

# AIML Online Capstone - AUTOMATIC TICKET ASSIGNMENT - Interim Report

December 5th, 2020

Vineeth Govind, Goutham Jayaraman, Nishad Jadhav, Sridhar  
Krishnaswamy

AIML Online Batch Group 4 - Year 2020

Mentor: Sumit Kumar

## Table of Contents

[Summary of problem statement, data and findings](#)

[Current 'Pain' Points](#)

[Objective of this Project](#)

[Milestone - 1](#)

[Our Approach for Milestone-1](#)

[Our approach to Pre Processing](#)

[Dataset for Deep Learning](#)

[EDA \(cont'd\)](#)

[Document Clustering](#)

[Topic Modeling \(aka Tagging\)](#)

[Pre Processing \(cont'd\)](#)

[Deciding Models and Model Building](#)

[Machine Learning Models:](#)

[Deep Learning Models](#)

[Deep Learning Model Training with GloVe: Global Vectors for Word Representation.](#)

[How to improve your model performance?](#)

***The title page contains the actual words from Tensorflow Embedding Projector <https://projector.tensorflow.org/> created with the provided dataset metadata and vectors. The code for this is in the EDA notebook (last few steps)***

## Summary of problem statement, data and findings

Incident Management and Response is a key component of any IT Service Management Strategy. These are the typical steps involved in the Incident Management Process:

- a. Receipt of the issue
- b. Create a ticket
- c. Review of the ticket by L1/L2 teams
- d. Attempt to resolve the ticket using Standard Operating Procedures by L1/L2
- e. If needed, transfer the ticket to the appropriate L3 team for further review and resolving.

### Current 'Pain' Points

Currently the organization sees these issues in the Incident Ticket Management Process:

- a. The process is largely 'manual'. L1/L2 teams need to spend time to review Standard Operating Procedures (SOPs) before assigning to functional teams. Minimum 25-30% incidents need to be reviewed for SOPs before ticket assignment.
- b. Minimum 1 FTE effort needed only for incident assignment to L3 teams**
- c. Human error - many times the incident gets assigned to the wrong L3 team. So additional effort needed to reassign to the correct team after re-review of the ticket, this not only increases the manual effort needed BUT *also leads to customer dis-satisfaction because the customer who opened the ticket is left frustrated because the ticket is in limbo being tossed between various teams before getting to the actual team who can help resolve the issued.*

### Objective of this Project

The dataset provided has about 8500 records showing Ticket Long & Short Description, the caller, and the Group to which the Ticket has been assigned to.

Create various Machine Learning Models that can help classify incidents and assign them to the right Functional Group. Our objective is to create NLP models that can predict with at least 80% accuracy.

## Milestone - 1

- a. EDA - Explore and understand the dataset provided
  - i. Visualizing different patterns
  - ii. Visualizing different text features
  - iii. Identify data discrepancies
- b. Text preprocessing
  - i. Dealing with missing values
  - ii. Text Translation
- c. Explore ways to augment the dataset provided without losing the relationship to the target label
- d. Creating word vocabulary from the corpus of report text data
- e. Creating tokens as required

### Our Approach for Milestone-1

EDA:

- There are 8500 records in the dataset
- Each Dataset contains 4 columns
- The column 'Caller' seems to contain only junk. So we dropped the column
- We identified junk characters using ftfy library
- There were very few columns with Nulls

An example of data that looked like junk at an initial glance. But on applying the 'fixes text for you - ftfy' library, we discovered that the data is actually Ticket Description in Chinese Simplified

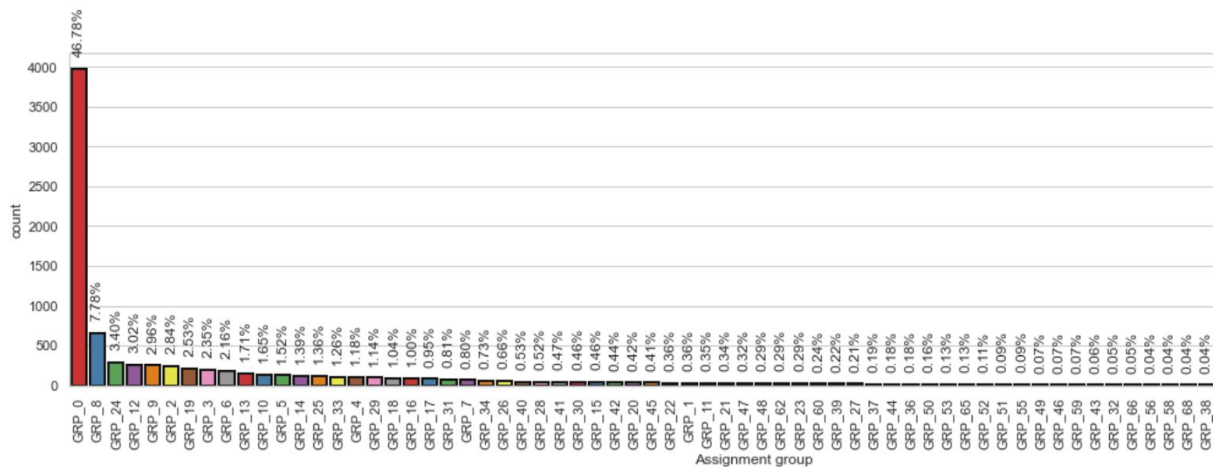
Junk text: ǫ"µè,, 'ăĵă,"èĴžă,ă,Šă†...ǫ½‘

Fixed text: 电脑卡且连不上内网

We discovered this kind of brokenness is called Mojibake

- when someone has encoded Unicode with one standard and decoded it with a different one. This often shows up as characters that turn into nonsense sequences (called “mojibake”)

Fig 1. Distribution of Tickets amongst various groups:



As the figure 1 shows, Group 0 has the most number of tickets, followed by Group 8. The brief (problem statement) provided indicated that about 54% are resolved by L1/L2 groups. Since we also find from the above graph GRP\_0 + GRP\_8 also equals 54% roughly, we guess GRP\_0 is L1 and GRP\_8 is probably L2. The remaining groups are probably Functional/L3 teams. (The word cloud for GRP\_8 also indicate this group could be related to monitoring tool OR job scheduler)

Some other interesting Figures are below. Fig2 shows the Assignment groups with the highest number of tickets. Figure 3 shows the groups with the lowest number of tickets. Some groups have only 1 ticket. We can choose to discard these groups. BUT, we have chosen a different approach - which we will describe more in the Data Augmentation section. We have done our best NOT to delete any data in the dataset provided. Figure 4 shows a pie chart % distribution by assignment group. Figures 6a, 6b, 6b show the Word Cloud.

