.h5
307/307 [==============================] - 20s 65ms/step - loss: 0.0528 - accuracy: 0.9819 - val_loss: 0.0818 - val_accuracy: 0.9768 - lr: 0.0010
```

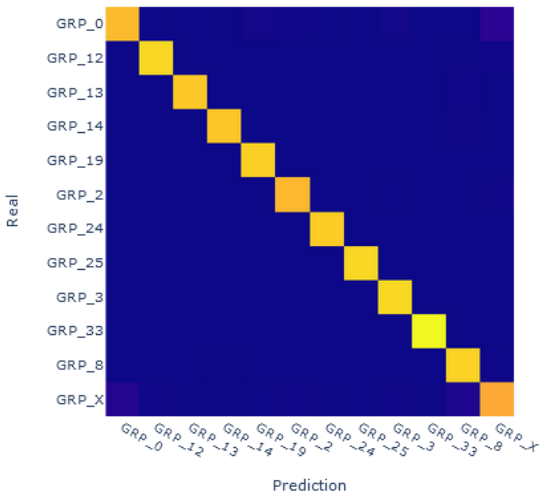
Checking Model Accuracy

```
y_pred = model.predict(x_test )

pred =[]
for val in y_pred:
  pred.append(np.argmax(val))
```

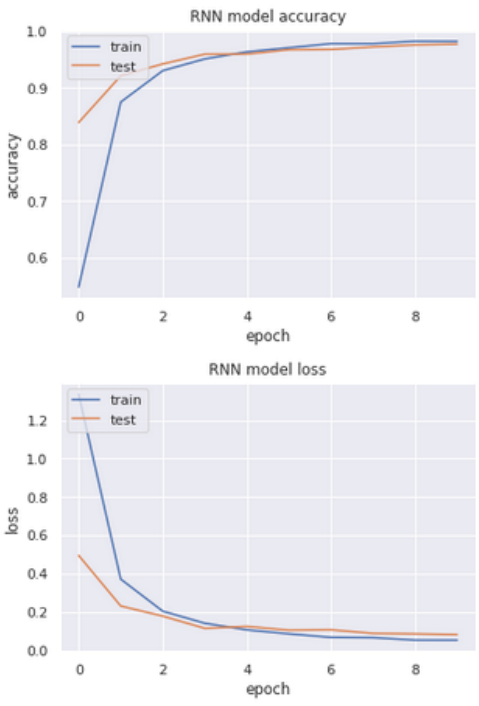
```
test_acc  = metrics.accuracy_score( y_test , pred )

con_mat = metrics.confusion_matrix(y_test , pred)


print("Test  Accuracy -> ",round(test_acc  *100,2),"%")
print("F1 score       -> ",metrics.f1_score( y_test , pred, average='micro'))

print("\n")



fig = px.imshow(con_mat,labels=dict(x="Prediction", y="Real", color="Count"),
                x=encoder.classes_,
                y=encoder.classes_)
fig.show()
```

```
Test  Accuracy ->  97.68 %
F1 score       ->  0.9768108389786347
```

Heatmap Prediction Vs Real




```
plt.plot(model_history.history['accuracy'])
plt.plot(model_history.history['val_accuracy'])

plt.title('RNN model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train','test'], loc='upper left')
plt.show()

plt.plot(model_history.history['loss'])
plt.plot(model_history.history['val_loss'])

plt.title('RNN model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train','test'], loc='upper left')
plt.show()
```



Evaluation of the Final Model (RNN)

Train Accuracy : 98.97%

Test Accuracy : 97.64%

F1 score : 0.98

Saving the model

The model is saved for further usage and User Interface (UI) development.

```
[ ] import joblib

    model.save("/content/drive/MyDrive/Colab Notebooks/capstone project/final/rnn_final.h5")
    joblib.dump(tokenizer, '/content/drive/MyDrive/Colab Notebooks/capstone project/final/tokenizer.pkl')
    joblib.dump(encoder, '/content/drive/MyDrive/Colab Notebooks/capstone project/final/label_encoder.pkl')

    ['/content/drive/MyDrive/Colab Notebooks/capstone project/final/label_encoder.pkl']
```

3. Step-by-step walk through the solution

Exploring the given Data files

- As the first step, imported the dataset – the excel file given.
- Viewed some sample records to understand the data.

	Short description	Description	Caller	Assignment group
7526	not possible to login due to a locked account	not possible to login due to a locked account	qufjnslk bvtfrwnu	GRP_0
3709	do not have permission to ess portal. at	please reach out to me for more information if...	ytwmgpbk cpawsihk	GRP_2
3237	erp accout had been locked	_x000D_\n_x000D_\nreceived from: qyidkvap.cxnf...	qyidkvap cxnfdjpk	GRP_0
7956	unable to login to vpn and reset password	unable to login to vpn and reset password	xiaurwpz hzusljmc	GRP_0
5751	when i create a recurring appointment in mds i...	when i create a recurring appointment in mds i...	acteiqdu bferalus	GRP_40
5050	unable to launch outlook	unable to launch outlook	vjwdyanl knfsjdgz	GRP_0

- Checked the shape of the dataset, it is found to have 8500 rows and 4 columns.

```
(8500, 4)  |  The dataset has 8500 rows and 4 columns
```

Understanding the structure of data

- To understand the data properly, viewed what are the index of the columns and it found to be “Short description”, “Description”, “Caller” and “Assignment group”

```
[ ]  dataset.columns
```

```
Index(['Short description', 'Description', 'Caller', 'Assignment group'], dtype='object')
```

- By checking the datatypes of each column, we found that, all of them are “object” datatype.

```
dataset.info()
```

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

- Our focus is on “Assignment group” so by checking unique, we found that there are 74 unique categories.

```
len(dataset["Assignment group"].unique())
```

```
74
```

- By checking value_counts of each group, it shows that there is a huge imbalance in the data. There are a lot of Assignment groups with <100 tickets. So better combine them for further processing to make it simpler.

```
pd.DataFrame( dataset["Assignment group"].value_counts() )
```

	Assignment group
GRP_0	3976
GRP_8	661
GRP_24	289
GRP_12	257
GRP_9	252
...	...
GRP_64	1
GRP_67	1
GRP_35	1
GRP_70	1
GRP_73	1

74 rows × 1 columns

- By checking number of unique values in each column, we could understand that there are 7481 unique values in “Short description”, 7817 unique values in “Description”, 2950 unique values in “Caller” and 74 unique values in “Assignment group”.

```
dataset.nunique()
```

```
Short description    7481
Description          7817
Caller               2950
Assignment group       74
dtype: int64
```

There seems to be duplicates in the dataset (number of unique values is lower than the total number of rows, 8500) so we should consider dropping them.

- Using info and describe we can check for extra information about the dataset. There seems to be a few null values in "Short description" and "Description" columns.

```
dataset.info()
```

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

```
dataset.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

Missing points in data

- We performed checking the presence of “NaN” and “null” values and found that there are 8 null/NaN values in "Short description" and 1 null/NaN value in "Description".

```
[ ]  dataset.isna().sum()
```

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

```
[ ]  dataset.isnull().sum()
```

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

- We also performed Checking the null / NaN datapoints and able to understand that 8 of the null/NaN values are from GRP_0 and 1 is from GRP_34 so before dropping / replacing checked the value_counts of GRP_0 and GRP_34 and found as 3976 numbers of GRP_0 and 62 numbers of GRP_34.

- Also found that only one of the fields is missing for each of the cases so we replaced NaN with " " (empty string). We did this while dealing with missing values.

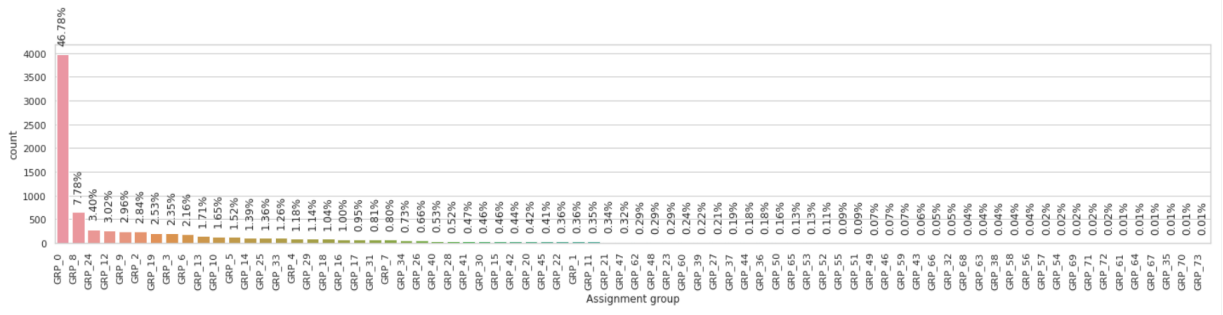
```
dataset[dataset.isna().any(axis=1)]
```

	Short description	Description	Caller	Assignment group
2604	NaN	_x000D_\n_x000D_\nreceived from: ohdrnswl.rezu...	ohdrnswl rezuibdt	GRP_34
3383	NaN	_x000D_\n-connected to the user system using t...	qftpazns fxpnytmk	GRP_0
3906	NaN	-user unable tologin to vpn._x000D_\n-connect...	awpcmsey ctdiuqwe	GRP_0
3910	NaN	-user unable tologin to vpn._x000D_\n-connect...	rhwsmefo tvphyura	GRP_0
3915	NaN	-user unable tologin to vpn._x000D_\n-connect...	hxripljo efzounig	GRP_0
3921	NaN	-user unable tologin to vpn._x000D_\n-connect...	cziadygo veiosxby	GRP_0
3924	NaN	name:wvqgbdhm fwchqjor\nlanguage:\nbrowser:mic...	wvqgbdhm fwchqjor	GRP_0
4341	NaN	_x000D_\n_x000D_\nreceived from: eqmuniov.ehxk...	eqmuniov ehxkcbgj	GRP_0
4395	i am locked out of skype	NaN	viyglzfo ajtfzpkb	GRP_0

```
counts = dataset["Assignment group"].value_counts()

print(f"Count of GRP_0  : {counts.GRP_0}")
print(f"Count of GRP_34 : {counts.GRP_34}")
```

```
Count of GRP_0  : 3976
Count of GRP_34 : 62
```

Finding inconsistencies in the data



- From the above graph we can infer that the dataset is highly unbalanced with 3976 datapoints in GRP_0
- By checking the duplicates, we found that, there are 83 duplicate records present.

```
sum(dataset.duplicated())
```

```
83
```

- We checked the “Caller” column, and understood that this column does not provide any useful data. So we can drop this column.

```
dataset["Caller"].sample(10)
```

```
7838    wphqnxly htvrbxmd
6162    jloygrwh acvztedi
2085    wtgbdjzl coliybmq
647     rkupnshb gsmzfojw
7755    fumkcsji sarmtlhy
7596    vkzwafuh tcjnuswg
7781    wbqtfzdv aectbluw
822     ljpgedia bzqcwsgf
5501    jyoqwxhz clhxsoqy
2070    rxoluzhy pnutohms
Name: Caller, dtype: object
```

Dealing with the inconsistencies

- We made a copy of the dataset for further processing
- Dropped the “Caller” column as decided

```
dataset_a.drop("Caller",axis=1,inplace = True)

dataset_a.head()
```

	Short description	Description	Assignment group
0	login issue	-verified user details.(employee# & manager na...	GRP_0
1	outlook	_x000D_\n_x000D_\nreceived from: hmjdrvpb.komu...	GRP_0
2	cant log in to vpn	_x000D_\n_x000D_\nreceived from: eylqgodm.ybqk...	GRP_0
3	unable to access hr_tool page	unable to access hr_tool page	GRP_0
4	skype error	skype error	GRP_0

- After dropping "Caller", the duplicate count increased from 83 to 591.

```
print("Number of duplicates :",sum(dataset_a.duplicated()))
```

```
Number of duplicates : 591
```

- Dropped all the duplicate records

```
dataset_a.drop_duplicates(inplace = True)

print("Number of duplicates :",sum(dataset_a.duplicated()))
```

```
Number of duplicates : 0
```

- Did a quick check of lower frequency groups and found as below, we can combine all the groups with less than 100 tickets to one group named "GRP_X".

```
print(f"Less than 50  : {threshold_counts[0]}")
print(f"Less than 100 : {threshold_counts[1]}")
print(f"Less than 150 : {threshold_counts[2]}")
print(f"Less than 200 : {threshold_counts[3]}")
print(f"Less than 250 : {threshold_counts[4]}")
```

```
Less than 50  : 50
Less than 100 : 59
Less than 150 : 65
Less than 200 : 66
Less than 250 : 69
```

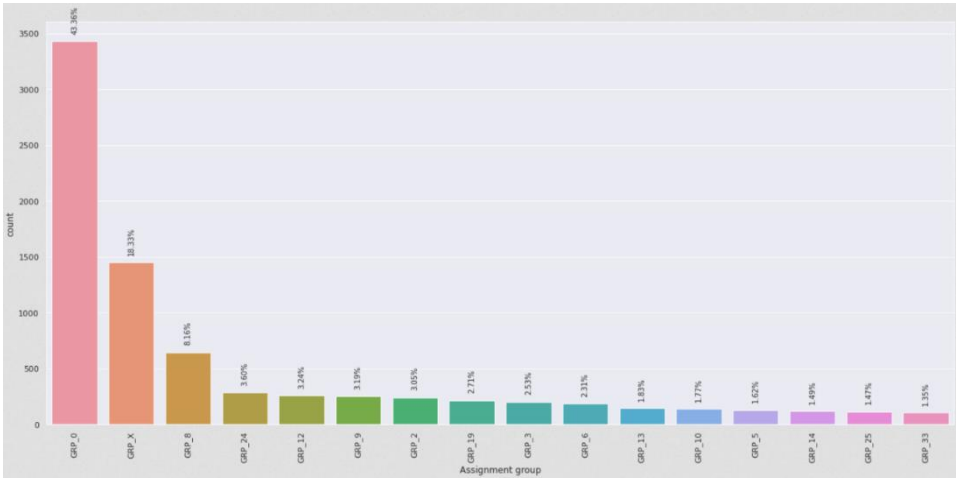
- After combining to GRP_X, the value count of each group is as below.

```
counts = dataset_a["Assignment group"].value_counts()

dataset_a["Assignment group"] = np.where(counts[dataset_a["Assignment group"]] < 100 , "GRP_X", dataset_a["Assignment group"])

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