Fig 2.

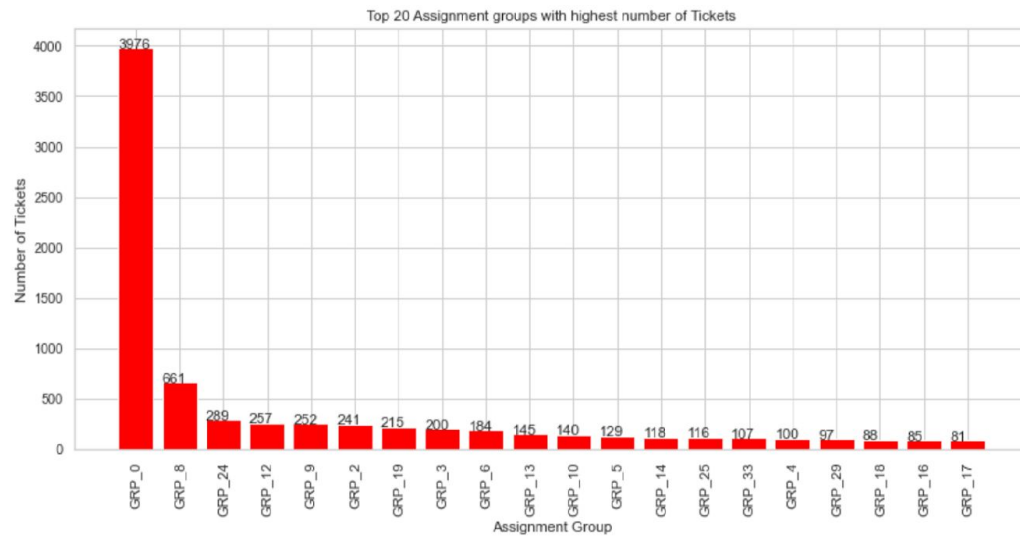


Fig 3.

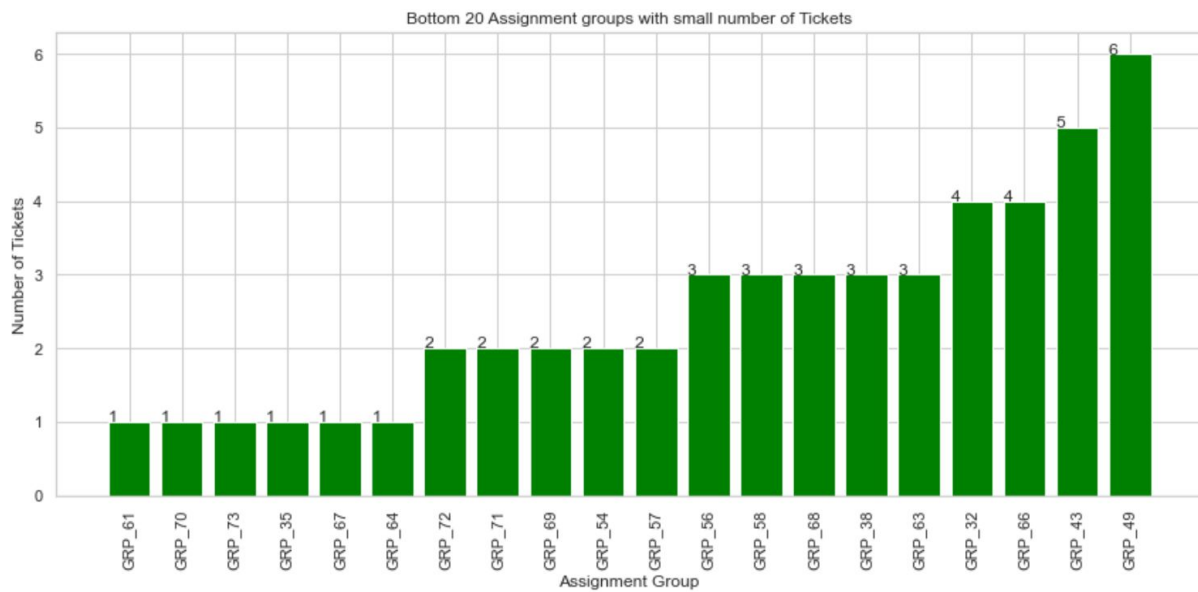


Fig 4.

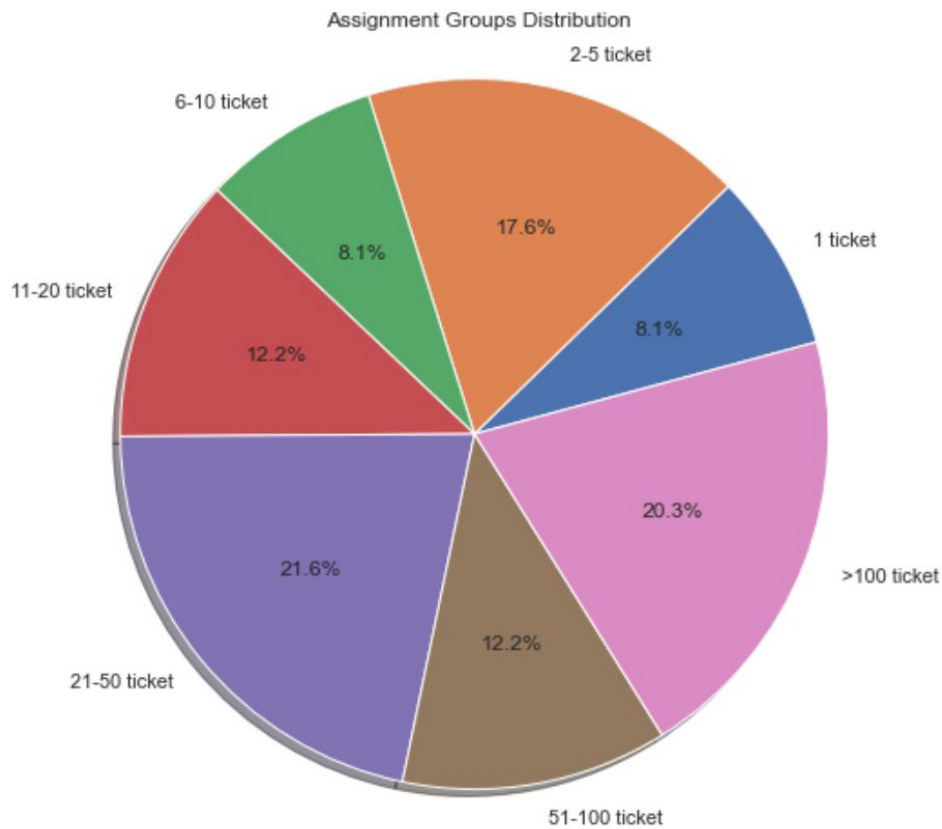


Fig 6a - Word Clouds for GRP\_0

- These ticket descriptions indicate issues typically resolved by Help Desk using Standard Operating Procedures