```
GRP_0     3429
GRP_X     1450
GRP_8      645
GRP_24     285
GRP_12     256
GRP_9      252
GRP_2      241
GRP_19     214
GRP_3      200
GRP_6      183
GRP_13     145
GRP_10     140
GRP_5      128
GRP_14     118
GRP_25     116
GRP_33     107
Name: Assignment group, dtype: int64
```

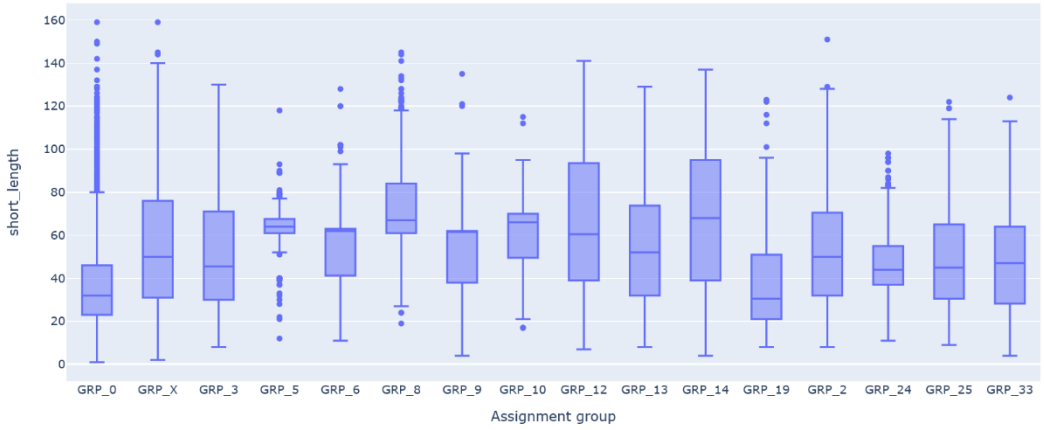
Visualizing different patterns

a. Assignment group Vs Count

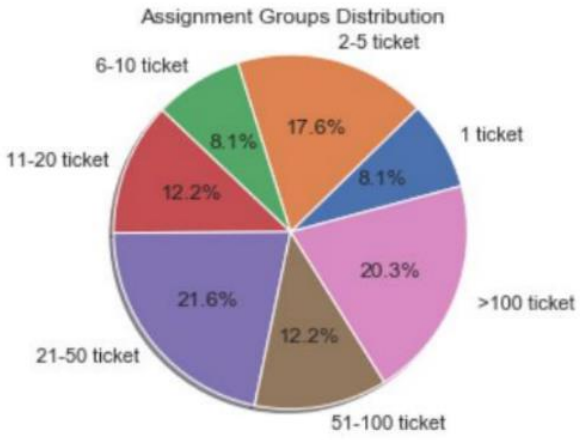


Even after merging lower frequency groups to one (GRP_X), the data is highly unbalanced.

b. Assignment group Vs short description length - the distribution of short description length across Assignment groups is shown as below:

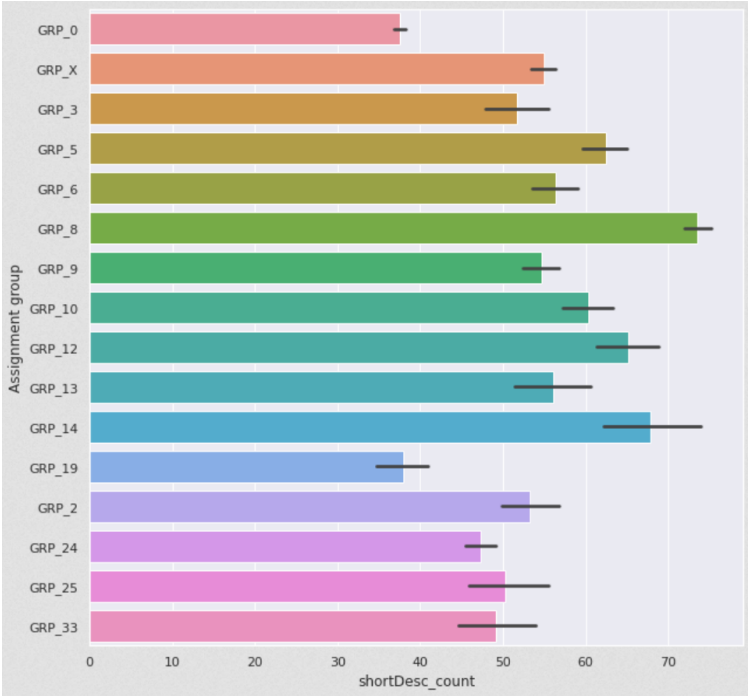


c. Distribution of ticket counts in various bins as below:



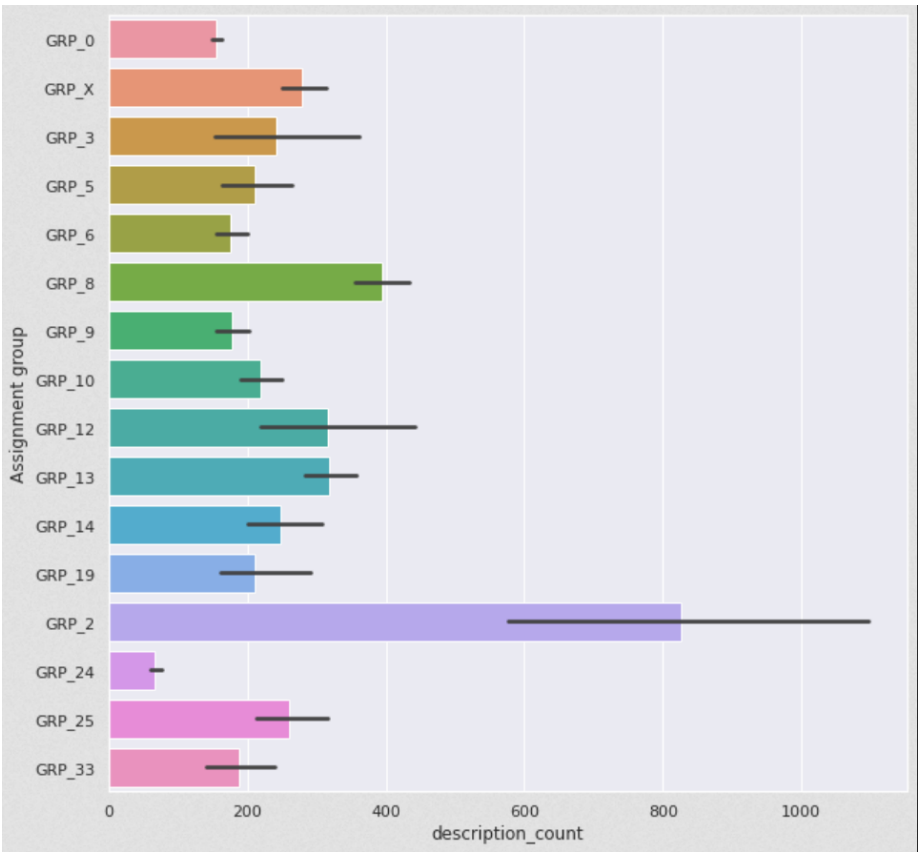
Visualizing different text features

a. Short description count Vs Assignment group – the distribution of word counts across Assignment groups for the 'Short description is shown as below:



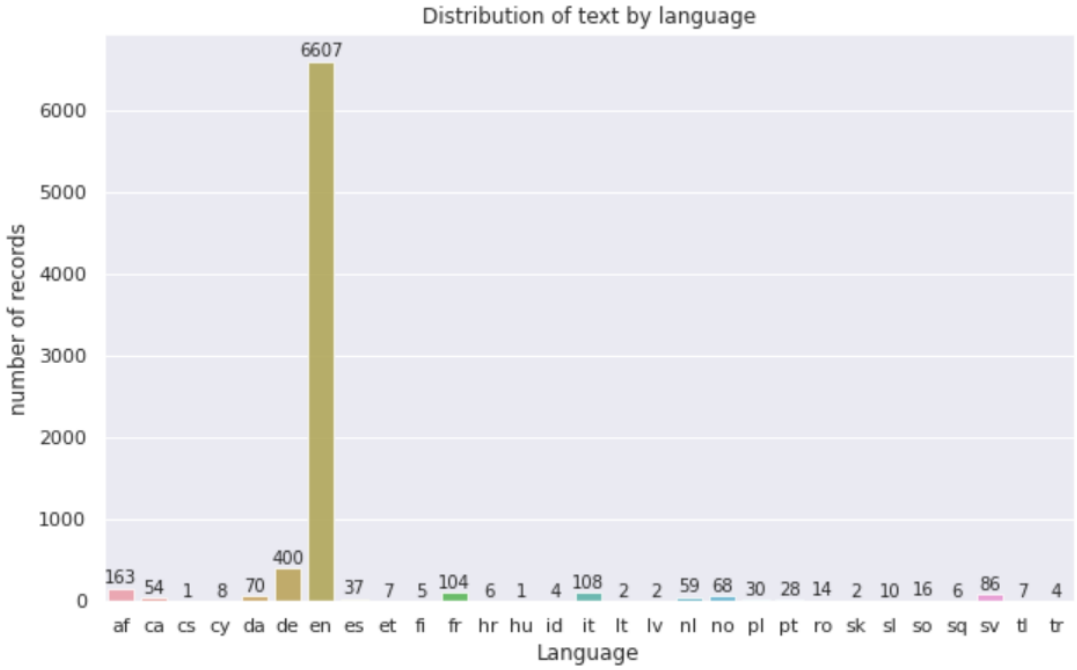
b. Word cloud image – Assignment group Vs short description - Using word could visualized the text features

c. Description count Vs Assignment group



d. Word cloud image – Assignment group Vs Description

- Some of the Groups seem to contain non-English words so we run language detection. Majority records are of English.



Dealing with missing values

As decided replaced all the NaN values with empty string

```
print("Before replacing \n")
dataset_a.isna().sum()
```

```
Before replacing

Short description    5
Description          1
Assignment group     0
dtype: int64
```

```
dataset_a.fillna('', inplace=True)
```

```
print("After replacing \n")
dataset_a.isna().sum()
```

```
After replacing

Short description    0
Description          0
Assignment group     0
dtype: int64
```

Text preprocessing

As this is a multi-label classification problem we will merged the Short description and Description columns together, so that all the description terms are together for a particular sentence.

	Assignment group	Text
0	GRP_0	[login issue, -verified user details.(employee...
1	GRP_0	[outlook, _x000D_\n_x000D_\nreceived from: hmj...
2	GRP_0	[cant log in to vpn, _x000D_\n_x000D_\nreceive...
3	GRP_0	[unable to access hr_tool page, unable to acce...
4	GRP_0	[skype error , skype error]

Removed unwanted characters and numbers

For further processing, we need useful data only, so we removed unwanted characters and numbers for the dataset.

```
dataset_text["Text"]=dataset_text["Text"].str.replace('[^A-Za-z]',' ')

dataset_text.head()
```

	Text
0	login issue verified user details emplo...
1	outlook x D n x D nreceived from ...
2	cant log in to vpn x D n x D nrece...
3	unable to access hr tool page unable to a...
4	skype error skype error

Converting to lower case letters

We converted all the text to lower case letters for simplicity of processing (to avoid the managing of upper- and lower-case letters)

```
dataset_text["Text"] = dataset_text["Text"].str.lower()

dataset_text.head()
```

	Text
0	login issue verified user details emplo...
1	outlook x d n x d nreceived from ...
2	cant log in to vpn x d n x d nrece...
3	unable to access hr tool page unable to a...
4	skype error skype error

Removing unnecessary white spaces

Using .strip() we removed all the unnecessary white spaces for the text.

```
dataset_text["Text"]= dataset_text["Text"].str.strip()

dataset_text.head()
```

	Text
0	login issue verified user details employe...
1	outlook x d n x d nreceived from hm...
2	cant log in to vpn x d n x d nreceiv...
3	unable to access hr tool page unable to acc...
4	skype error skype error

Removing stopwords

Using stopwords from nltk library, removed stopwords

```
dataset_text["Text"]= dataset_text["Text"].apply(lambda x: ' '.join([word for word in x.split() if word not in sw]))

dataset_text.head()
```

	Text
0	login issue verified user details employee man...
1	outlook x n x nreceived hmjdrvpb komuaywn gmai...
2	cant log vpn x n x nreceived eylqgodm ybqkwiam...
3	unable access hr tool page unable access hr to...
4	skype error skype error

Performing Lemmatization

Performed Lemmatization using WordNetLemmatizer

```
for index, row in dataset_text.iterrows():

  dataset_text.at[index, 'Text']= lemmatize_sentence(row["Text"])

dataset_text.head()
```

	Text
0	login issue verify user detail employee manage...
1	outlook x n x nreceived hmjdrvpb komuaywn gmai...
2	cant log vpn x n x nreceived eylqgodm ybqkwiam...
3	unable access hr tool page unable access hr to...
4	skype error skype error

Replacing the text column of dataset

After performing Lemmatization, replaced the text column of the dataset with the new text

```
dataset_a['Text']=dataset_text['Text']

dataset_a.head()
```

	Assignment group	Text
0	GRP_0	login issue verify user detail employee manage...
1	GRP_0	outlook x n x nreceived hmjdrvpb komuaywn gmai...
2	GRP_0	cant log vpn x n x nreceived eylqgodm ybqkwiam...
3	GRP_0	unable access hr tool page unable access hr to...
4	GRP_0	skype error skype error

Visualizing the text column after processing

Using wordclould again visualized the text features.

Creating word vocabulary from the corpus of report text data

```
word_counts=dict()

for words in dataset_a.Text.str.split():
    for word in words:
        if word in word_counts:
            word_counts[str(word)]+=1
        else:
            word_counts[str(word)]=1

word_counts
```

```
{'login': 669,
 'issue': 1430,
 'verify': 184,
 'user': 1452,
 'detail': 252,
 'employee': 192,
 'manager': 305,
 'name': 672,
 'x': 24198,
 'n': 22403,
 'check': 451,
 'ad': 128,
 'reset': 1221,
 'password': 2109,
 'advise': 142,
 'caller': 64,
 'confirm': 140,
 'able': 469,
 'resolve': 217,
```

The total number of words in this dictionary is 15480 words

```
print("Total number of words in the dictionary : ", len(word_counts))
```

```
Total number of words in the dictionary :  15480
```

Creating Tokens

Did tokenization using word_tokenize for the Text column for further processing.

```
dataset_a["text_tokens"] = [ word_tokenize(txt) for txt in dataset_a["Text"] ]

dataset_a.head()
```

	Assignment group	Text	text_tokens
0	GRP_0	login issue verify user detail employee manage...	[login, issue, verify, user, detail, employee,...
1	GRP_0	outlook x n x nreceived hmjdrvpb komuaywn gmai...	[outlook, x, n, x, nreceived, hmjdrvpb, komuay...
2	GRP_0	cant log vpn x n x nreceived eylqgodm ybqkwiam...	[cant, log, vpn, x, n, x, nreceived, eylqgodm,...
3	GRP_0	unable access hr tool page unable access hr to...	[unable, access, hr, tool, page, unable, acces...
4	GRP_0	skype error skype error	[skype, error, skype, error]

Model Building

Tokenizing and encoding

```
[ ] import tensorflow as tf
    from sklearn.model_selection import train_test_split
    from sklearn.preprocessing import LabelEncoder

    from sklearn import model_selection, preprocessing, linear_model, naive_bayes, metrics, svm
    from sklearn.tree import DecisionTreeClassifier
    from sklearn.neighbors import KNeighborsClassifier
    from sklearn.ensemble import BaggingClassifier,AdaBoostClassifier,GradientBoostingClassifier
    from sklearn.linear_model import SGDClassifier
    from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
    from sklearn import ensemble

    import  numpy, textblob, string
```

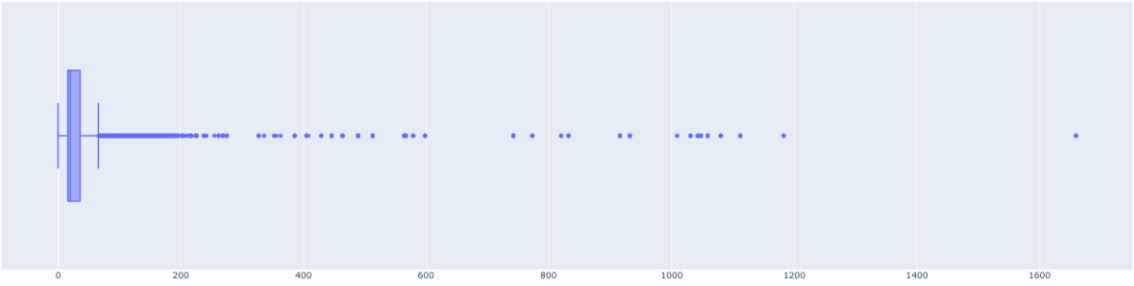
```
    top_k = 10000
    tokenizer = tf.keras.preprocessing.text.Tokenizer(num_words=top_k,
                                                      oov_token="<unk>",
                                                      filters='!"#$%&()*+.,-/:;=?@[\]^_`{|}~ ')
    tokenizer.fit_on_texts(dataset_resampled['Text'])
```

```
[ ] tokenizer.word_index['<pad>'] = 0
    tokenizer.index_word[0] = '<pad>'
```

```
[ ] train_seqs = tokenizer.texts_to_sequences(dataset_resampled['Text'])
```

Finding the length distribution of the sequences

```
[ ] fig = px.box( x=[ len(y) for y in train_seqs])
    fig.show()
```



4. Model Evaluation

Creating the function “evaluate_model” for evaluating the models in the same way

```
[ ] max_features = 10000
    maxlen = 300
    embedding_size = 200
```

```
[ ] X = tf.keras.preprocessing.sequence.pad_sequences(train_seqs, padding='post',maxlen=maxlen)

    Y = dataset_resampled["Assignment group"].astype("category")
```

```
[ ] Y.dtype
```

```
CategoricalDtype(categories=['GRP_0', 'GRP_10', 'GRP_12', 'GRP_13', 'GRP_14', 'GRP_19',
                  'GRP_2', 'GRP_24', 'GRP_25', 'GRP_3', 'GRP_33', 'GRP_5',
                  'GRP_6', 'GRP_8', 'GRP_9', 'GRP_X'],
, ordered=False)
```

```
[ ] X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0.20, random_state = 42)
    print(X_train.shape,Y_train.shape)
    print(X_test.shape,Y_test.shape)
```

```
(43891, 300) (43891,)
(10973, 300) (10973,)
```

```
[ ] encoder = LabelEncoder()
    train_y = encoder.fit_transform(Y_train)
    test_y = encoder.fit_transform(Y_test)
```

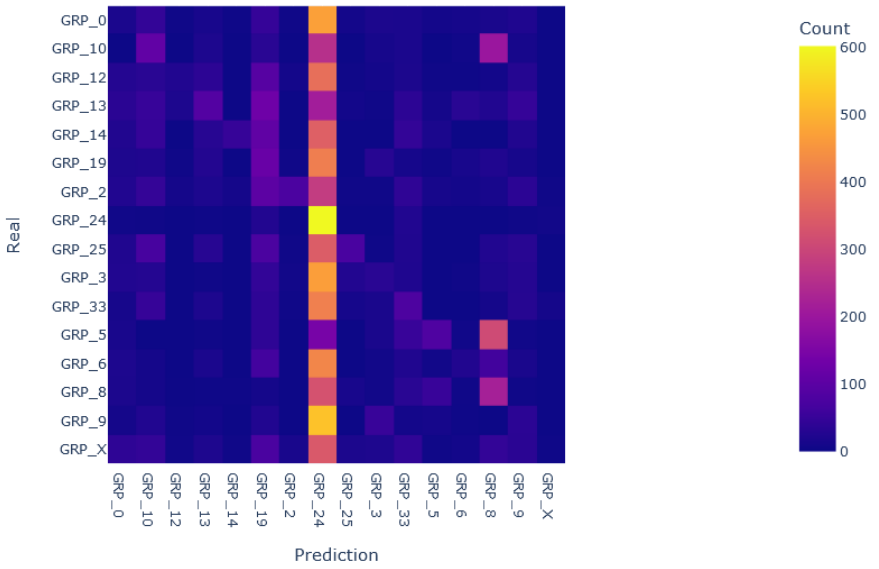
```
[ ] def evaluate_model(model, x_train, y_train, x_test, y_test):

      model.fit(x_train,y_train)

      train_acc = metrics.accuracy_score( y_train, model.predict(x_train) )
      test_acc  = metrics.accuracy_score( y_test , model.predict(x_test ) )

      con_mat = metrics.confusion_matrix(y_test , model.predict(x_test ))


      print("Train Accuracy -> ",round(train_acc *100,2),"%")
      print("Test  Accuracy -> ",round(test_acc  *100,2),"%")
      print("F1 score       -> ",metrics.f1_score( y_test , model.predict(x_test ), average='micro'))

      print("\n")

      fig = px.imshow(con_mat,labels=dict(x="Prediction", y="Real", color="Count"),
                      x=encoder.classes_,
                      y=encoder.classes_)
      fig.show()
```

Using the above “evaluate_model” function, we evaluated all the possible options of the model.

Naive Bays

```
[ ] evaluate_model(naive_bayes.MultinomialNB() ,X_train, train_y, X_test, test_y )
```

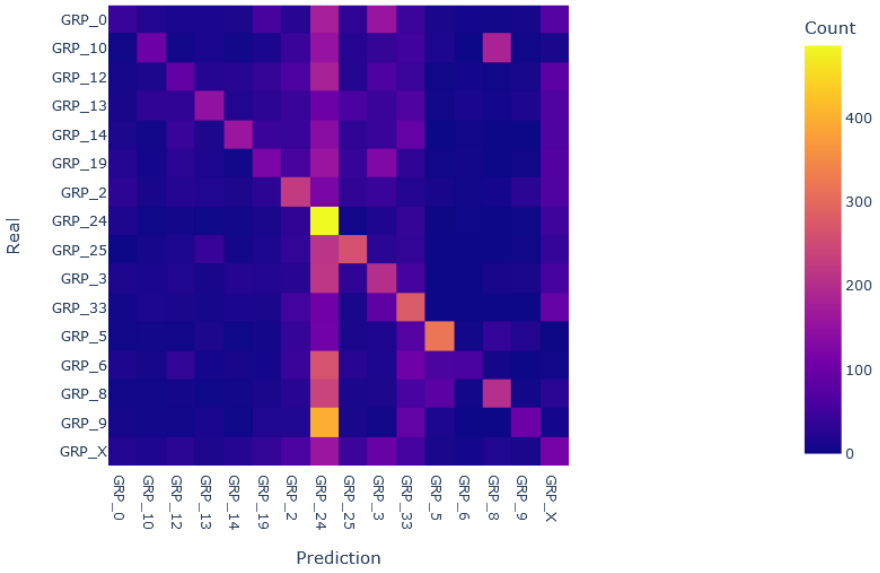
```
Train Accuracy ->  15.33 %
Test  Accuracy ->  14.78 %
F1 score       ->  0.14781736990795588
```



Logistic Regression