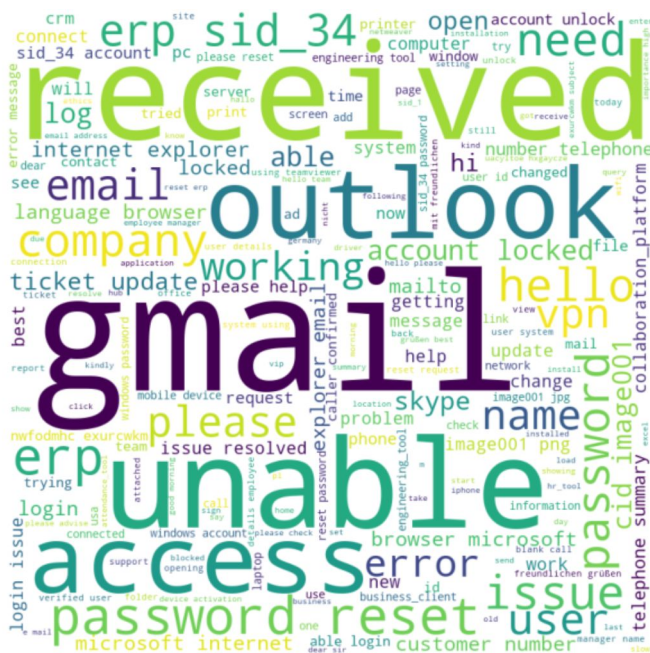




Fig 6b - Word Clouds for GRP\_8

The words indicate issues reported by Monitoring tool OR tools like Job Scheduler



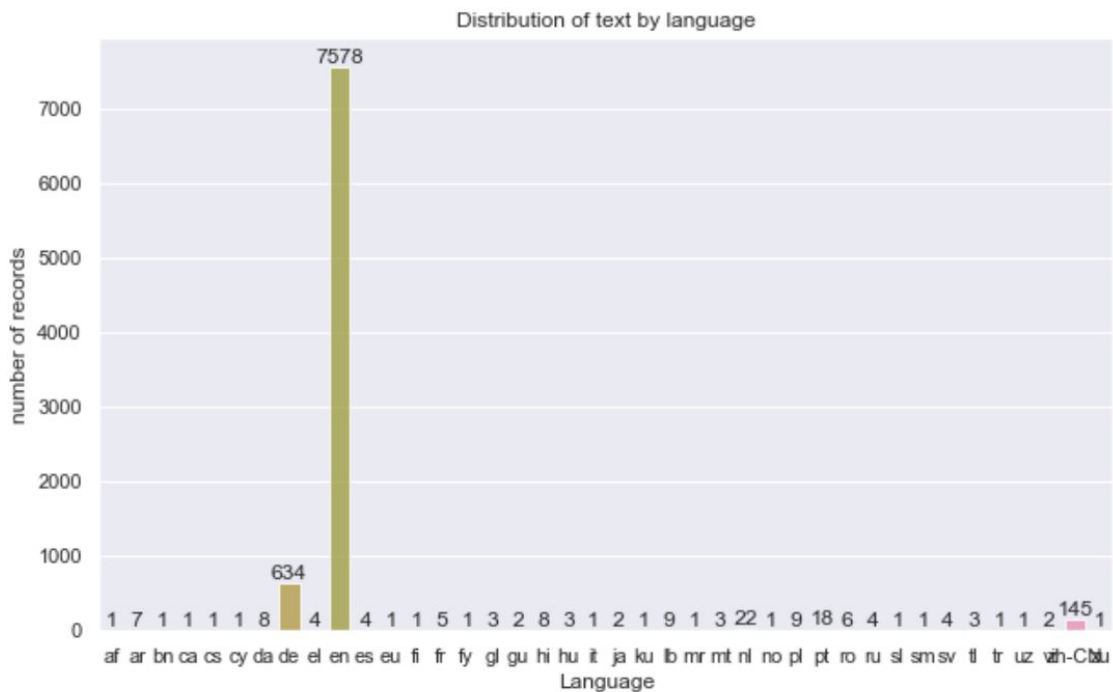
Fig 6c Word Cloud for GRP\_24



GRP\_24 Word Cloud led us into looking into Tickets with languages other than English. We used the Google Translation Engine to look into this further



As Fig 7 shows, most of the Tickets are in English followed by German and Simplified Chinese.



We also found in some tickets, the Short Description and Description were in different languages. An example below where the Description is in english and Short Description is in Portuguese

Fig 8.

	Short description	Description	Assignment group	Raw Combined description	raw_word_count	Combined description	language	language name	Translated Short description
8498	machine não está funcionando	i am unable to access the machine utilities to...	GRP_62	machine não está funcionando i am unable to ac...	18	machine is not working i am unable to access t...	pt	portuguese	machine is not working i am ur

For more details on the EDA, please review this notebook:  
***capstone\_nlp\_merged\_EDA\_Preprocessing\_Translation\_v13.ipynb***

## Our approach to Pre Processing

So we decided to translate the Short Description and Description fields separately first and then combine them for further preprocessing.

Fig 8 shows the Translated Short Description and Description field of a sample.

Short description	电话机没有声音
Description	电话机没有声音
Assignment group	GRP_30
Raw Combined description	电话机没有声音
raw_word_count	1
Combined description	No sound from the phone
language	zh-CN
language name	chinese (simplified)
Translated Short description	No sound from the phone
Translated Description	No sound from the phone

For translation we used the Google Translator API in multithreading mode which performed the translation in a few seconds (all 8500 records)

We also found a couple of records where the Translator got confused so we fixed the data manually.