```
[ ] evaluate_model(linear_model.LogisticRegression() ,X_train, train_y, X_test, test_y )
```

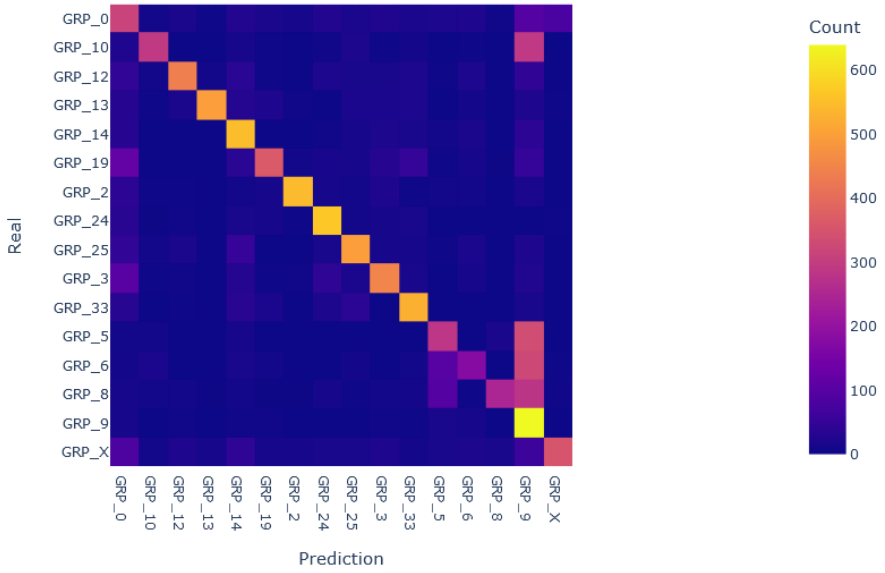
```
Train Accuracy ->  27.14 %
Test  Accuracy ->  26.42 %
F1 score       ->  0.2641939305568213
```



SVM

```
[ ]  evaluate_model(svm.SVC() ,X_train, train_y, X_test, test_y )
```

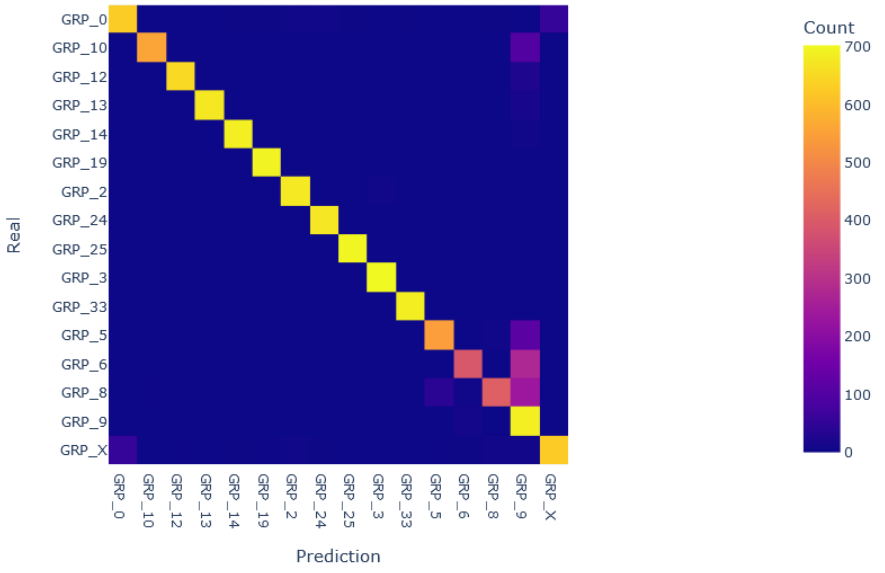
```
Train Accuracy ->  63.29 %
Test  Accuracy ->  61.28 %
F1 score       ->  0.6127768158206507
```



Random Forest

```
[ ]  evaluate_model(ensemble.RandomForestClassifier() ,X_train, train_y, X_test, test_y )
```

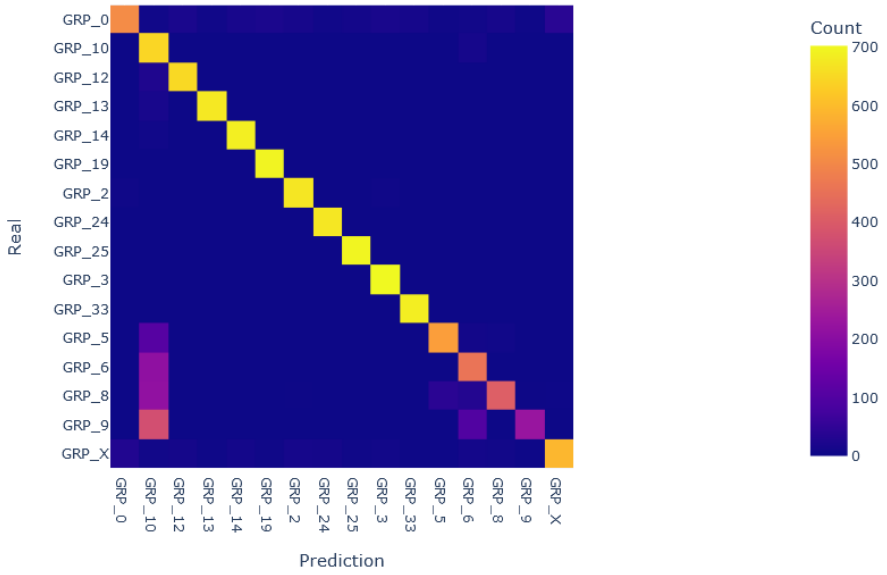
```
Train Accuracy ->  92.19 %
Test  Accuracy ->  90.96 %
F1 score       ->  0.9095962817825571
```



KNN Classifier

```
[ ]  evaluate_model(KNeighborsClassifier(n_neighbors= 5 , weights = 'distance') ,X_train, train_y, X_test, test_y )
```

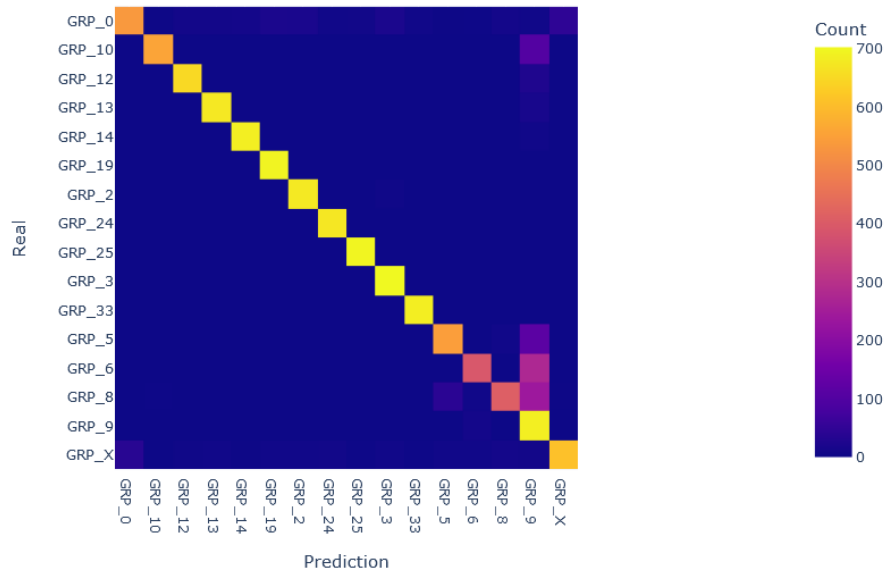
```
Train Accuracy ->  89.73 %
Test  Accuracy ->  86.69 %
F1 score       ->  0.8669461405267475
```



Decision Tree

```
[ ]  model = DecisionTreeClassifier(criterion = "entropy")

     evaluate_model(model ,X_train, train_y, X_test, test_y )
```

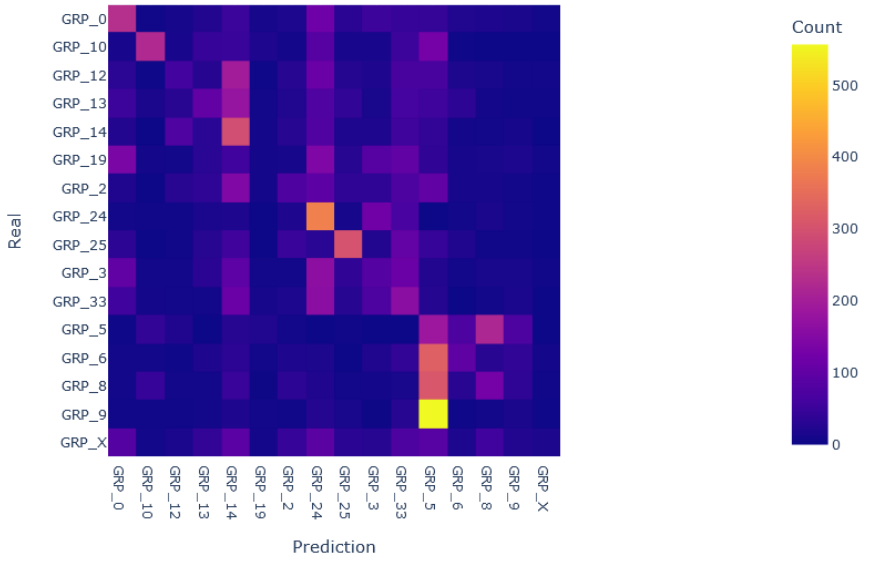
```
Train Accuracy ->  92.19 %
Test  Accuracy ->  89.92 %
F1 score       ->  0.8992071448099882
```



ADA Boosting

```
[ ]  evaluate_model(AdaBoostClassifier(n_estimators=50,random_state=1) ,X_train, train_y, X_test, test_y )
```

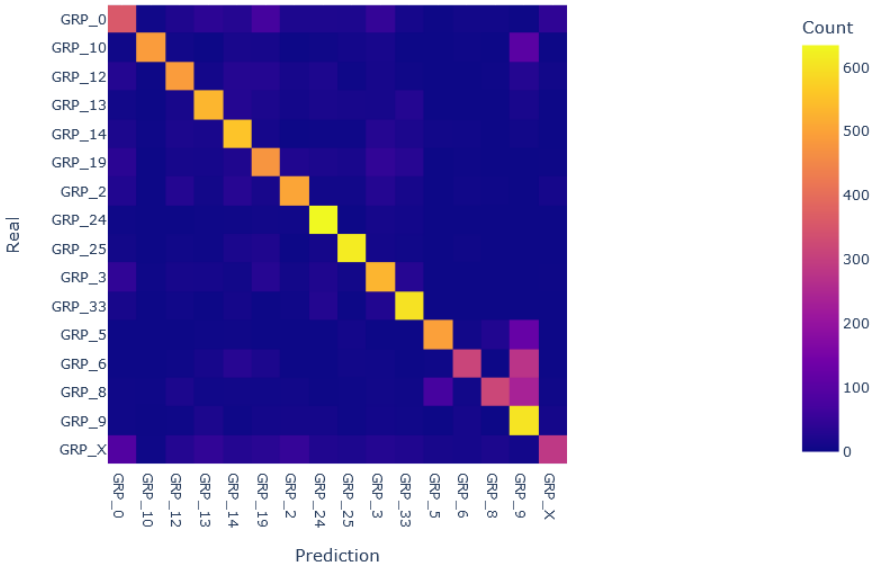
```
Train Accuracy ->  21.43 %
Test  Accuracy ->  21.47 %
F1 score       ->  0.2147088307664267
```



Gradient Boosting

```
[ ]  evaluate_model(GradientBoostingClassifier(n_estimators = 50,random_state=1) ,X_train, train_y, X_test, test_y )
```

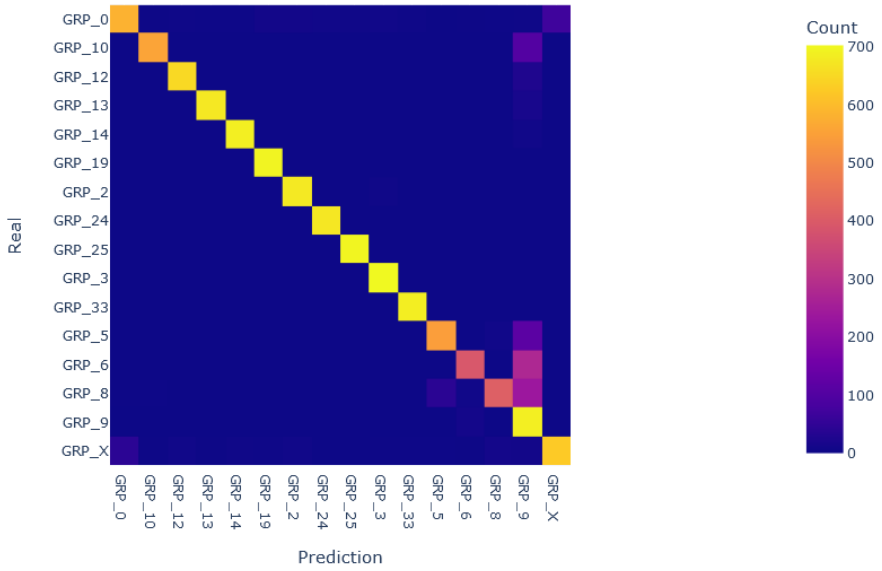
```
Train Accuracy ->  73.28 %
Test  Accuracy ->  71.05 %
F1 score       ->  0.710471156474984
```



Bagging

```
[ ]  dTreeR = DecisionTreeClassifier()
     evaluate_model(BaggingClassifier(base_estimator=dTreeR, n_estimators=50,random_state=1) ,X_train, train_y, X_test,
```

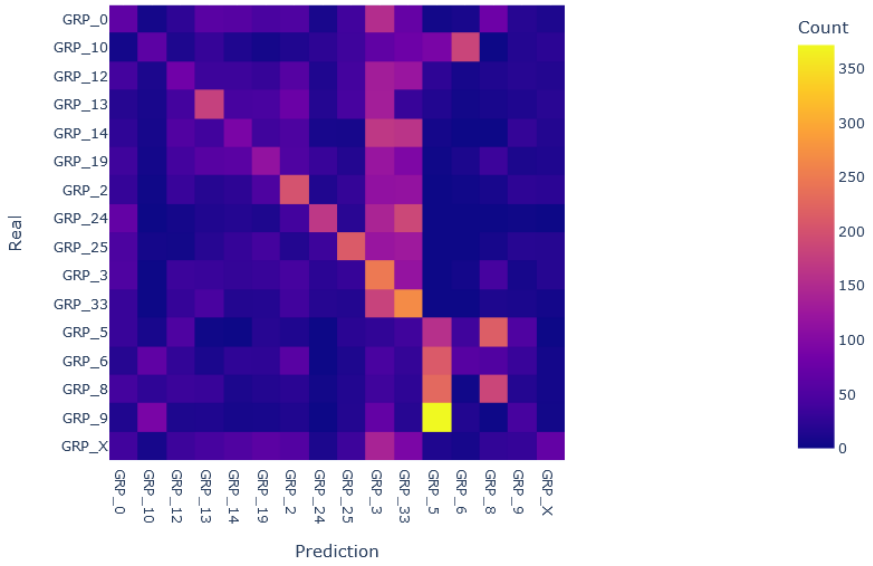
```
Train Accuracy ->  92.19 %
Test  Accuracy ->  90.5 %
F1 score       ->  0.9050396427595006
```



SGD Classifier

```
[ ]  evaluate_model(SGDClassifier(loss="hinge", penalty="l2") ,X_train, train_y, X_test, test_y )
```

```
Train Accuracy ->  20.07 %
Test  Accuracy ->  19.99 %
F1 score       ->  0.19994532033172333
```



Comparing models using LazyText

For a quick comparison of the possible models, we used LazyText

```
[ ] !pip install lazytext
    from lazytext.supervised import LazyTextPredict

    lazy_text = LazyTextPredict(
        classification_type="multiclass",
        )

    print(lazy_text.get_all_classifiers)

    models = lazy_text.fit(X_train, X_test, train_y, test_y)
```

Result of LazyText Analysis

Result Analysis

Model	Accuracy	Balanced Accuracy	F1 Score
AdaBoostClassifier	0.21470883076642…	0.21574487438851…	0.1989704548323668
BaggingClassifier	0.90294358880889…	0.90230347364342…	0.9078010401975063
BernoulliNB	0.17488380570491…	0.17542266443754…	0.11146512334545577
CalibratedClassifierCV	0.23293538685865…	0.23412049838176…	0.19363065316714978
ComplementNB	0.12913514991342…	0.13012322236087…	0.10548330616814479
DecisionTreeClassifier	0.89784015310307…	0.89727184372477…	0.9023135753058475
DummyClassifier	0.05987423676296…	0.0625	0.0070614789337919…
ExtraTreeClassifier	0.89866034812722…	0.89809149253015…	0.9032641728144752
ExtraTreesClassifier	0.90977854734347…	0.90899424596747…	0.9149950807407792
GradientBoostingClassifi…	0.80588717761778…	0.806337022169324	0.80603973958625
KNeighborsClassifier	0.823749202588171	0.82550204926477…	0.8176615458943616
LinearSVC	0.10079285519001…	0.10179830191988…	0.090160011124007
LogisticRegression	0.26419393055682…	0.26494003077250…	0.2593797306251261
LogisticRegressionCV	0.27112002187186…	0.27173874908486…	0.26661835950072477
MLPClassifier	0.74619520641574…	0.74717998972350…	0.7509030212329035
MultinomialNB	0.14781736990795…	0.14896905187543…	0.1267277805899355
NearestCentroid	0.17980497584981…	0.18123025805248…	0.155788160915052
NuSVC	0.54880160393693…	0.54849568335512…	0.5492471563849667
PassiveAggressiveClassif…	0.21407090130319…	0.21406388978498…	0.19889440949761428
Perceptron	0.17962271028889…	0.17959800973142…	0.1909172492037351
RandomForestClassifier	0.908776086758407	0.90798843170886…	0.9140067883343734
RidgeClassifier	0.24879249065889	0.25066819981446…	0.22833549926773059
SGDClassifier	0.189373917798232	0.18977828607895…	0.18661854583824933
SVC	0.61277681582065…	0.61192608695007…	0.6223427738297788

Deep NN

```
def Build_Model_DNN_Text(shape, nClasses, dropout=0.3):
    """
    buildModel_DNN_Tex(shape, nClasses,dropout)
    Build Deep neural networks Model for text classification
    Shape is input feature space
    nClasses is number of classes
    """
    model = Sequential()
    node = 512 # number of nodes
    nLayers = 4 # number of  hidden layer
    model.add(Dense(node,input_dim=shape,activation='relu'))
    model.add(Dropout(dropout))
    for i in range(0,nLayers):
        model.add(Dense(node,input_dim=node,activation='relu'))
        model.add(Dropout(dropout))
        model.add(BatchNormalization())
    model.add(Dense(nClasses, activation='softmax'))
    model.compile(loss='sparse_categorical_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy'])
    print(model.summary())
    return model
```

```
model_DNN.fit(X_train_tfidf, y_train,
              validation_data=(X_test_tfidf,y_test),
              callbacks=call_backs("NN"),
              epochs=5,
              batch_size=128,
              verbose=2)
predicted = model_DNN.predict(X_test_tfidf)
```

Python

```
WARNING:tensorflow:`period` argument is deprecated. Please use `save_freq` to specify the frequency in
number of batches seen.
Epoch 1/5

Epoch 1: val_loss improved from inf to 3.03589, saving model to Weights/NN_epoch01_loss3.0359.h5
343/343 - 12s - loss: 0.9284 - accuracy: 0.7023 - val_loss: 3.0359 - val_accuracy: 0.1166 - 12s/epoch
- 34ms/step
Epoch 2/5

Epoch 2: val_loss improved from 3.03589 to 0.25882, saving model to Weights/NN_epoch02_loss0.2588.h5
343/343 - 6s - loss: 0.3236 - accuracy: 0.8840 - val_loss: 0.2588 - val_accuracy: 0.9037 - 6s/epoch -
17ms/step
Epoch 3/5

Epoch 3: val_loss did not improve from 0.25882
343/343 - 6s - loss: 0.2715 - accuracy: 0.8998 - val_loss: 0.2615 - val_accuracy: 0.9076 - 6s/epoch -
17ms/step
Epoch 4/5

Epoch 4: val_loss did not improve from 0.25882
343/343 - 6s - loss: 0.2593 - accuracy: 0.9019 - val_loss: 0.2596 - val_accuracy: 0.9067 - 6s/epoch -
17ms/step
Epoch 5/5

Epoch 5: val_loss improved from 0.25882 to 0.25406, saving model to Weights/NN_epoch05_loss0.2541.h5
343/343 - 6s - loss: 0.2501 - accuracy: 0.9063 - val_loss: 0.2541 - val_accuracy: 0.9079 - 6s/epoch -
17ms/step
```

LSTM

```
deep_inputs = Input(shape=(maxlen,),dtype='int32')
embedding_layer = Embedding(vocab_size, 100, weights=[embedding_matrix], trainable=False, input_length=maxlen)(deep_inputs)


deep_inputs = Input(shape=(maxlen,),dtype='int32')
embedding_layer = Embedding(vocab_size, 100, weights=[embedding_matrix], trainable=False, input_length=maxlen)(deep_inputs)
LSTM_Layer_1 = LSTM(128)(embedding_layer)
outputs = Dense(16, activation='sigmoid')(LSTM_Layer_1)
model = Model(inputs=deep_inputs, outputs=outputs)

model.compile(loss='sparse_categorical_crossentropy',optimizer="adam",metrics=['accuracy'] )
```

```
test_score = model.evaluate(X_test, y_test, verbose=1)

print("Test Score:", test_score[0])
print("Test Accuracy:", test_score[1])
```

```
343/343 [==============================] - 8s 23ms/step - loss: 0.3171 - accuracy: 0.8895
Test Score: 0.31711724400520325
Test Accuracy: 0.8894559144973755
```

CNN

```
input_ = Input(shape=(maxlen,))
embedding_layer= Embedding(vocab_size, 100, weights=[embedding_matrix], trainable=False, input_length=maxlen)(input_)
x = Conv1D(128, 3, activation='relu')(embedding_layer)
x = MaxPooling1D(3)(x)
x = Conv1D(128, 3, activation='relu')(x)
x = MaxPooling1D(3)(x)
x = Conv1D(128, 3, activation='relu')(x)
x = GlobalMaxPooling1D()(x)
x = Dense(128, activation='relu')(x)
output = Dense(16, activation='sigmoid')(x)

model_cnn = Model(input_, output)
model_cnn.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
test_score_cnn = model_cnn.evaluate(X_test, y_test, verbose=1)

print("Test Score:", test_score_cnn[0])
print("Test Accuracy:", test_score_cnn[1])
```

```
343/343 [==============================] - 2s 6ms/step - loss: 0.5076 - accuracy: 0.8336
Test Score: 0.5076003074645996
Test Accuracy: 0.8335915207862854
```

GRU

```
input_layer = Input(shape=(maxlen,),dtype='int32')
embed = Embedding(vocab_size,output_dim=100,input_length=maxlen,weights=[embedding_matrix], trainable=True)(input_layer)
gru=GRU(128)(embed)
drop=Dropout(0.3)(gru)
dense =Dense(100,activation='relu')(drop)
out=Dense(16,activation='softmax')(dense)
model_gru = Model(input_layer,out)
model_gru.compile(loss='sparse_categorical_crossentropy',optimizer="adam",metrics=['accuracy'])
```

```
train_score_gru = model_gru.evaluate(X_train, y_train, verbose=1)

print("Train Score:", train_score_gru[0])
print("Train Accuracy:", train_score_gru[1])
```

```
1372/1372 [==============================] - 32s 24ms/step - loss: 0.2364 - accuracy: 0.9154
Train Score: 0.23635515570640564
Train Accuracy: 0.9153813123703003
```

```
test_score_gru = model_gru.evaluate(X_test, y_test, verbose=1)

print("Test Score:", test_score_gru[0])
print("Test Accuracy:", test_score_gru[1])
```

```
343/343 [==============================] - 8s 24ms/step - loss: 0.3010 - accuracy: 0.9049
Test Score: 0.3010004758834839
Test Accuracy: 0.9048573970794678
```

In-depth analysis / fine-tuning of Random Forest

NOTE: The following code will take a very long time to execute. so after trial we have commented it out to save time

```
[ ] from sklearn.model_selection import GridSearchCV
    param_grid = {
        'max_depth': [80, 90, 100, 110],
        'max_features': [2, 3],
        'n_estimators': [100, 200, 300, 1000]
    }# Create a based model
    rf = ensemble.RandomForestClassifier()# Instantiate the grid search model
    grid_search = GridSearchCV(estimator = rf, param_grid = param_grid,
                               cv = 3, n_jobs = -1, verbose = 2)
```

```
[ ] param_grid = {
        'max_depth': [50, 60, 70],
        'max_features': [5,10 ,20],
        'n_estimators': [800, 1000, 1200]
    }# Create a based model
    rf = ensemble.RandomForestClassifier()# Instantiate the grid search model
    grid_search = GridSearchCV(estimator = rf, param_grid = param_grid,
                               cv = 3, n_jobs = -1, verbose = 2)
```

```
[ ] grid_search.fit(X_train, train_y)
    grid_search.best_params_
```

```
[ ] best_random = grid_search.best_estimator_

    y_pred = best_random.predict(X_test )

    train_acc = metrics.accuracy_score( train_y, best_random.predict(X_train) )
    test_acc  = metrics.accuracy_score( test_y , y_pred )

    con_mat = metrics.confusion_matrix(test_y ,y_pred)


    print("Train Accuracy -> ",round(train_acc *100,2),"%")
    print("Test  Accuracy -> ",round(test_acc  *100,2),"%")
    print("F1 score       -> ",metrics.f1_score( test_y , y_pred, average='micro'))

    print("\n")

    fig = px.imshow(con_mat)
    fig.show()
```

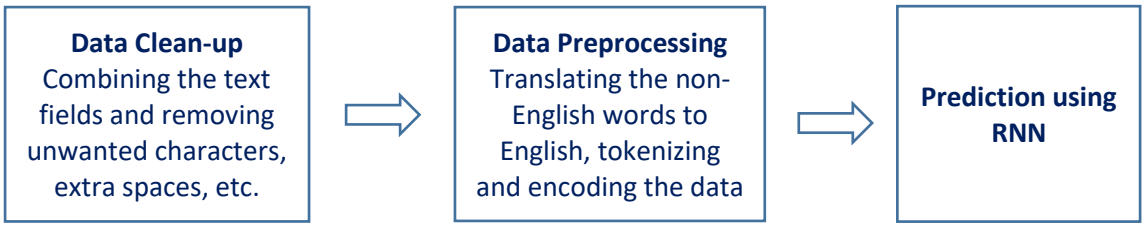
Comparison of Models

Comparison of models after in-depth analysis is as below. We are able to increase the accuracy of the RNN much higher than the other models after fine-tuning.

Sorted in decending order of Test Accuracy				
#	Model	Train Accuracy (%)	Test Accuracy (%)	F1 Score
	RNN (After fine-tuning)	98.80	97.60	0.98
1	Random Forest	92.19	90.90	0.91
2	RNN (Initial result)	91.07	90.68	0.89
3	Bagging	92.19	90.50	0.91
4	Decision Tree	92.19	90.04	0.90
5	KNN Classifier	92.05	89.25	0.89
6	Gradient Boosting	74.02	71.70	0.72
7	SVM	63.45	61.53	0.62
8	Logistic Regression	27.43	26.65	0.27
9	ADA Boosting	23.05	23.11	0.23
10	SGD Classifier	19.90	19.23	0.19
11	Naive Bayes	15.48	14.95	0.15

Final model (RNN)

Finally we used RNN based algorithm for assigning the IT Tickets to the relevant groups. The overall process can be represented as below:



RNN based final model

Merging the "Short description" and "Description" and removing unnecessary Columns

Created a new column "Text" my margining the "Short description" and "Description" and then removed the unwanted columns - "Short description" , "Description" and "Caller"

```
dataset["Text"] = dataset["Short description"] + " " + dataset["Description"]

dataset.drop(["Short description","Description","Caller"],axis=1,inplace=True)

dataset
```

	Assignment group	Text
0	GRP_0	login issue -verified user details.(employee# ...
1	GRP_0	outlook _x000D_\n_x000D_\nreceived from: hmjdr...
2	GRP_0	cant log in to vpn _x000D_\n_x000D_\nreceived ...
3	GRP_0	unable to access hr_tool page unable to access...
4	GRP_0	skype error skype error

Removing Unwanted Characters

We tried different methods to remove the unwanted characters but the below method resulted in higher accuracy of the final model.

```
def removeString(data, regex):
    return data.str.lower().str.replace(regex.lower(), ' ')


def cleanDataset(dataset, columnsToClean, regexList):
    for column in columnsToClean:
        for regex in regexList:
            dataset[column] = removeString(dataset[column], regex)
    return dataset


def getRegexList():
    '''
    Adding regex list as per the given data set to flush off the unnecessary text

    '''
    regexList = []
    regexList += ['From:(.*)\r\n']  # from line
    regexList += ['Sent:(.*)\r\n']  # sent to line
    regexList += ['received from:(.*)\r\n']  # received data line
    regexList += ['received']  # received data line
    regexList += ['To:(.*)\r\n']  # to line
    regexList += ['CC:(.*)\r\n']  # cc line
    regexList += ['(.*)infection']  # footer
    regexList += ['\[cid:(.*)]']  # images cid
    regexList += ['https?:[^\]\n\r]+']  # https & http
    regexList += ['Subject:']
    regexList += ['[\w\d\-\_\.]+@[\w\d\-\_\.]+']  # emails are not required
    regexList += ['[0-9][\-0-90-9 ]+']  # phones are not required
    regexList += ['[0-9]']  # numbers not needed
    regexList += ['[^a-zA-z 0-9]+']  # anything that is not a letter
    regexList += ['[\r\n]']  # \r\n

    regexList += ['^[_a-z0-9-]+(\.[_a-z0-9-]+)*@[a-z0-9-]+(\.[a-z0-9-]+)*(\.[a-z]{2,4})$']
    regexList += ['[\w\d\-\_\.]+ @ [\w\d\-\_\.]+']
    regexList += ['Subject:']
    regexList += ['[^a-zA-Z]']

    regexList += [' [a-zA-Z] ']  # single letters makes no sense
    regexList += [' [a-zA-Z][a-zA-Z] ']  # two-letter words makes no sense
    regexList += ["  "]  # double spaces

    return regexList
```

```
columnsToClean = ['Text']

clean_dataset = cleanDataset(dataset, columnsToClean, getRegexList())

clean_dataset
```

	Assignment group	Text
0	GRP_0	login issue verified user details employee ma...
1	GRP_0	outlook d d from d hello team d d meetings s...
2	GRP_0	cant log to vpn d d from d d d cannot log to...
3	GRP_0	unable access tool page unable access tool page
4	GRP_0	skype error skype error

Language Translation to English

We are using langdetect to identify the language and GoogleTranslate to translate non-English rows to English.

```
[ ]  def fn_lan_detect(df):
         try:
             return detect(df)
         except:
             return 'no'

     clean_dataset['language'] =clean_dataset['Text'].apply(fn_lan_detect)

     clean_dataset["language"].value_counts()
```

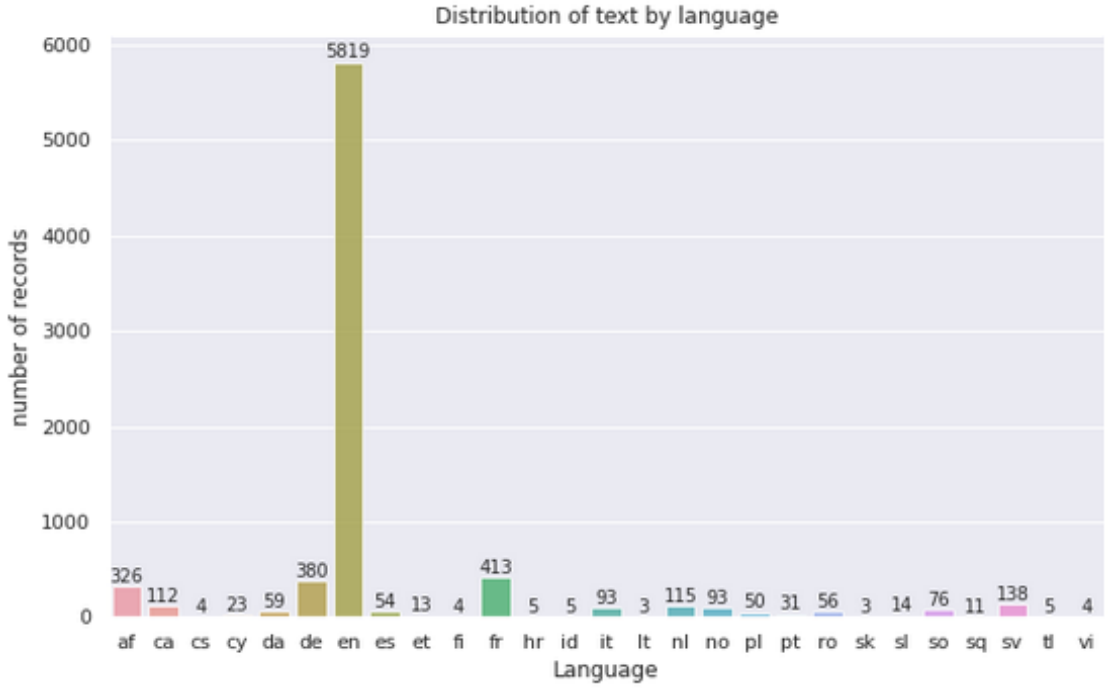
```
en    6169
sl     444
de     393
fr     318
af     305
da     162
no     155
sv     128
ca     104
it      72
```

```
[ ]  total = len(clean_dataset)

     for i, row in clean_dataset.iterrows():

       if (row["language"] != "en"):

         try :
           clean_dataset["Text"][i] = GoogleTranslator(source='auto', target='en').translate(row['Text'])
         except :
           clean_dataset["Text"][i]  = row["Text"]
       print(f"entrys completed : {i}/{total}",end="\r")
```

```
[ ]  clean_dataset.drop("language",axis=1,inplace=True)
     clean_dataset
```

	Assignment group	Text
0	GRP_0	login issue verified user details employee ma...
1	GRP_0	outlook d d from d hello team d d meetings s...
2	GRP_0	cant log to vpn d d from d d d cannot log to...
3	GRP_0	unable access tool page unable access tool page
4	GRP_0	skype error skype error

Distribution of Test by Language



Grouping the classes

We grouped all the groups having <100 counts to a common group called “GRP_X”.