Fig 9 - In the sample below, the Translator mistook the sentences as Greek. We had to manually fix the data.

	Short description	Description	Assignment group	Raw Combined description	raw_word_count	Combined description	language	language name	Translated Short description	Translated Description
8043	setup new ws \xaqzisk ahbgjqz	setup new ws \xaqzisk ahbgjqz	GRP_24	setup new ws \xaqzisk ahbgjqz	5		el	greek	σετύπ νέω ως \χακζηορκ αχβγερκζ	σετύπ νέω ως \χακζηορκ αχβγερκζ
8072	setup new ws \pnwbkitv phbnwmkl	setup new ws \pnwbkitv phbnwmkl	GRP_24	setup new ws \pnwbkitv phbnwmkl	5		el	greek	σετύπ νέω ως \πνωβκίτυ φβνωμκλ	σετύπ νέω ως \πνωβκίτυ φβνωμκλ

We also fixed about 17 other records manually .

Using the ftfy library, we fixed the junk characters as noted above.

There were a couple of null records with null values for Short Description and Description. We fixed those to empty spaces.

After the translation, we combined the Short Description and Description fields and then applied the logic to remove special characters, numbers and things like email addresses .

*For more details on the Preprocessing, please review this notebook:*  
**capstone\_nlp\_merged\_EDA\_Preprocessing\_Translation\_v13.ipynb**

## Dataset for Deep Learning

The preprocess is still NOT complete, but, at this point, we created the dataset for Deep Learning. The reason we did this is because we wanted to retain the stop words and 'un lemmatized' words so the Deep Learning Algorithms will be able to process them in a context sensitive manner.

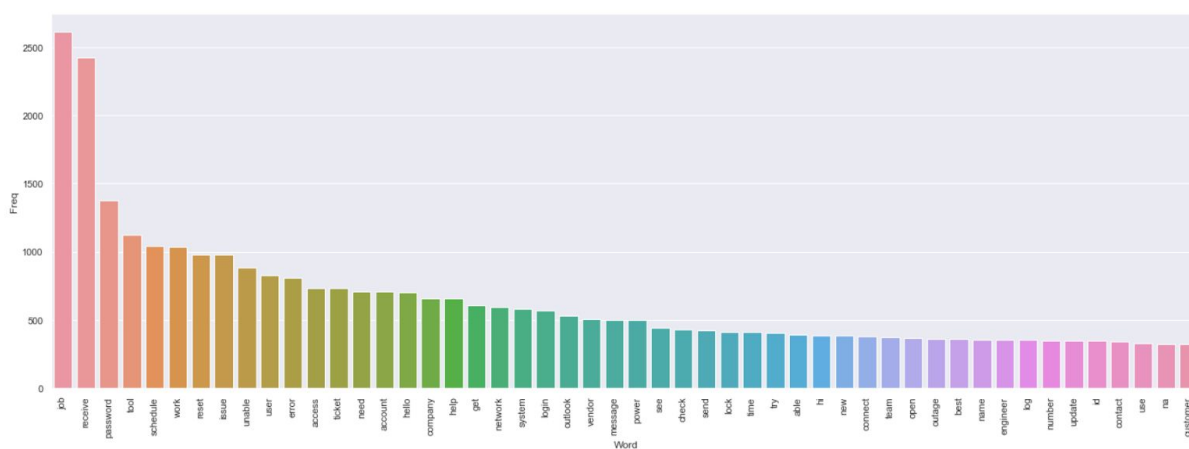
## EDA (cont'd)

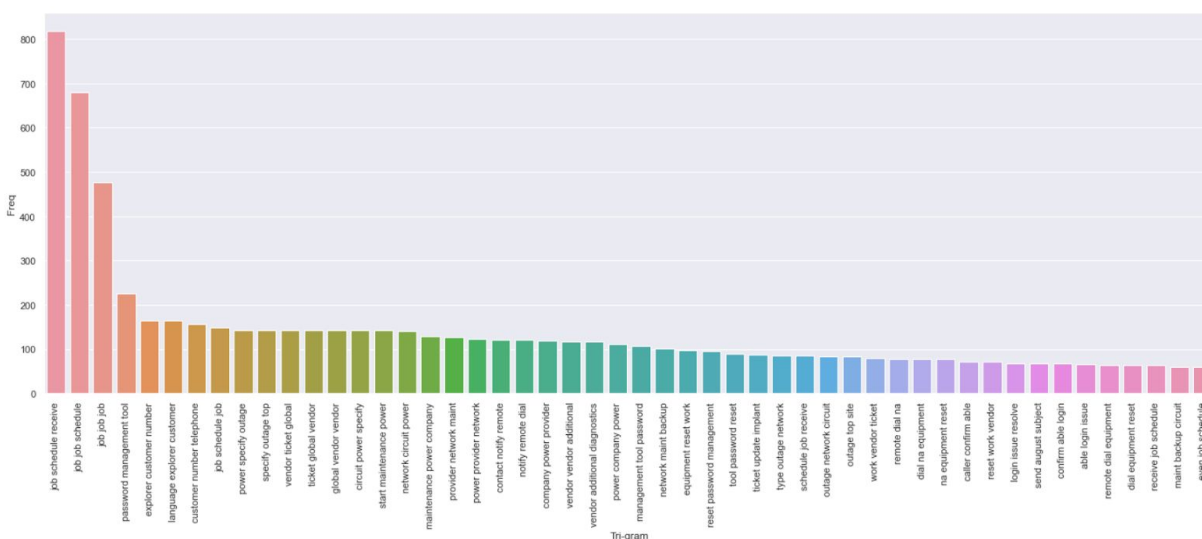
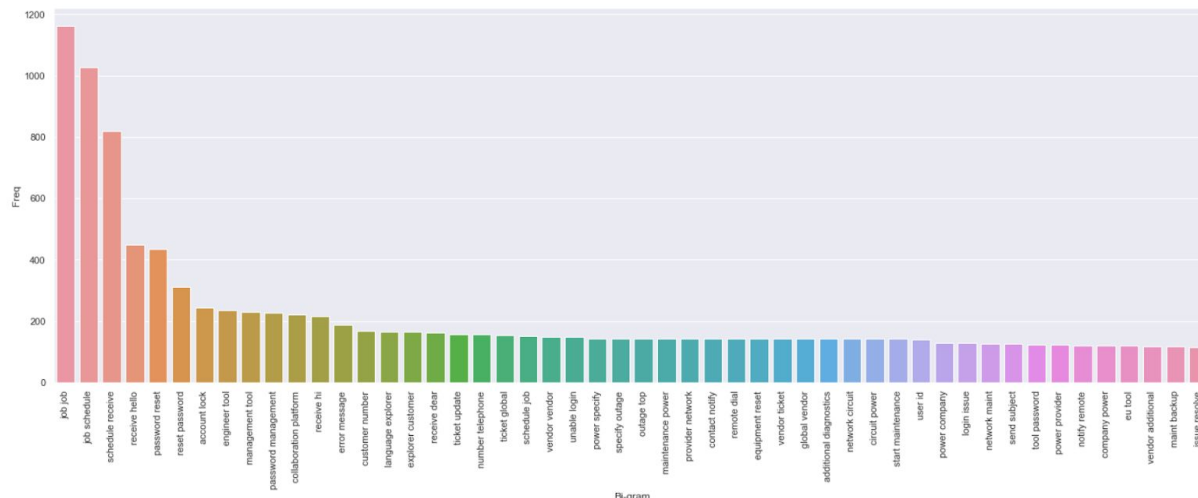
We then used the NLTK library for stopwords in English. We also used punkt and wordnet for lemmatization .

Total Corpus Word Count before lemmatization: 137508

Total Corpus Word Count after lemmatization: 82245

Fig 10. **Top 20 uni-grams, bi-grams & tri-grams**





## Document Clustering

We also attempted to use the K-Means algorithm to do Document clustering.

Why is document clustering an important tool in NLP?

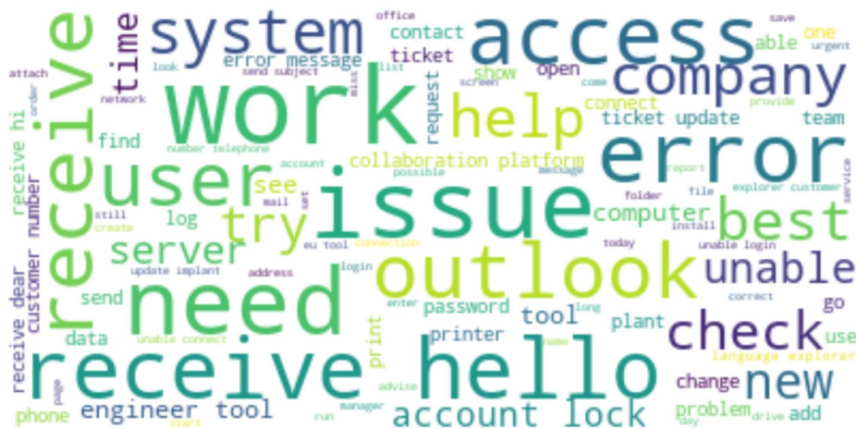
Document Clustering is a process of grouping similar items together. Each group, also called as a cluster, contains items that are similar to each other. Clustering algorithms are unsupervised learning algorithms i.e. we do not need to have labelled datasets. There are many clustering algorithms for clustering including KMeans, DBSCAN, Spectral clustering, hierarchical clustering etc and they have their own advantages and disadvantages. The choice of the algorithm mainly depends on whether or not you already know how many clusters to create. Some algorithms such as KMeans need you to specify the number of clusters to create whereas DBSCAN does

First we decided on the optimal k - we found the optimal k to be 4 as shown by the 'elbow method'

Cluster:0 contains words like outlook, unable, etc

Cluster: 2 contains words related password management and reset

Cluster: 0



Topic modeling is an interesting problem in NLP applications where we want to get an idea of what topics we have in our dataset. A topic is nothing more than a collection of words that describe the overall theme. For example, in case of news articles, we might think of topics as politics, sports etc. but topic modeling won't directly give you names of the topics but rather a set of most probable words that might describe a topic. It is up to us to determine what topic the set of words might refer to



In our dataset, we identified these words highlighted that seem to suggest the most important topics are job schedule, password resets, account locks, login issues, vendor issues.

0 **job schedule** receive hot cold payroll na notch prod tool eu fail arc hand way

1 **password reset** management tool request user manager unlock production change receive es need login id

2 **account lock** unlock ad user lockout id check login issue hello frequent need team receive

3 unable **login** outlook issue connect tool access user work error open engineer receive help log

4 ticket update implant **vendor** power network outage circuit na yes work site global company active

At a later point, we may use these techniques to generate more training data if needed.

## Pre Processing (cont'd)

We used the WordNetLemmatizer to lemmatize the corpus

After lemmatization and word - gram analysis we created the dataset required for the Machine Learning Models.

We also applied spell check and Contraction logic (for example: change "there'd've" to "there would have")

Finally we build the processed dataset required for Machine Learning Models.

*For more details on the Preprocessing, please review this notebook:*

***capstone\_nlp\_merged\_EDA\_Preprocessing\_Translation\_v13.ipynb***



### Our final numbers after preprocessing

In [133]:

1 mydata.isna().apply(pd.value\_counts)

Out[133]:

	Short description	Description	Assignment group	Raw Combined description	raw_word_count	Combined description	language	language name	Translated Short description	Translated Description	CombinedWordCount	D
False	8500	8500	8500	8500	8500	8500	8500	8500	8500	8500	8500	

After preprocessing, we ran an initial Model with Logistic Regression. It scored about 62% in accuracy and an f1 score of 57%

Fig 11.

	Algorithm	Accuracy	f1	Precision Score	Recall Score
1	LR 30% Raw Test Data after Cleansing	62.12%	56.91%	61.34%	67.72%

### Deciding Models and Model Building

For more details on the Machine Learning Models, please review this notebook: [capstone-nlp-ML-Model-initial-run.ipynb](#)

For more details on the Machine Learning Models, please review this notebook: [capstone-nlp-DL-Model-initial-run.ipynb](#)

The dataset indicates a Multi-Classification problem with the 74 for labels. Our initial thinking is these models will be applicable in this class of problem.

We used the dataset after Augmentation for running the Machine Learning models. The dataset name is  
input\_data\_after\_data\_for\_ml\_\_with\_augmentation\_knn\_round2.csv

We did these steps to prepare the data for feeding into the ML algorithms:

- We applied the Label Encoder on the labels.
- For the Features, we used the Count Vectorizer and did a fit transform

- c. We applied the Tfidf Transformer to create a matrix of Tf-idf features.
- d. We used the sklearn train test split to do a 80:20 split of the Training and Test data.