```
[ ] clean_dataset.drop_duplicates(inplace = True)
    clean_dataset.fillna('', inplace=True)
```

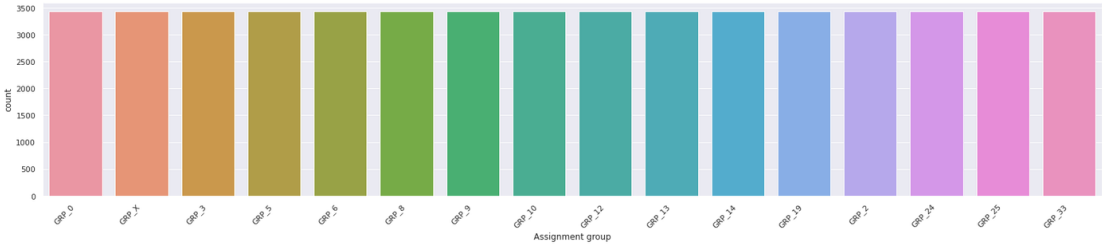
```
[ ] counts = clean_dataset["Assignment group"].value_counts()

    clean_dataset["Assignment group"] = np.where(counts[clean_dataset["Assignment group"]] < 100 , "GRP_X", clean_dataset["Assignment group"])

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

```
GRP_0     3198
GRP_X     1688
GRP_8      329
GRP_24     277
GRP_12     245
```

Oversampling

There is a huge imbalance in the data, so to reduce its effect, we did oversampling. Using resampling, we upsampled the data to match with the count of the most frequent group.

```
[ ] maxOthers =dataset_a["Assignment group"].value_counts().max()

    dataset_resampled = dataset_a[0:0]
    for grp in dataset_a['Assignment group'].unique():
        ticket_dataset_group = dataset_a[dataset_a['Assignment group'] == grp]
        resampled = resample(ticket_dataset_group, replace=True, n_samples=int(maxOthers), random_state=123)
        dataset_resampled = dataset_resampled.append(resampled)


    descending_order = dataset_resampled['Assignment group'].value_counts().sort_values(ascending=False).index
    plt.subplots(figsize=(22,5))

    ax=sns.countplot(x='Assignment group', data=dataset_resampled)
    ax.set_xticklabels(ax.get_xticklabels(), rotation=45, ha="right")
    plt.tight_layout()
    plt.show()
```



Encoding the Sentences and Labels

We are using keras Tokenizer to tokenize the rows and Label Encoder to encode the groups.

```
[ ] maxlen = 300
    numWords=9000
    epochs = 10
    batch_size = 100
```

```
[ ] tokenizer = Tokenizer(num_words=numWords, oov_token="<unk>", filters='!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n')
    tokenizer.fit_on_texts(dataset_resampled['Text'])

    train_seqs = tokenizer.texts_to_sequences(dataset_resampled['Text'])
```

We are padding the sequences to a length of 300

```
encoder = LabelEncoder()

X = pad_sequences(train_seqs, padding='post',maxlen=maxlen)

Y = dataset_resampled["Assignment group"].astype("category")
Y = encoder.fit_transform(Y)
```

Test Train Split

We did Test Train Split with test_size=0.20 and random_state =42

```
x_train, x_test, y_train, y_test = train_test_split(X,Y, test_size = 0.20, random_state = 42)
print(x_train.shape, y_train.shape )
print(x_test.shape,  y_test.shape  )
```

```
(30700, 300) (30700,)
(7676, 300) (7676,)
```

Model building

Final model we used is below

```
glove_file = '/content/drive/MyDrive/Colab Notebooks/capstone project/glove.6B.50d.txt'

embeddings_glove = {}
for o in open(glove_file):
    word = o.split(" ")[0]

    embd = o.split(" ")[1:]
    embd = np.asarray(embd, dtype='float32')

    embeddings_glove[word] = embd
```

```
embedding_matrix = np.zeros((numWords+1, 50))

for i,word in tokenizer.index_word.items():
  if i<numWords+1:
    embedding_vector = embeddings_glove.get(word)
    if embedding_vector is not None:
      embedding_matrix[i] = embedding_vector
```

```
embed = Embedding(numWords+1,output_dim=50,input_length=maxlen,weights=[embedding_matrix], trainable=True)

model=Sequential()
model.add(Input(shape=(maxlen,),dtype=tf.int64))
model.add(embed)
model.add(Conv1D(100,10,activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(Dropout(0.3))
model.add(Conv1D(100,10,activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(Bidirectional(LSTM(128)))
model.add(Dropout(0.3))
model.add(Dense(100,activation='relu'))
model.add(Dense(len((pd.Series(y_train)).unique()),activation='softmax'))
model.compile(loss='sparse_categorical_crossentropy',optimizer="adam",metrics=['accuracy'])

model.summary()


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

model_history = model.fit(x_train,y_train,batch_size=batch_size, epochs=epochs, callbacks=[checkpoint,reduceLoss], validation_data=(x_test, y_test))
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 300, 50)           450050

 conv1d (Conv1D)             (None, 291, 100)          50100

 max_pooling1d (MaxPooling1D  (None, 145, 100)         0
 )

 dropout (Dropout)           (None, 145, 100)          0

 conv1d_1 (Conv1D)           (None, 136, 100)          100100

 max_pooling1d_1 (MaxPooling  (None, 68, 100)          0
 1D)

 bidirectional (Bidirectiona  (None, 256)              234496
 l)

 dropout_1 (Dropout)         (None, 256)               0

 dense (Dense)               (None, 100)               25700

 dense_1 (Dense)             (None, 12)                1212

=================================================================
Total params: 861,658
Trainable params: 861,658
Non-trainable params: 0
_________________________________________________________________
```

```
Epoch 1/10
307/307 [==============================] - ETA: 0s - loss: 1.3357 - accuracy: 0.5484
Epoch 1: val_accuracy improved from -inf to 0.83872, saving model to model-001-0.838718.h5
307/307 [==============================] - 34s 69ms/step - loss: 1.3357 - accuracy: 0.5484 - val_loss: 0.4949 - val_accuracy: 0.8387 - lr: 0.0010
Epoch 2/10
307/307 [==============================] - ETA: 0s - loss: 0.3712 - accuracy: 0.8747
Epoch 2: val_accuracy improved from 0.83872 to 0.92092, saving model to model-002-0.920922.h5
307/307 [==============================] - 20s 65ms/step - loss: 0.3712 - accuracy: 0.8747 - val_loss: 0.2309 - val_accuracy: 0.9209 - lr: 0.0010
Epoch 3/10
307/307 [==============================] - ETA: 0s - loss: 0.2039 - accuracy: 0.9301
Epoch 3: val_accuracy improved from 0.92092 to 0.94203, saving model to model-003-0.942027.h5
307/307 [==============================] - 20s 65ms/step - loss: 0.2039 - accuracy: 0.9301 - val_loss: 0.1781 - val_accuracy: 0.9420 - lr: 0.0010
Epoch 4/10
307/307 [==============================] - ETA: 0s - loss: 0.1409 - accuracy: 0.9508
Epoch 4: val_accuracy improved from 0.94203 to 0.95922, saving model to model-004-0.959224.h5
307/307 [==============================] - 20s 65ms/step - loss: 0.1409 - accuracy: 0.9508 - val_loss: 0.1144 - val_accuracy: 0.9592 - lr: 0.0010
Epoch 5/10
307/307 [==============================] - ETA: 0s - loss: 0.1063 - accuracy: 0.9633
Epoch 5: val_accuracy improved from 0.95922 to 0.95935, saving model to model-005-0.959354.h5
307/307 [==============================] - 20s 65ms/step - loss: 0.1063 - accuracy: 0.9633 - val_loss: 0.1243 - val_accuracy: 0.9594 - lr: 0.0010
Epoch 6/10
307/307 [==============================] - ETA: 0s - loss: 0.0855 - accuracy: 0.9706
Epoch 6: val_accuracy improved from 0.95935 to 0.96704, saving model to model-006-0.967040.h5
307/307 [==============================] - 20s 65ms/step - loss: 0.0855 - accuracy: 0.9706 - val_loss: 0.1045 - val_accuracy: 0.9670 - lr: 0.0010
Epoch 7/10
307/307 [==============================] - ETA: 0s - loss: 0.0673 - accuracy: 0.9778
Epoch 7: val_accuracy improved from 0.96704 to 0.96756, saving model to model-007-0.967561.h5
307/307 [==============================] - 20s 65ms/step - loss: 0.0673 - accuracy: 0.9778 - val_loss: 0.1066 - val_accuracy: 0.9676 - lr: 0.0010
Epoch 8/10
307/307 [==============================] - ETA: 0s - loss: 0.0655 - accuracy: 0.9777
Epoch 8: val_accuracy improved from 0.96756 to 0.97199, saving model to model-008-0.971991.h5
307/307 [==============================] - 20s 65ms/step - loss: 0.0655 - accuracy: 0.9777 - val_loss: 0.0882 - val_accuracy: 0.9720 - lr: 0.0010
Epoch 9/10
307/307 [==============================] - ETA: 0s - loss: 0.0530 - accuracy: 0.9821
Epoch 9: val_accuracy improved from 0.97199 to 0.97551, saving model to model-009-0.975508.h5
307/307 [==============================] - 20s 65ms/step - loss: 0.0530 - accuracy: 0.9821 - val_loss: 0.0856 - val_accuracy: 0.9755 - lr: 0.0010
Epoch 10/10
307/307 [==============================] - ETA: 0s - loss: 0.0528 - accuracy: 0.9819
Epoch 10: val_accuracy improved from 0.97551 to 0.97681, saving model to model-010-0.976811.h5
307/307 [==============================] - 20s 65ms/step - loss: 0.0528 - accuracy: 0.9819 - val_loss: 0.0818 - val_accuracy: 0.9768 - lr: 0.0010
```

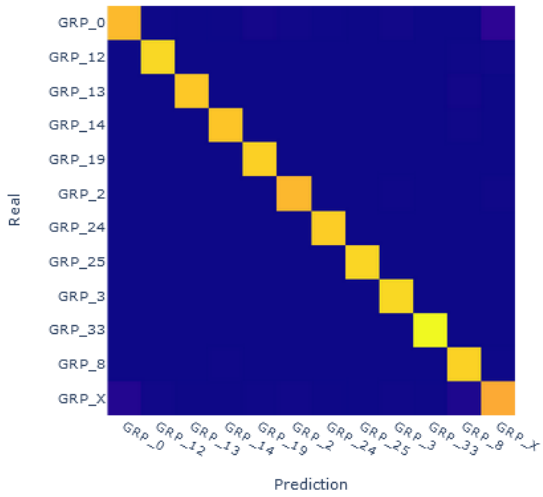
Checking Model Accuracy

```
y_pred = model.predict(x_test )

pred =[]
for val in y_pred:
  pred.append(np.argmax(val))
```

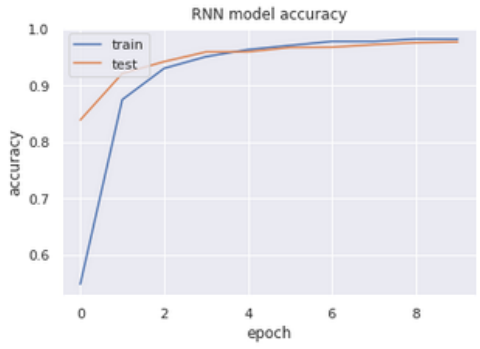
```
test_acc  = metrics.accuracy_score( y_test , pred )

con_mat = metrics.confusion_matrix(y_test , pred)


print("Test  Accuracy -> ",round(test_acc  *100,2),"%")
print("F1 score       -> ",metrics.f1_score( y_test , pred, average='micro'))

print("\n")



fig = px.imshow(con_mat,labels=dict(x="Prediction", y="Real", color="Count"),
                x=encoder.classes_,
                y=encoder.classes_)
fig.show()
```

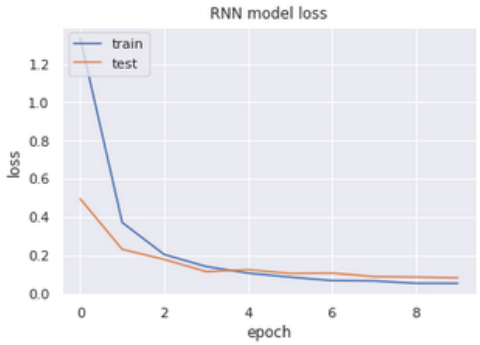
```
Test  Accuracy ->  97.68 %
F1 score       ->  0.9768108389786347
```

Heatmap Prediction Vs Real




```
plt.plot(model_history.history['accuracy'])
plt.plot(model_history.history['val_accuracy'])

plt.title('RNN model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train','test'], loc='upper left')
plt.show()

plt.plot(model_history.history['loss'])
plt.plot(model_history.history['val_loss'])

plt.title('RNN model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train','test'], loc='upper left')
plt.show()
```





Evaluation of the Final Model (RNN)

Train Accuracy	**: 98.97%**
Test Accuracy	**: 97.64%**
F1 score	**: 0.98**

Saving the model

The model is saved for further usage and User Interface (UI) development.

```
[ ] import joblib

    model.save("/content/drive/MyDrive/Colab Notebooks/capstone project/final/rnn_final.h5")
    joblib.dump(tokenizer, '/content/drive/MyDrive/Colab Notebooks/capstone project/final/tokenizer.pkl')
    joblib.dump(encoder, '/content/drive/MyDrive/Colab Notebooks/capstone project/final/label_encoder.pkl')

    ['/content/drive/MyDrive/Colab Notebooks/capstone project/final/label_encoder.pkl']
```

5. Comparison to benchmark

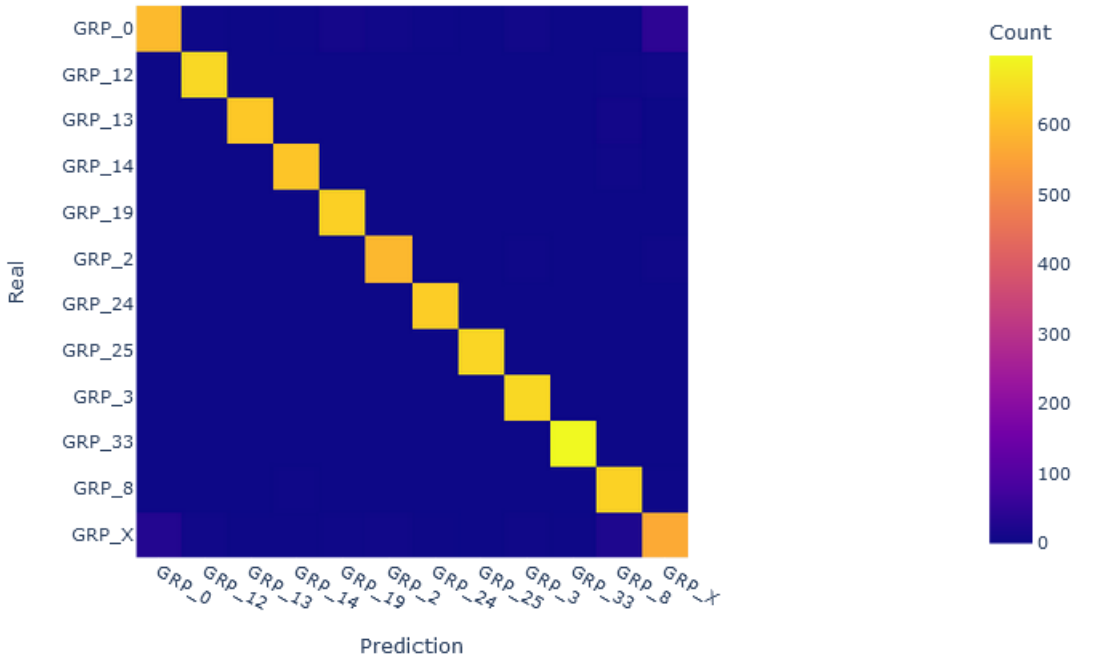
After performing a series of iterations on various models we tabulated the result to a tabular form based on the descending order of test accuracy

Sorted in decending order of Test Accuracy				
#	Model	Train Accuracy (%)	Test Accuracy (%)	F1 Score
	RNN (After fine-tuning)	98.80	97.60	0.98
1	Random Forest	92.19	90.90	0.91
2	RNN (Initial result)	91.07	90.68	0.89
3	Bagging	92.19	90.50	0.91
4	Decision Tree	92.19	90.04	0.90
5	KNN Classifier	92.05	89.25	0.89
6	Gradient Boosting	74.02	71.70	0.72
7	SVM	63.45	61.53	0.62
8	Logistic Regression	27.43	26.65	0.27
9	ADA Boosting	23.05	23.11	0.23
10	SGD Classifier	19.90	19.23	0.19
11	Naive Bayes	15.48	14.95	0.15

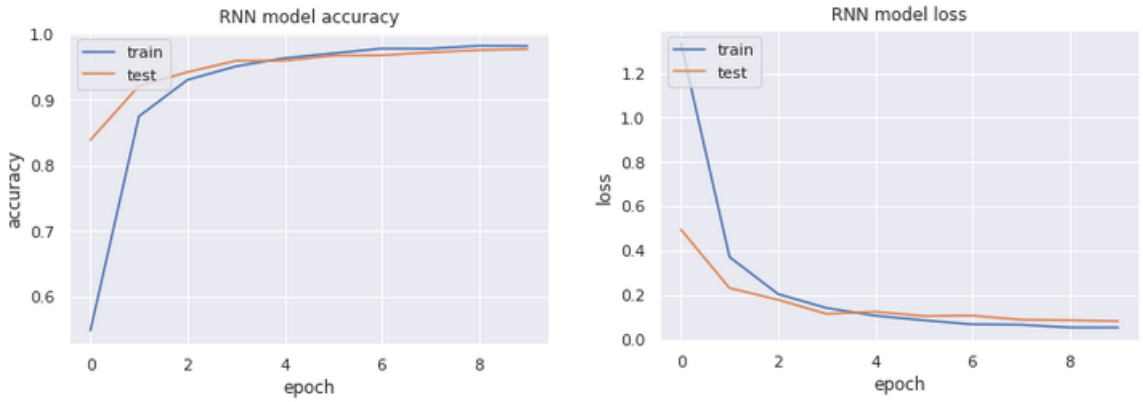
We focused on the first five classifiers as all of them are closer to or above 90% test accuracy and also there is no significant difference between test and train accuracy for each of them. Also, their f1 scores are reasonably high. While doing fine tuning RNN classifier outperformed the others so we developed our Automatic ticketing system based on RNN classifier. There is a huge imbalance in the class which can lead to overfitting so same algorithm with more balanced data will be suitable for our system.

Before starting our experimentation of models, we took 92.8% test accuracy as our benchmark form internet search. And our RNN model out performed it with a test accuracy of 97.6% and train accuracy of 98.8 and an f1 score of 0.98 so even though we had a doubt of overfitting we proceeded with the model to develop our Automatic Ticketing System as it can be improved with more balanced data for training.