#### Machine Learning Models:

1. **Logistic Regression:** Logistic regression is one of the most simple Machine Learning models. They are easy to understand, interpretable, and can give pretty good results. Every practitioner using Logistic Regression out there must know about the Log-Odds which is the main concept behind this learning algorithm. The Logistic regression is very much interpretable considering the business needs and explanation regarding how the model works concerning different independent variables used in the model.

#### **Parameters we used for Logistic Regression:**

`solver='lbfgs', max_iter=1000, multi_class='multinomial'`

For multiclass problems, 'lbfgs' handles multinomial loss

For 'multinomial' the loss minimised is the multinomial loss fit across the entire probability distribution, *even when the data is binary*.

#### **Results from Logistic Regression:**

	Algorithm	Accuracy	f1	Precision Score	Recall Score
1	LR 30% Test Data after Data Augmentation Word Embedding Round 2	76.48%	76.10%	78.98%	76.77%

2. **Support Vector Machines - SVM is a sparse technique.** Like nonparametric methods, SVM requires that all the training data be available, that is, stored in memory during the training phase, when the parameters of the SVM model are learned. However, once the model parameters are identified, SVM depends only on a subset of these training instances, called *support vectors*, for future prediction. Support vectors define the margins of the hyperplanes.

## Parameters we used for SVC:

We first used the GridSearchCV to find the best parameters, Some Parameters we are trying the find the ideal values for:

- C is used during the training phase and says how much outliers are taken into account in calculating Support Vectors
- gamma determines the radius of the area of influence of the support vectors if it is too large, the radius of the area of influence of the support vectors only includes the support vector itself and no amount of regularization with C will be able to prevent overfitting. When gamma is very small, the model is too constrained and cannot capture the complexity or “shape” of the data.
- kernel parameter selects the type of hyperplane used to separate the data.
- We will let the Hypersearch determine the best kernel to use among
  - rbf : The RBF kernel is a stationary kernel. It is also known as the “squared exponential” kernel.
  - poly : A polynomial kernel is a more generalized form of the linear kernel. The polynomial kernel can distinguish curved or nonlinear input space. The polynomial and RBF are especially useful when the data-points are not linearly separable.
  - sigmoid: The Sigmoid Kernel comes from the Neural Networks field, where the bipolar sigmoid function is often used as an activation function for artificial neurons
- probability: We will choose between True & False

## Results from SVC:

	Algorithm	Accuracy	f1	Precision Score	Recall Score
1	SVC 30% Test Data after Data Augmentation Word Embedding Round 2	91.43%	91.44%	94.37%	89.24%
2	LR 30% Test Data after Data Augmentation Word Embedding Round 2	76.48%	76.10%	78.98%	76.77%

3. K Nearest Neighbors (KNN) - KNN is another useful algorithm for Multi classification problems. The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each

other . As we note later, we used this algorithm also to efficiently find synonyms of the key words in our corpus.

### Parameters we used for KNN:

We first ran the KNN to find the optimal k between 1 and 20. We then applied the KNN Classifier with the optimal k value found .

### Results from SVC:

	Algorithm	Accuracy	f1	Precision Score	Recall Score
1	SVC 30% Test Data after Data Augmentation Word Embedding Round 2	91.43%	91.44%	94.37%	89.24%
2	KNN 30% Test Data after Data Augmentation Word Embedding Round 2	88.81%	88.70%	89.36%	90.07%
3	LR 30% Test Data after Data Augmentation Word Embedding Round 2	76.48%	76.10%	78.98%	76.77%

## 4. Ensembles

- a. Bagging Classifier: Bagging uses a simple approach that shows up in statistical analyses again and again — improve the estimate of one by combining the estimates of many. Bagging constructs n classification trees using bootstrap sampling of the training data and then combines their predictions to produce a final meta-prediction.

### Parameters used for Bagging Classifier:

n\_estimators=10, max\_samples= .7, bootstrap=True

### Results from Bagging Classifier:

	Algorithm	Accuracy	f1	Precision Score	Recall Score
1	SVC 30% Test Data after Data Augmentation Word Embedding Round 2	91.43%	91.44%	94.37%	89.24%
2	KNN 30% Test Data after Data Augmentation Word Embedding Round 2	88.81%	88.70%	89.36%	90.07%
3	Bagging Classifier 30% Test Data after Data Augmentation Word Embedding Round 2	83.94%	83.94%	92.17%	84.95%
4	DT 30% Test Data after Data Augmentation Word Embedding Round 2	78.37%	78.35%	82.03%	79.56%
5	LR 30% Test Data after Data Augmentation Word Embedding Round 2	76.48%	76.10%	78.98%	76.77%

- b. Decision Tree: Decision tree builds classification or regression models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with **decision nodes** and **leaf nodes**. A decision node (e.g., Outlook) has two or more branches (e.g., Sunny, Overcast and

Rainy). Leaf node (e.g., Play) represents a classification or decision. The topmost decision node in a tree which corresponds to the best predictor called **root node**. Decision trees can handle both categorical and numerical data.

### Parameters used for Decision Tree:

We built our model using the DecisionTreeClassifier function. Using default 'gini' criteria to split. Another option is 'entropy'.

Gini: measures how often a randomly chosen element from the set would be incorrectly labeled.

Entropy: It is used to measure the impurity or randomness of a dataset

### Results from Decision Tree:

	Algorithm	Accuracy	f1	Precision Score	Recall Score
1	SVC 30% Test Data after Data Augmentation Word Embedding Round 2	91.43%	91.44%	94.37%	89.24%
2	KNN 30% Test Data after Data Augmentation Word Embedding Round 2	88.81%	88.70%	89.36%	90.07%
3	DT 30% Test Data after Data Augmentation Word Embedding Round 2	78.37%	78.35%	82.03%	79.56%
4	LR 30% Test Data after Data Augmentation Word Embedding Round 2	76.48%	76.10%	78.98%	76.77%

- c. XGBoost: The XGBoost algorithm is effective for a wide range of regression and classification predictive modeling problems. It is an efficient implementation of the stochastic gradient boosting algorithm and offers a range of hyperparameters that give fine-grained control over the model training procedure. Although the algorithm performs well in general, even on imbalanced classification datasets, it offers a way to tune the training algorithm to pay more attention to misclassification of the minority class for datasets with a skewed class distribution

### Parameters used for XGB:

We used GridSearchCV to look for best parameters for XGB. These are some of the parameters fed into the GridSearchCV algorithm:

- `min_child_weight` : Minimum sum of instance weight (hessian) needed in a child.

- **gamma**: Minimum loss reduction required to make a further partition on a leaf node of the tree. The larger gamma is, the more conservative the algorithm will be.
- **subsample**: Subsample ratio of the training instances. Setting it to 0.5 means that XGBoost would randomly sample half of the training data prior to growing trees. and this will prevent overfitting.
- **colsample\_bytree**: is the subsample ratio of columns when constructing each tree. Subsampling occurs once for every tree constructed.

### Results from XGB:

	Algorithm	Accuracy	f1	Precision Score	Recall Score
1	SVC 30% Test Data after Data Augmentation Word Embedding Round 2	91.43%	91.44%	94.37%	89.24%
2	KNN 30% Test Data after Data Augmentation Word Embedding Round 2	88.81%	88.70%	89.36%	90.07%
3	XGBoost Classifier 30% Test Data after Data Augmentation Word Embedding Round 2	84.06%	84.16%	90.65%	83.31%
4	Bagging Classifier 30% Test Data after Data Augmentation Word Embedding Round 2	83.94%	83.94%	92.17%	84.95%
5	DT 30% Test Data after Data Augmentation Word Embedding Round 2	78.37%	78.35%	82.03%	79.56%
6	LR 30% Test Data after Data Augmentation Word Embedding Round 2	76.48%	76.10%	78.98%	76.77%

### 5. Stacker:

We also tried with Stacker Classifier with layer 1 and layer 2 estimators. In Layer 1 we used these estimators: SVC, Logistic Regression, Decision Tree. Final Estimator used was XGB.

Layer 2 - we used these estimators: Bagging Classifier, Logistic Regression, XG Boost. Layer 2 served as the final estimator for Layer 1.

	Algorithm	Accuracy	f1	Precision Score	Recall Score
1	SVC 30% Test Data after Data Augmentation Word Embedding Round 2	91.43%	91.44%	94.37%	89.24%
2	Stacker 30% Test Data after Data Augmentation Word Embedding Round 2	90.50%	90.46%	93.74%	88.29%
3	KNN 30% Test Data after Data Augmentation Word Embedding Round 2	88.81%	88.70%	89.36%	90.07%
4	XGBoost Classifier 30% Test Data after Data Augmentation Word Embedding Round 2	84.06%	84.16%	90.65%	83.31%
5	Bagging Classifier 30% Test Data after Data Augmentation Word Embedding Round 2	83.94%	83.94%	92.17%	84.95%
6	DT 30% Test Data after Data Augmentation Word Embedding Round 2	78.37%	78.35%	82.03%	79.56%
7	LR 30% Test Data after Data Augmentation Word Embedding Round 2	76.48%	76.10%	78.98%	76.77%



## Deep Learning Models

So far we have implemented the following models for unaugmented data, level1 augmented data and level2 augmented data.

1. LSTM
2. GRU
3. Bidirectional LSTM
4. Bidirectional GRU

There are 2 variations of each model one without hyperparameter tuning and one with hyperparameter tuning. The hyperparameter tuning was done for batch size, number of epochs, optimizer(Adam, SGD, RMSprop) and learning rate. Thus we have a total of 24 models. The accuracy scores are shown in the table below

**Table1:Accuracy Scores**

\*hp-hyperparameter tuned

Model	Unaugmented data	Unaugmented data with hp*	Level1 Augmented data	Level1 Augmented data with hp*	Level2 Augmented data	Level2 Augmented data with hp*
LSTM	56.9	58.9	77.2	83.4	85.5	89.4
GRU	58.3	58.3	81.5	84.7	89	91
BILSTM	57.6	58.5	76.4	83.7	86.7	91.6
BIGRU	58	55.4	81.4	86.5	89.5	92

## Conclusions

It can be seen that augmentation of the data gives a substantial improvement in the accuracy score and hyperparameter tuning gives a marginal improvement in most cases. The Bidirectional GRU using level 2 augmented data gives the highest accuracy of 92%.The level 2 augmented data with hyperparameter tuned models has an accuracy improvement of at least over 32% in all models.



Deep Learning Model Training with GloVe: Global Vectors for Word Representation.

For details please refer to notebook capstone-nlp-DL-GIOve.ipynb

GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.

We tried these Deep Learning Models after processing the dataset to use the GloVe Vectors . The glove dataset used was glove.840B.300d.txt.

We first used the sklearn train test split to split the dataset. We converted the dataset to GloVe embedded matrix.

Parameters used:

- EMBEDDING\_FILE =  
'embeddings/glove.840B.300d/glove.840B.300d.txt'
- MAX\_SEQUENCE\_LENGTH = 500
- EMBEDDING\_DIM=300
- MAX\_NB\_WORDS=75000
- BATCH\_SIZE = 32

The models trained, parameters used and results are below:

### CNN:

**Optimizer used: tf.keras.optimizers.SGD(lr=0.001, momentum=0.9)**

Model: "model"

Layer (type)	Output Shape	Param #	Connected to
=====			
input_1 (InputLayer)	[(None, 500)]	0	
=====			
embedding (Embedding)	(None, 500, 300)	1353000	input_1[0][0]

conv1d (Conv1D)	(None, 499, 128)	76928	embedding[0][0]
conv1d_1 (Conv1D)	(None, 498, 128)	115328	embedding[0][0]
conv1d_2 (Conv1D)	(None, 497, 128)	153728	embedding[0][0]
conv1d_3 (Conv1D)	(None, 496, 128)	192128	embedding[0][0]
conv1d_4 (Conv1D)	(None, 495, 128)	230528	embedding[0][0]
max_pooling1d (MaxPooling1D)	(None, 99, 128)	0	conv1d[0][0]
max_pooling1d_1 (MaxPooling1D)	(None, 99, 128)	0	conv1d_1[0][0]
max_pooling1d_2 (MaxPooling1D)	(None, 99, 128)	0	conv1d_2[0][0]
max_pooling1d_3 (MaxPooling1D)	(None, 99, 128)	0	conv1d_3[0][0]
max_pooling1d_4 (MaxPooling1D)	(None, 99, 128)	0	conv1d_4[0][0]
concatenate (Concatenate)	(None, 495, 128)	0	max_pooling1d[0][0] max_pooling1d_1[0][0] max_pooling1d_2[0][0] max_pooling1d_3[0][0] max_pooling1d_4[0][0]
conv1d_5 (Conv1D)	(None, 491, 128)	82048	concatenate[0][0]