Test prediction heat map of RNN:



Accuracy and loss graph of RNN:



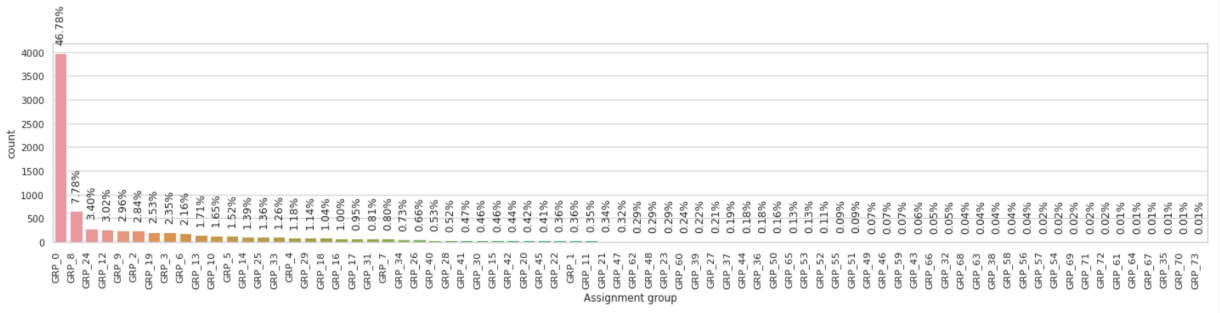
6. Visualizations

Visualization of Raw Data (Given dataset):

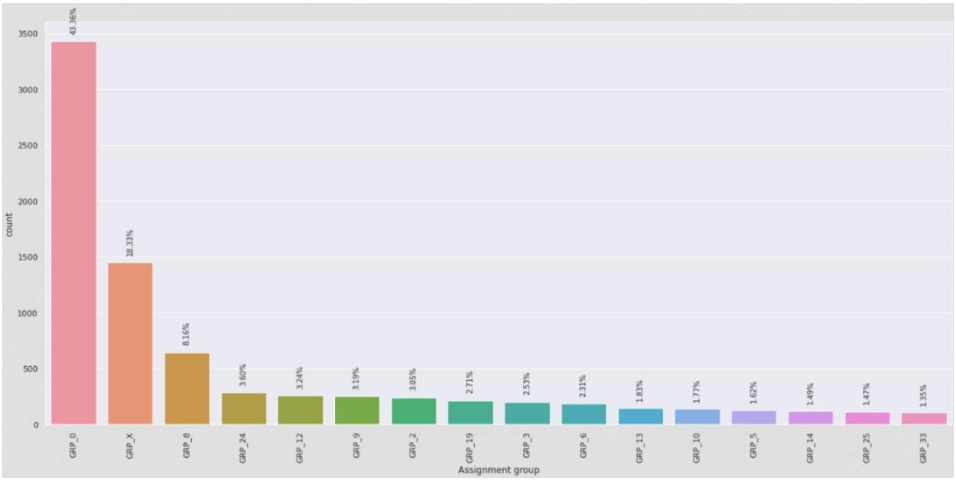
Data structure

	Short description	Description	Caller	Assignment group
7526	not possible to login due to a locked account	not possible to login due to a locked account	qufjnslk bvtfrwnu	GRP_0
3709	do not have permission to ess portal. at	please reach out to me for more information if...	ytwmgpbk cpawsihk	GRP_2
3237	erp accout had been locked	_x000D_\n_x000D_\nreceived from: qyidkvap.cxnf...	qyidkvap cxnfdjpk	GRP_0
7956	unable to login to vpn and reset password	unable to login to vpn and reset password	xiaurwpz hzusljmc	GRP_0
5751	when i create a recurring appointment in mds i...	when i create a recurring appointment in mds i...	acteiqdu bferalus	GRP_40
5050	unable to launch outlook	unable to launch outlook	vjwdyanl knfsjdgz	GRP_0

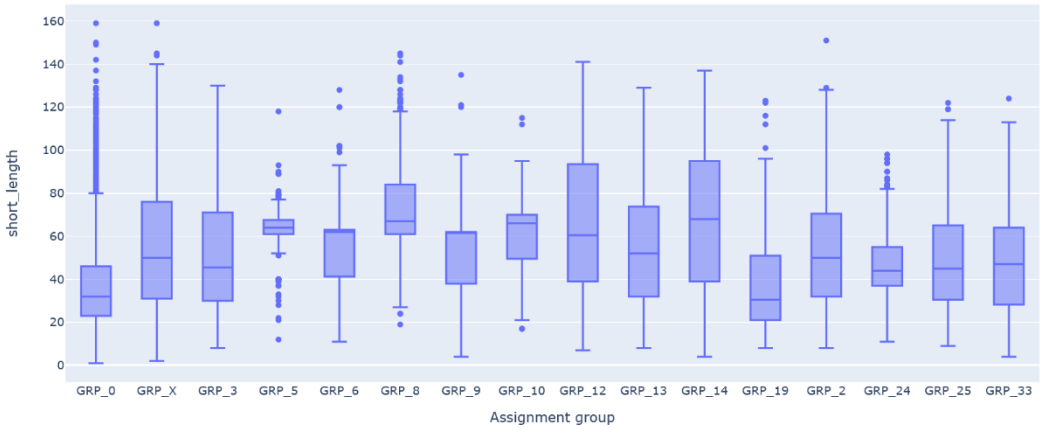
Distribution of Raw Classes



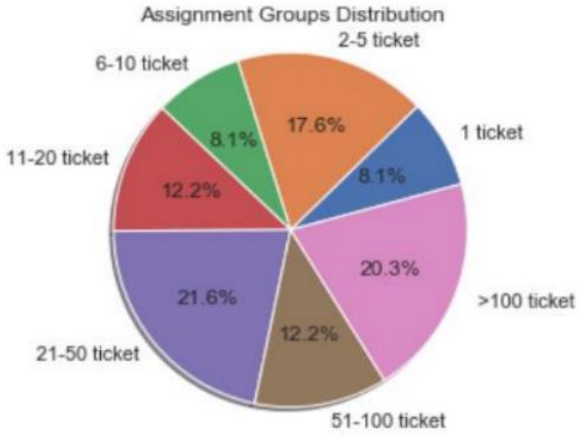
Assignment group Vs Count



Assignment group Vs short description length

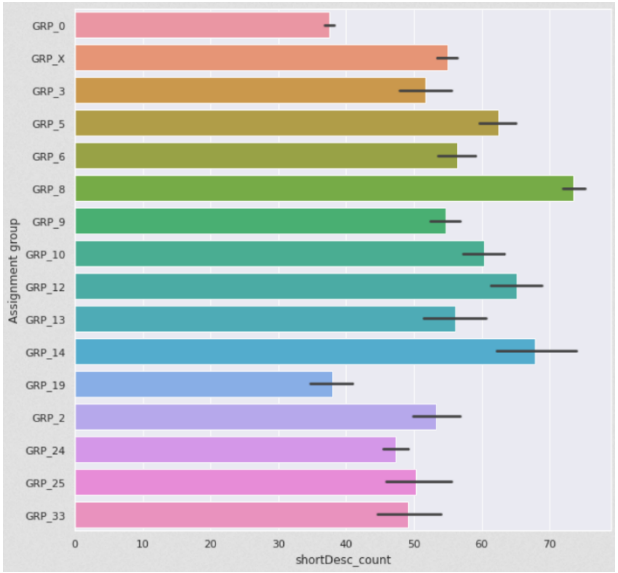


Distribution of ticket counts in various bins as below



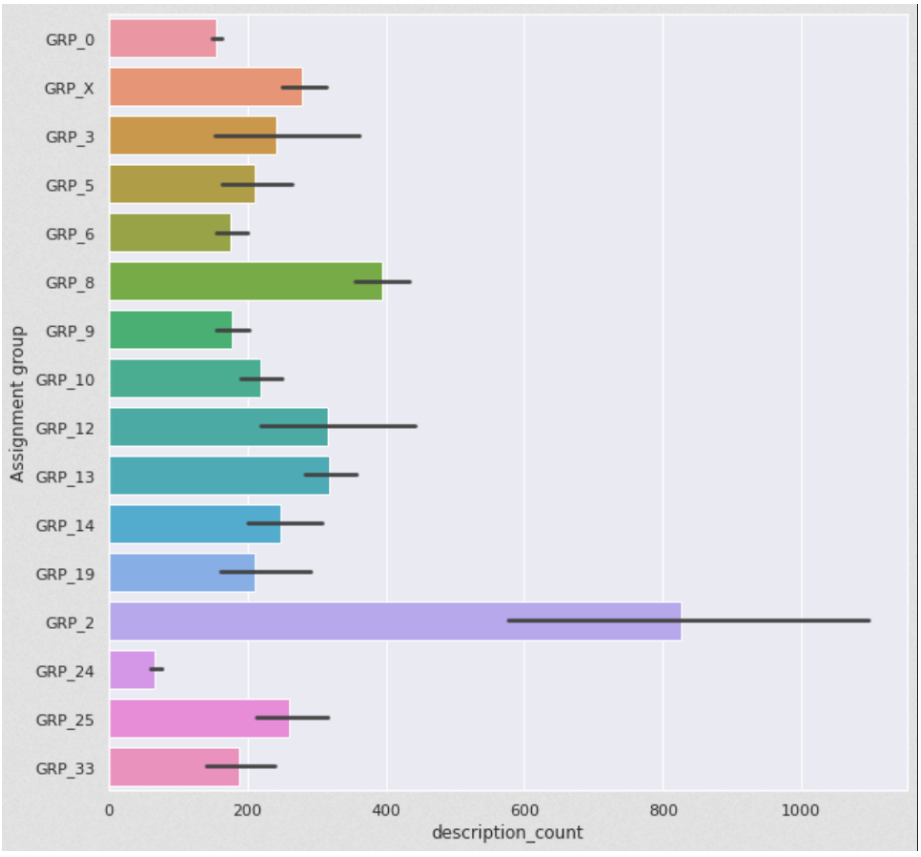
Visualizing different text features

Short description count Vs Assignment



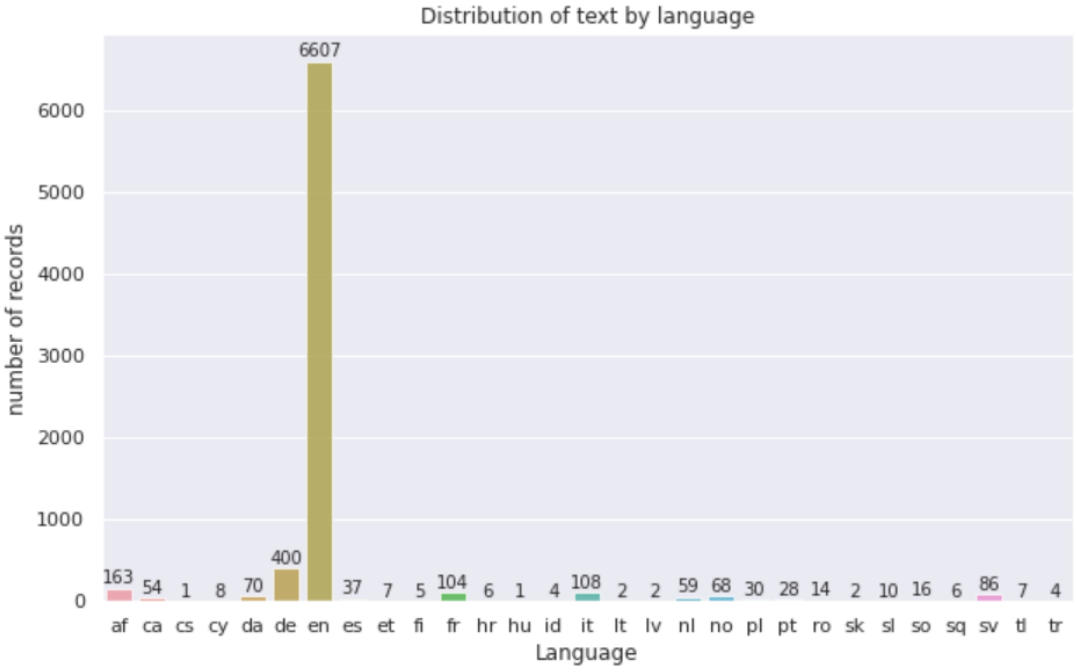
Word cloud image – Assignment group Vs short description

Description count Vs Assignment group



Word cloud image – Assignment group Vs Description

Distribution of text by language – Language Vs number of records



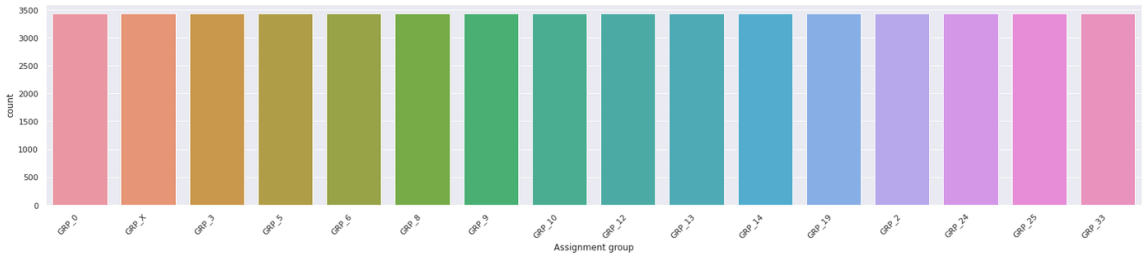
Visualization of Processed Data

Visualization of the dataset after preprocessing

	Assignment group	Text
0	GRP_0	login issue verify user detail employee manage...
1	GRP_0	outlook x n x nreceived hmjdrvpb komuaywn gmai...
2	GRP_0	cant log vpn x n x nreceived eylqgodm ybqkwiam...
3	GRP_0	unable access hr tool page unable access hr to...
4	GRP_0	skype error skype error

Visualization of the text column after prepressing

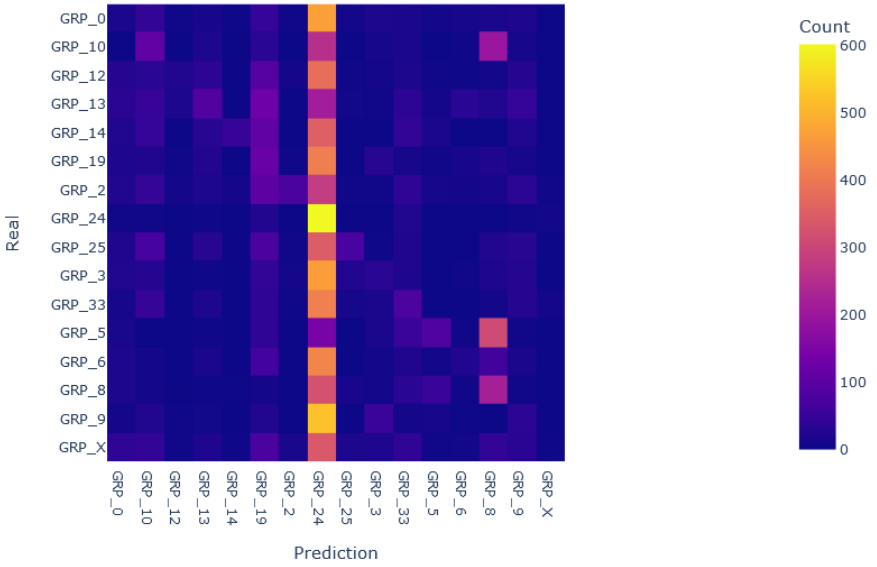
Visualizing the data after taking care of the unbalanced data



Visualization of the models:

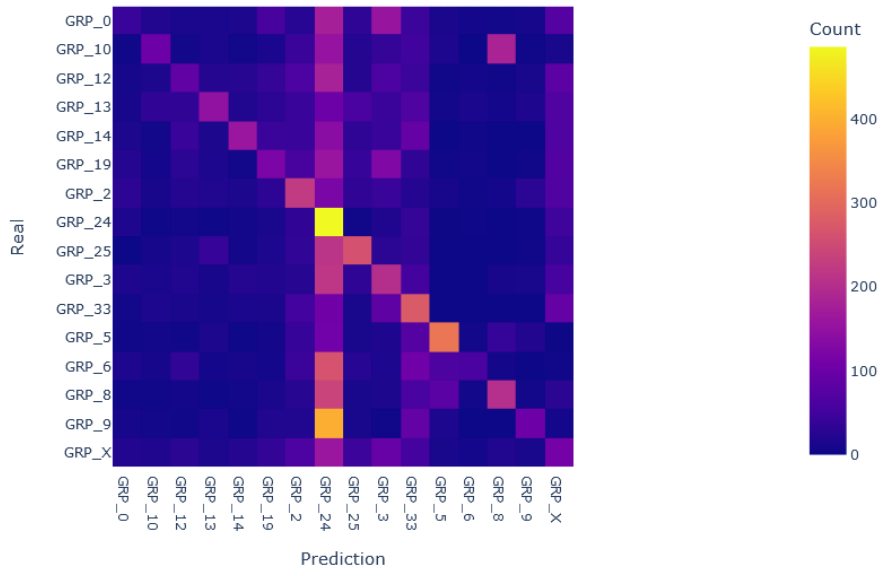
Naive Bays

```
Train Accuracy ->  15.33 %
Test  Accuracy ->  14.78 %
F1 score       ->  0.14781736990795588
```



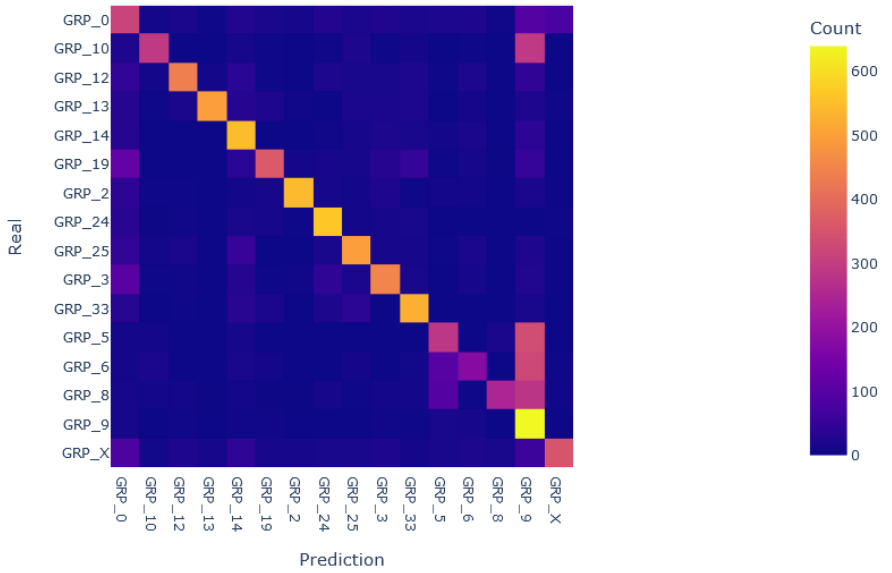
Logistic Regression

```
Train Accuracy ->  27.14 %
Test  Accuracy ->  26.42 %
F1 score       ->  0.2641939305568213
```



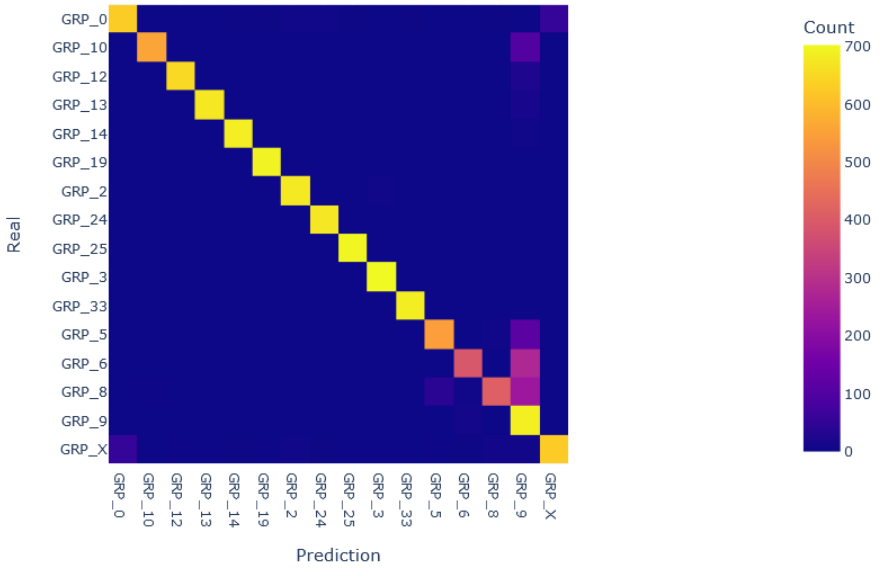
SVM

```
Train Accuracy ->  63.29 %
Test  Accuracy ->  61.28 %
F1 score       ->  0.6127768158206507
```



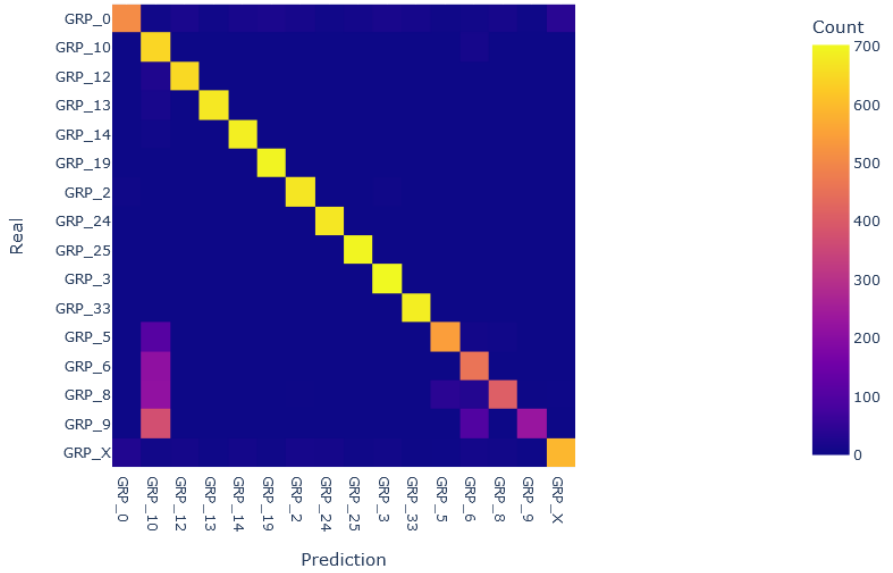
Random Forest

```
Train Accuracy ->  92.19 %
Test  Accuracy ->  90.96 %
F1 score       ->  0.9095962817825571
```



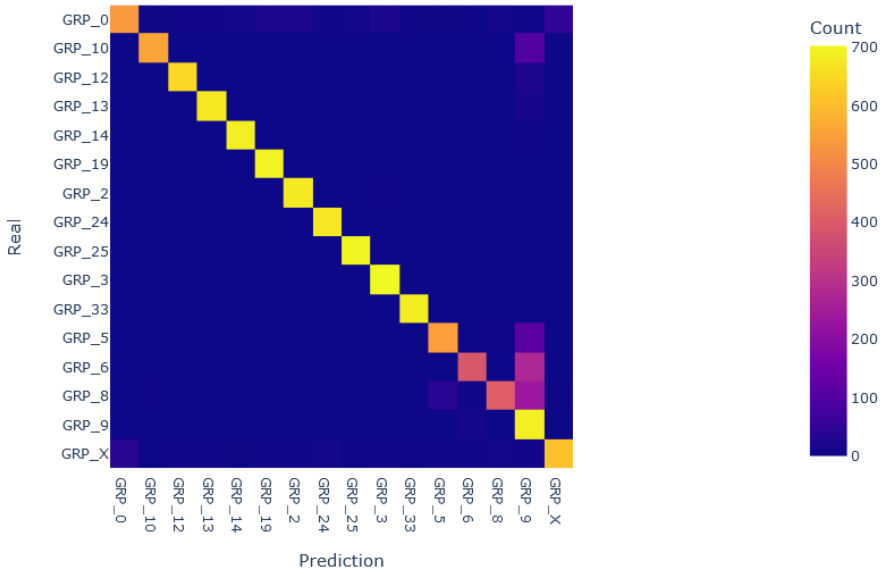
KNN Classifier

```
Train Accuracy ->  89.73 %
Test  Accuracy ->  86.69 %
F1 score       ->  0.8669461405267475
```



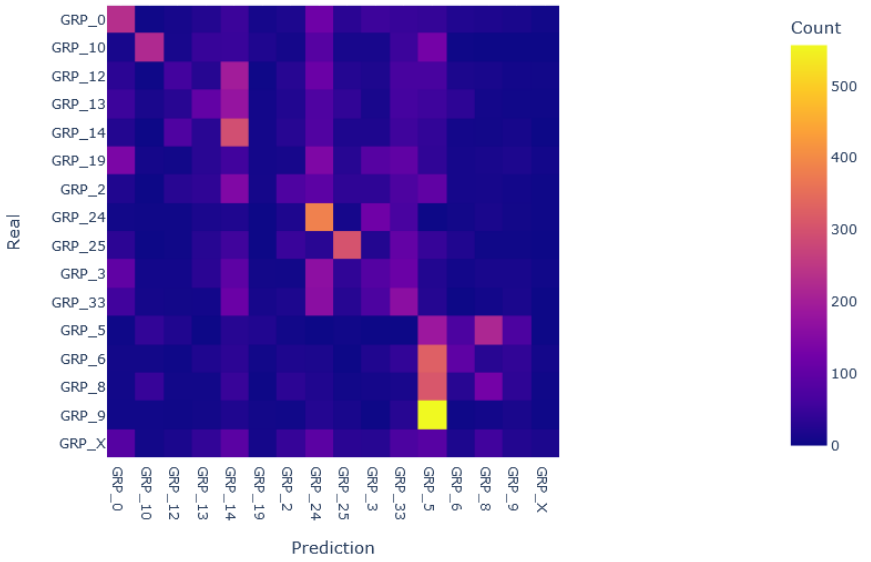
Decision Tree

```
Train Accuracy ->  92.19 %
Test  Accuracy ->  89.92 %
F1 score       ->  0.8992071448099882
```



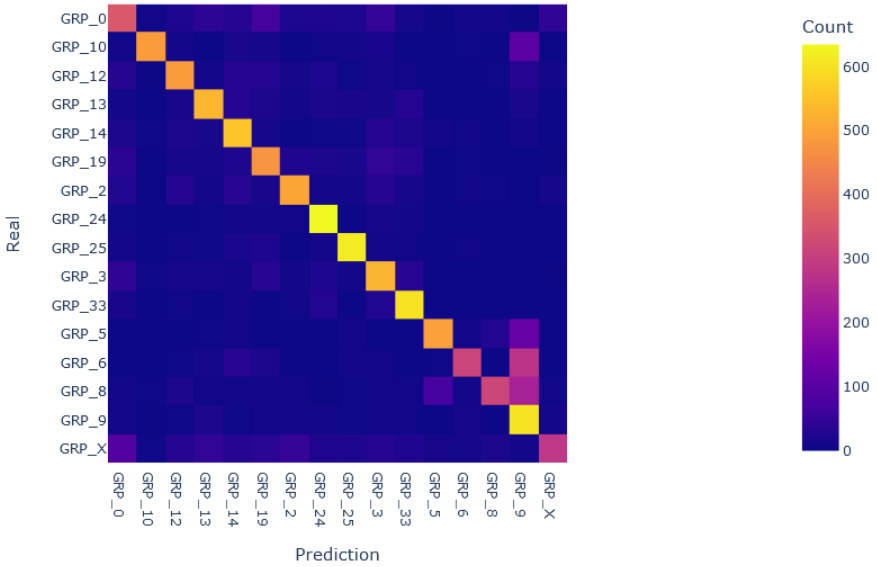
ADA Boosting

```
Train Accuracy ->  21.43 %
Test  Accuracy ->  21.47 %
F1 score       ->  0.2147088307664267
```



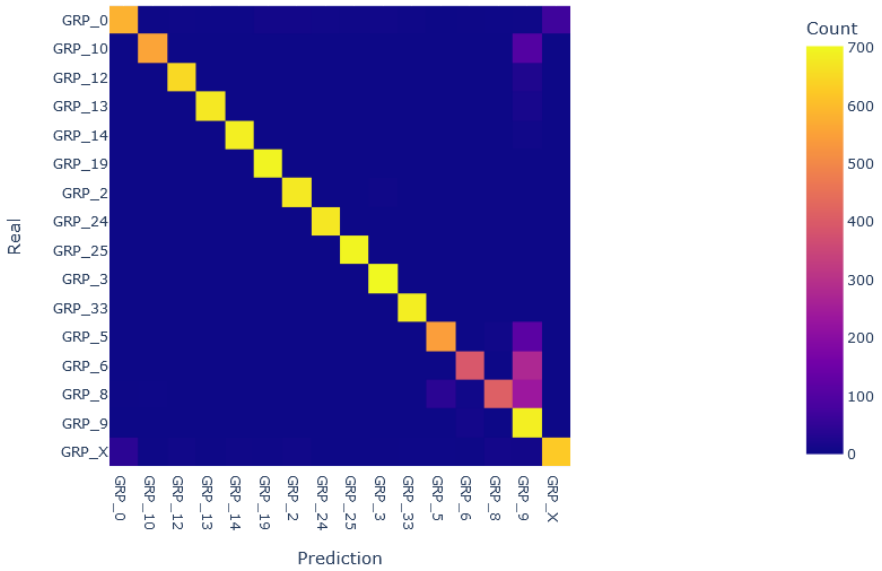
Gradient Boosting

```
Train Accuracy ->  73.28 %
Test  Accuracy ->  71.05 %
F1 score       ->  0.710471156474984
```



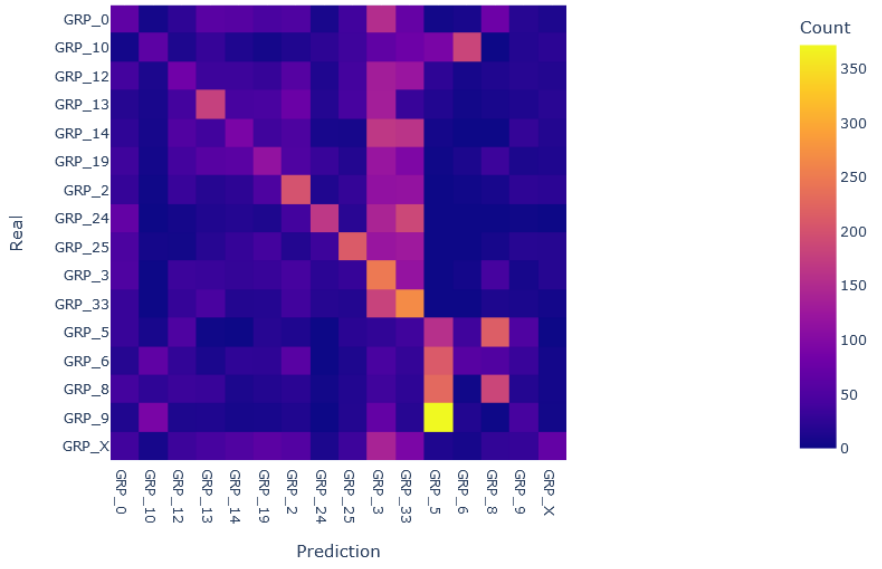
Bagging

```
Train Accuracy ->  92.19 %
Test  Accuracy ->  90.5 %
F1 score       ->  0.9050396427595006
```



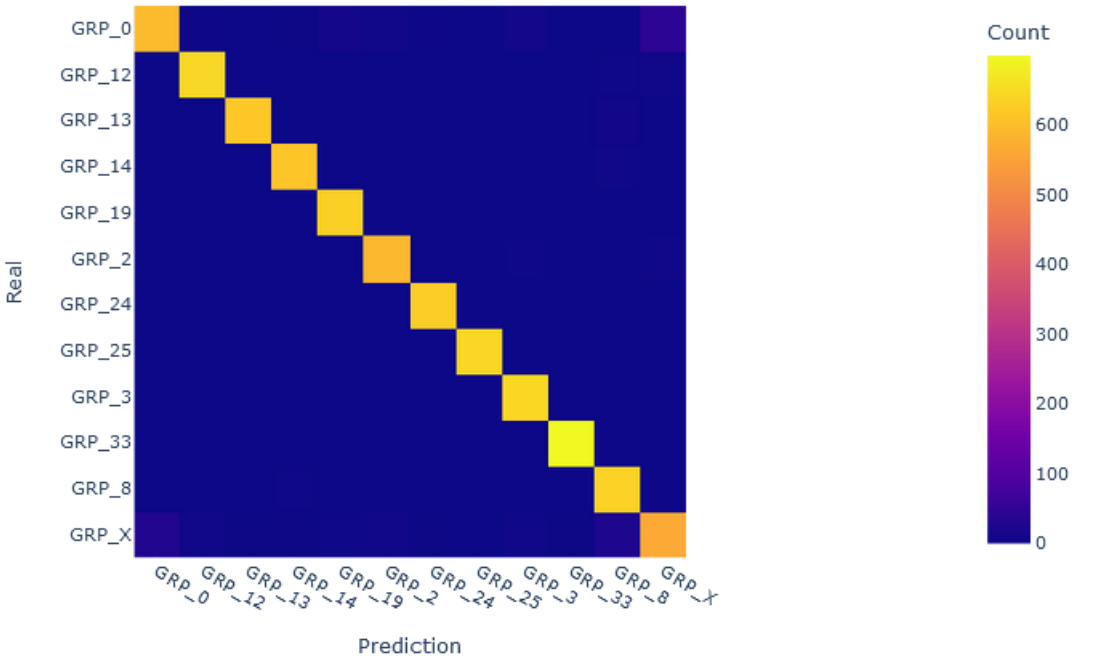
SGD Classifier

```
Train Accuracy ->  20.07 %
Test  Accuracy ->  19.99 %
F1 score       ->  0.19994532033172333
```

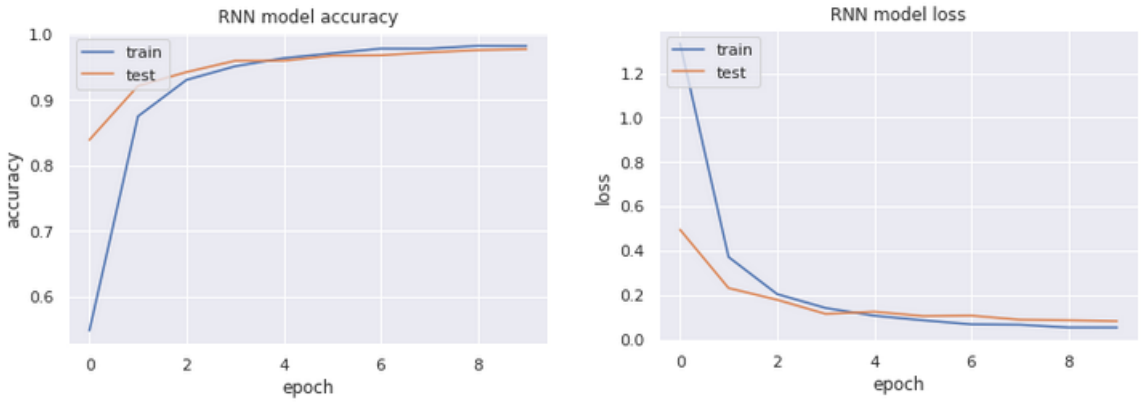


Visualization of final RNN model:

Test prediction heat map of RNN:



Accuracy and loss graph of RNN:



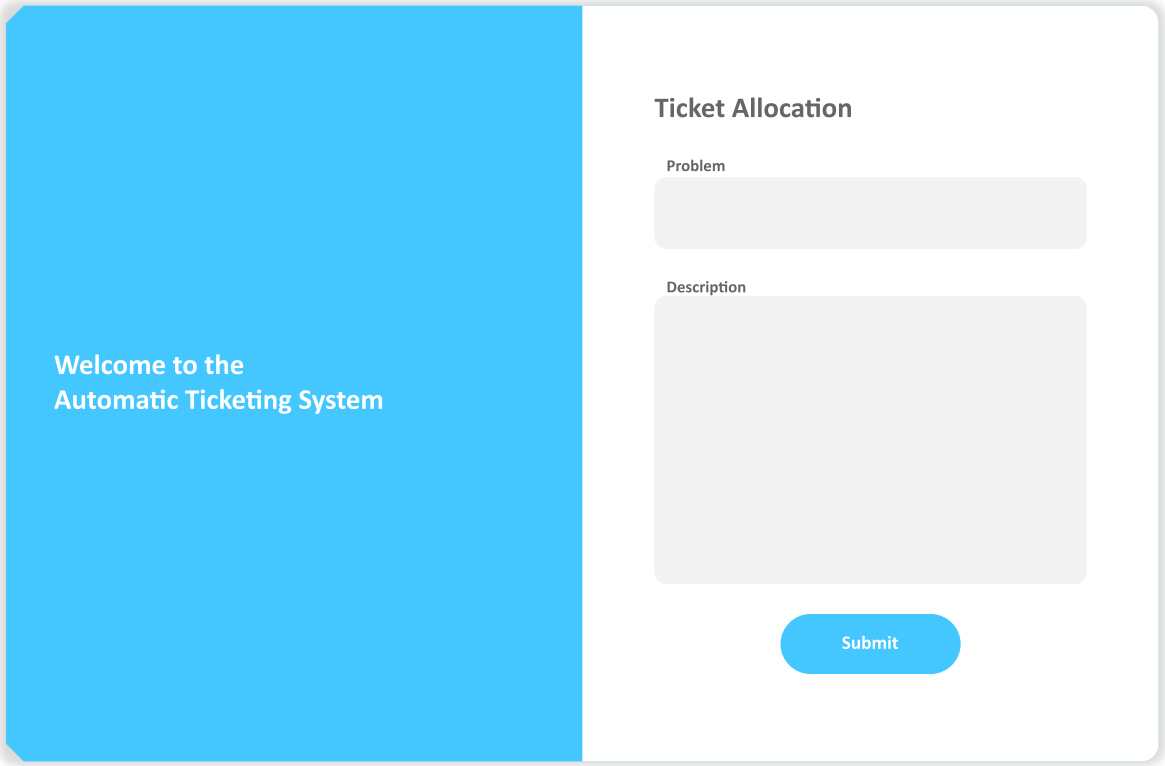
7. User Interface Development

For easy entry of problem and Description of an IT ticket by a user we developed a simple User Interface (UI) which will use our developed RNN model to assign the Tickets to the respective groups. For this we use the saved model and encoder files.

UI Development:

We used Figma to create the layout of the UI as shown below

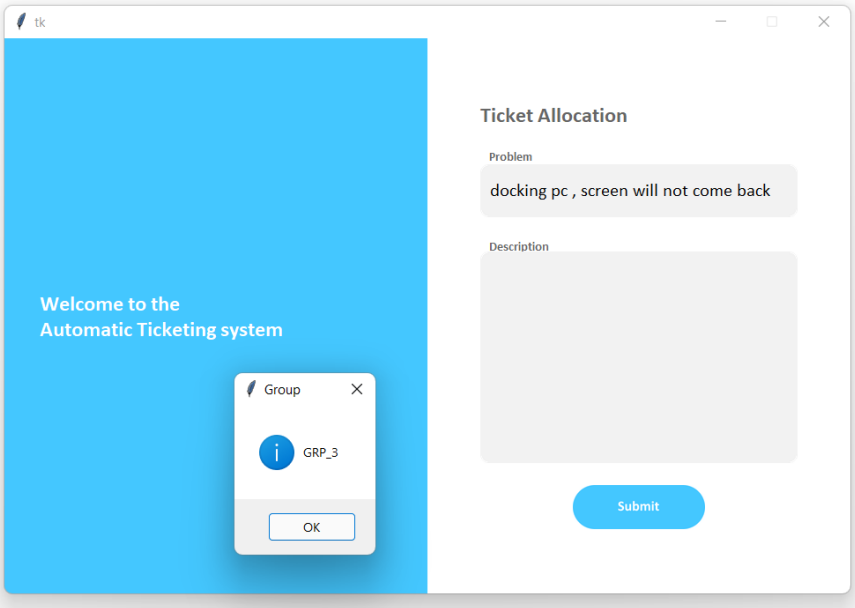
https://www.figma.com

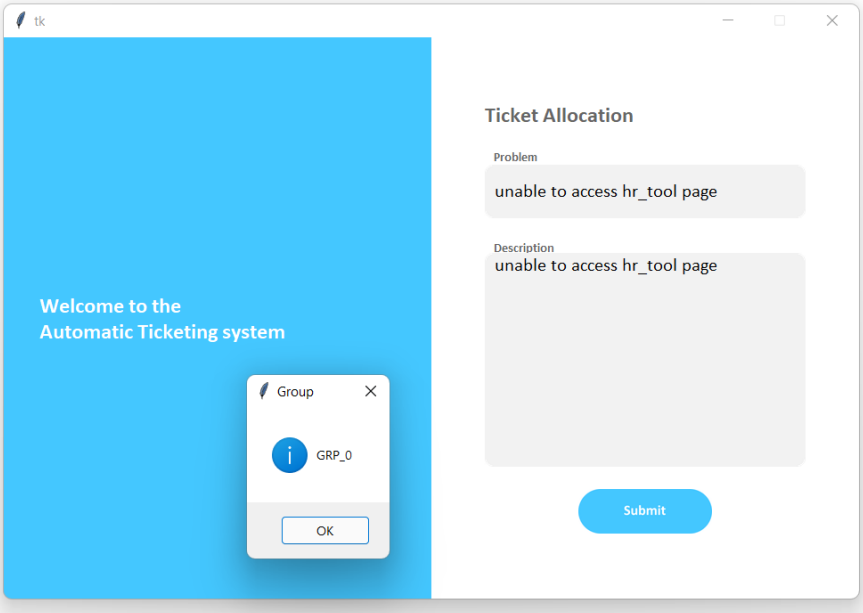


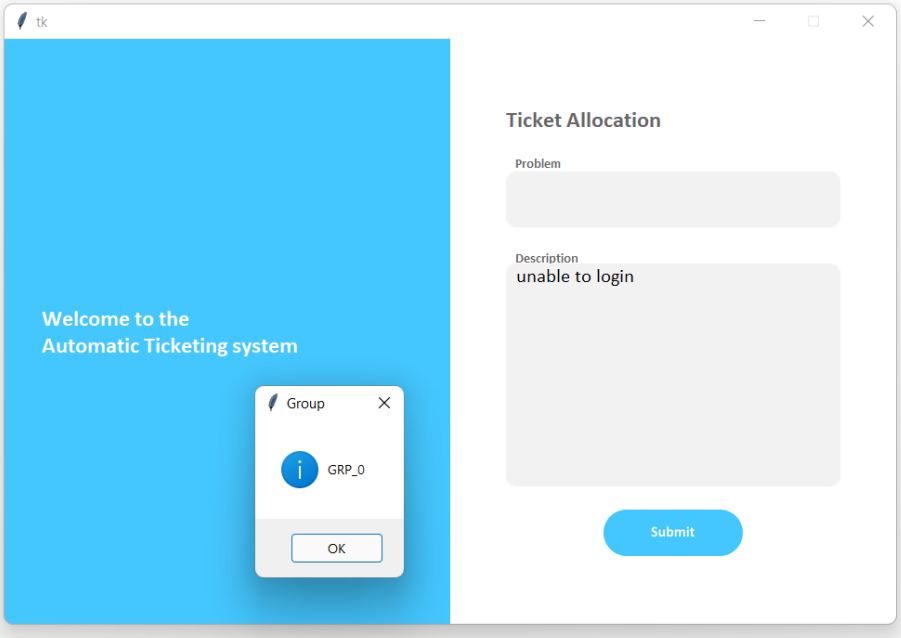
We used Tkinter-Designer package to convert the Figma file to a Tkinter based python code.

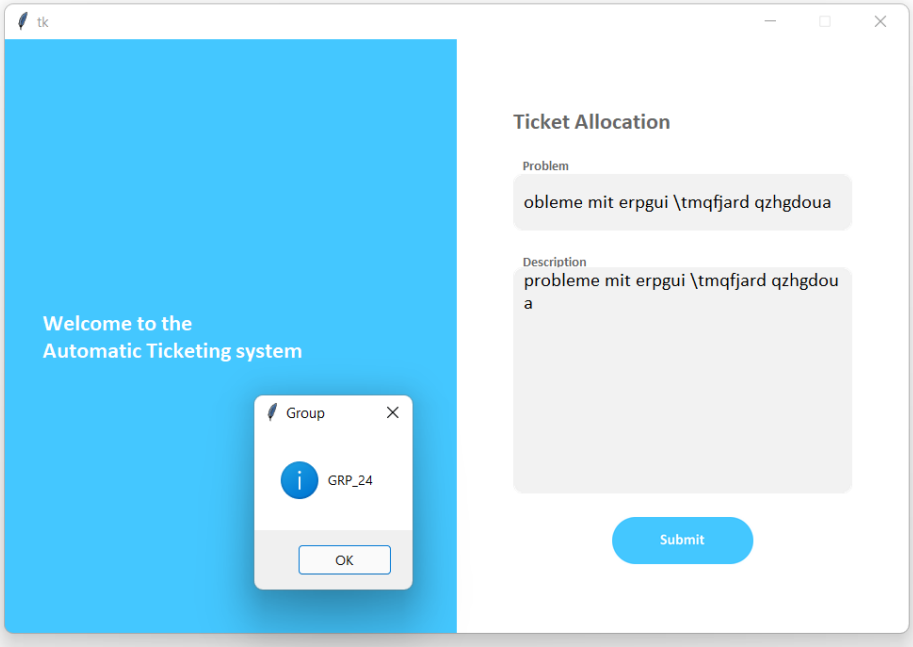
https://github.com/ParthJadhav/Tkinter-Designer

Few Trials using the ATS App we developed:






tk
Welcome to the
Automatic Ticketing system
Group
GRP_0
OK
Ticket Allocation
Problem
Description
unable to login
Submit


tk
Welcome to the
Automatic Ticketing system
Group
GRP_24
OK
Ticket Allocation
Problem
obleme mit erpgui \tmqfjard qzhgdoua
Description
probleme mit erpgui \tmqfjard qzhgdou
a
Submit

File 1 : gui.py – this file contains the generated UI code (we run this file to start the app)

```
1
2  # This file was generated by the Tkinter Designer by Parth Jadhav
3  # https://github.com/ParthJadhav/Tkinter-Designer
4
5
6  from pathlib import Path
7  import ctypes
8
9  # from tkinter import *
10 # Explicit imports to satisfy Flake8
11 from tkinter import Tk, Canvas, Entry, Text, Button, PhotoImage
12
13 from scripts import predict
14
15
16 OUTPUT_PATH = Path(__file__).parent
17 ASSETS_PATH = OUTPUT_PATH / Path("./assets")
18
19
20 def relative_to_assets(path: str) -> Path:
21     return ASSETS_PATH / Path(path)
22
23 ctypes.windll.shcore.SetProcessDpiAwareness(1)
24 window = Tk()
25
26 window.geometry("960x630")
27 window.configure(bg = "#44C7FF")
28
29
30 canvas = Canvas(
31     window,
32     bg = "#44C7FF",
33     height = 630,
34     width = 960,
35     bd = 0,
36     highlightthickness = 0,
37     relief = "ridge"
38 )
39
40 canvas.place(x = 0, y = 0)
41 canvas.create_rectangle(
42     480.0,
43     7.105427357601002e-15,
44     960.0,
45     630.0,
46     fill="#FFFFFF",
47     outline="")
48
49 canvas.create_text(
50     540.0,
51     72.0,
52     anchor="nw",
53     text="Ticket Allocation",
54     fill="#666666",
55     font=("Calibri Bold", 24 * -1)
56 )
57
58 entry_image_1 = PhotoImage(
59     file=relative_to_assets("entry_1.png"))
60 entry_bg_1 = canvas.create_image(
61     720.0,
62     173.0,
63     image=entry_image_1
64 )
65 entry_1 = Entry(
66     bd=0,
67     bg="#F2F2F2",
68     highlightthickness=0,
69     font=("Calibri", 20 * -1)
70
71 )
```

```
entry_1.place(
    x=550.0,
    y=143.0,
    width=340.0,
    height=58.0
)

canvas.create_text(
    550.0,
    126.0,
    anchor="nw",
    text="Problem",
    fill="#666666",
    font=("Calibri Bold", 14 * -1)
)

entry_image_2 = PhotoImage(
    file=relative_to_assets("entry_2.png"))
entry_bg_2 = canvas.create_image(
    720.0,
    362.0,
    image=entry_image_2
)
entry_2 = Text(
    bd=0,
    bg="#F2F2F2",
    highlightthickness=0,
    font=("Calibri", 20 * -1)

)
entry_2.place(
    x=550.0,
    y=242.0,
    width=340.0,
    height=238.0
)

canvas.create_text(
    550.0,
    228.0,
    anchor="nw",
    text="Description",
    fill="#666666",
    font=("Calibri Bold", 14 * -1)
)

button_image_1 = PhotoImage(
    file=relative_to_assets("button_1.png"))
button_1 = Button(
    image=button_image_1,
    borderwidth=0,
    highlightthickness=0,
    command=lambda: predict(entry_1,entry_2),
    relief="flat"
)
button_1.place(
    x=645.0,
    y=507.0,
    width=150.0,
    height=50.0
)

canvas.create_text(
    40.0,
    286.0,
    anchor="nw",
    text="Welcome to the\nAutomatic Ticketing system",
    fill="#FFFFFF",
    font=("Calibri Bold", 24 * -1)
)
window.resizable(False, False)
window.mainloop()

```

File 2: scripts.py – contains the predict function

```
from json import encoder
import joblib
from preprocessing import clean_text, translate
from keras.preprocessing.sequence import pad_sequences
from keras.models import load_model
import numpy as np
import tkinter as tk
from tkinter import messagebox

model = load_model('models/rnn_final.h5')
tokenizer = joblib.load('models/tokenizer.pkl')
encoder = joblib.load('models/label_encoder.pkl')



def predict(short, desc):

    raw = short.get() + " " + desc.get("1.0", tk.END)

    clean = translate(clean_text(raw))

    sequence = tokenizer.texts_to_sequences([clean])

    entry = pad_sequences(sequence, padding='post',maxlen=300)

    pred = np.argmax( model.predict(entry))


    pred_str = encoder.inverse_transform([pred])[0]

    print(pred_str)

    messagebox.showinfo("Group", pred_str)

```

File 3: preprocessing.py – contains the preprocessing functions

```python
from langdetect import detect
from deep_translator import GoogleTranslator

def removeString(data, regex):
    return data.lower().replace(regex.lower(), ' ')


def getRegexList():

    regexList = []
    regexList += ['From:(.*)\r\n']  # from line
    regexList += ['Sent:(.*)\r\n']  # sent to line
    regexList += ['received from:(.*)\r\n']  # received data line
    regexList += ['received']  # received data line
    regexList += ['To:(.*)\r\n']  # to line
    regexList += ['CC:(.*)\r\n']  # cc line
    regexList += ['(.*)infection']  # footer
    regexList += ['\[cid:(.*)]']  # images cid
    regexList += ['https?:[^\]\n\r]+']  # https & http
    regexList += ['Subject:']
    regexList += ['[\w\d\-\_\.]+@[\w\d\-\_\.]+']  # emails are not required
    regexList += ['[0-9][\-0-90-9 ]+']  # phones are not required
    regexList += ['[0-9]']  # numbers not needed
    regexList += ['[^a-zA-z 0-9]+']  # anything that is not a letter
    regexList += ['[\r\n]']  # \r\n

    regexList += ['^[_a-z0-9-]+(\.[_a-z0-9-]+)*@[a-z0-9-]+(\.[a-z0-9-]+)*(\.[a-z]{2,4})$']
    regexList += ['[\w\d\-\_\.]+ @ [\w\d\-\_\.]+']
    regexList += ['Subject:']
    regexList += ['[^a-zA-Z]']

    regexList += [' [a-zA-Z] ']  # single letters makes no sense
    regexList += [' [a-zA-Z][a-zA-Z] ']  # two-letter words makes no sense
    regexList += ["  "]  # double spaces

    return regexList


def clean_text(text):

    for regex in getRegexList():
        text = removeString(text, regex)

    return text


def translate(text):
    if detect(text) == 'en':
        return text

    try :
        return GoogleTranslator(source='auto', target='en').translate(text)
    except :
        return text
```

8. Implications

Present manual ticketing system is not error proof and fast. Guided by powerful AI techniques that can classify incidents to right functional groups can help organizations reduce the issue's resolving time and focus on more productive tasks.

Advanced AI tools can learn, adapt, and determine actions to take, without human input.

Automatic ticket classification with machine learning uses natural language processing (NLP), which helps machines process, understand, and generate human language quickly, consistently, and cost-effectively.

Correct ticket classification helps businesses sort their unstructured data, which provides many more valuable insights than structured data and utilize that for business growth.

As the business expands, customer queries, issues, feedback, requests also increase. Using AI techniques makes it scalable, it's also fast and highly accurate if models have been trained correctly. It can sort millions of pieces of data at a fraction of the cost of manual tagging, save time, and avoid burdening teams with tedious and repetitive tasks.

It can work around the clock, so can respond immediately.

Consistent criteria- Ticket classification with machine learning applies the same criteria to measure each set of data, plus a machine will never be subjective lack alertness, and rush through tickets without understanding them properly.

It can be easily integrated with other AI tools, including chatbots and automatic data collection and reporting thus helping in extracting more insights

9. Limitations

Even though the Test and Train accuracy is high and close, due to the huge imbalance and the need to oversample has resulted in a level of overfitting (not 100% overfit - we evaluated with queries from existing dataset). But when we implement this system in real environments, we need to retrain the model on better and balanced data. It will significantly improve the prediction capability.

The current groups have a lot of overlapping characteristics and we are not provided with relevant information as to what each class represents so it is hard to optimize from a human perspective.

Providing the guidelines used by the people using the manual ticketing system will also help optimize the model further and make it more reliable for IT companies to use in the real world.

10. Closing Reflections

Data is key. A clean confusion free dataset can help to develop a wonderful AI based prediction system.

Balanced data with very minimal misclassification is needed for better performance of Classification models. From our dataset we clearly understood that because of heavily unbalanced data with misclassification leads to un-reliable predictions in almost all the classifiers.

There is a very strong relation between Data pre-processing and performance of the model. In our case different pre-processing especially the unwanted character removal process significantly improved the performance of almost all the models.

Possible Trials for next time:

Improving the character removal algorithm. We noticed a significant jump in performance on changing our method of removal of unwanted characters to one with more rigorous rules.

Try removing the common words between classes to reduce misclassification. There are many words that are commonly used across the groups which can lead to misclassification in learning. We need to find a method to address this issue as we do not know the impact of this clearly as of now. We need to perform more tests.

Furter experimentation of the model itself to maybe attain a better system