dropout (Dropout)	(None, 491, 128)	0	conv1d_5[0][0]
batch_normalization (BatchNorma	(None, 491, 128)	512	dropout[0][0]
max_pooling1d_5 (MaxPooling1D)	(None, 98, 128)	0	batch_normalization[0][0]
conv1d_6 (Conv1D)	(None, 94, 128)	82048	max_pooling1d_5[0][0]
dropout_1 (Dropout)	(None, 94, 128)	0	conv1d_6[0][0]
batch_normalization_1 (BatchNor	(None, 94, 128)	512	dropout_1[0][0]
max_pooling1d_6 (MaxPooling1D)	(None, 3, 128)	0	batch_normalization_1[0][0]
flatten (Flatten)	(None, 384)	0	max_pooling1d_6[0][0]
dense (Dense)	(None, 1024)	394240	flatten[0][0]
dropout_2 (Dropout)	(None, 1024)	0	dense[0][0]
dense_1 (Dense)	(None, 512)	524800	dropout_2[0][0]
dropout_3 (Dropout)	(None, 512)	0	dense_1[0][0]

dense\_2 (Dense) (None, 74) 37962 dropout\_3[0][0]

Total params: 3,243,762

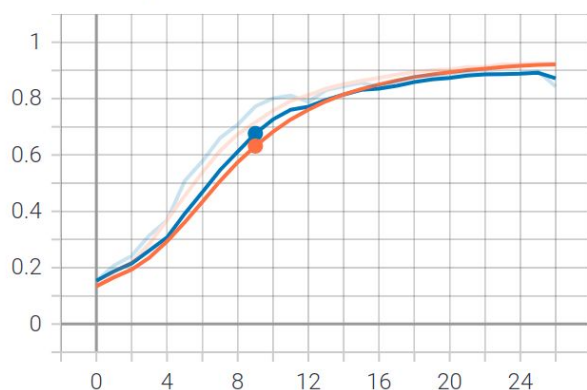
Trainable params: 3,243,250


Non-trainable params: 512

## Results for CNN:

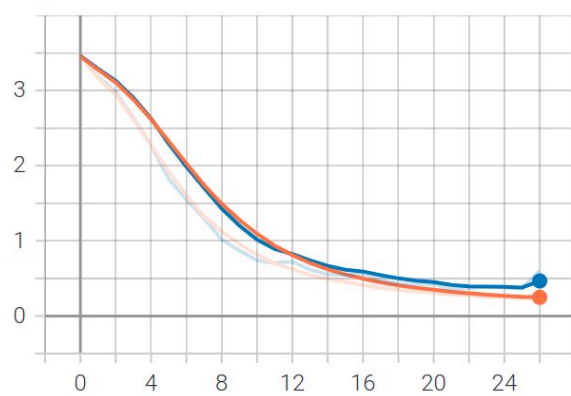
Orange - training, blue validation

epoch\_accuracy

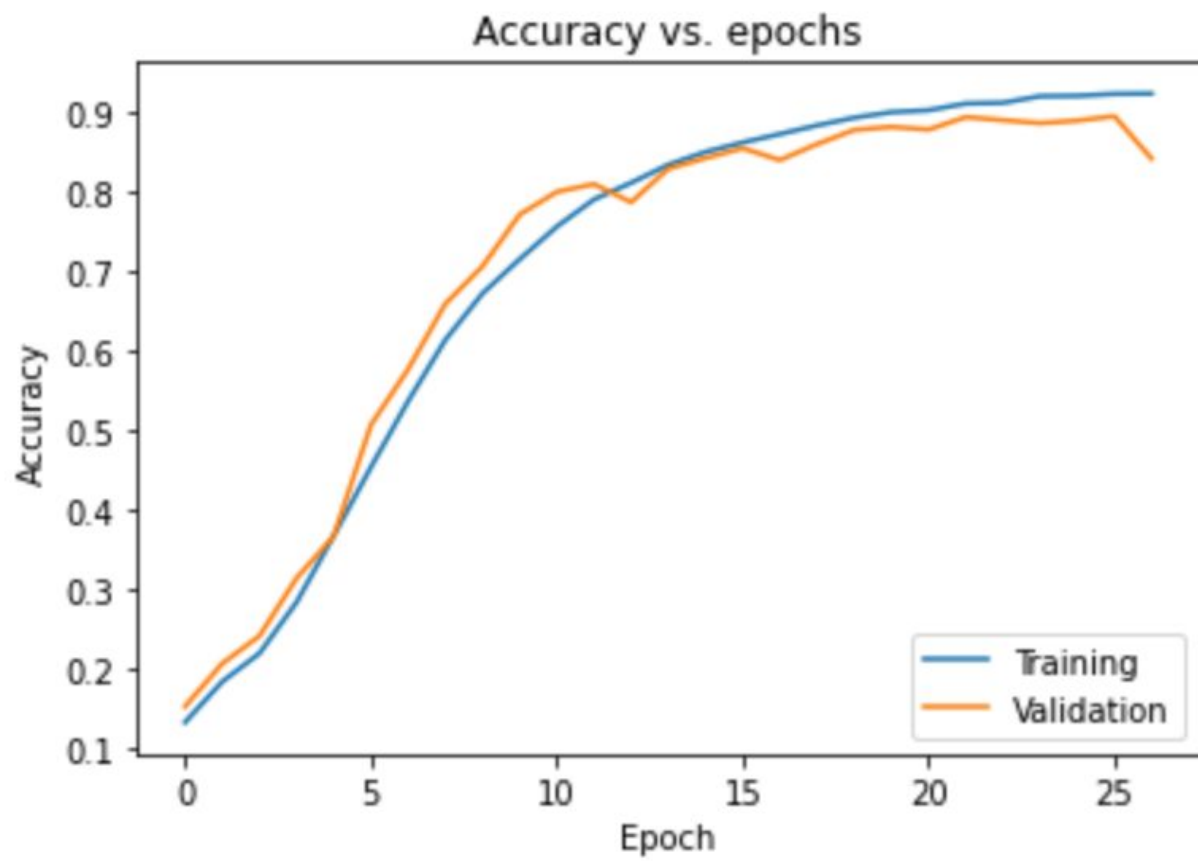


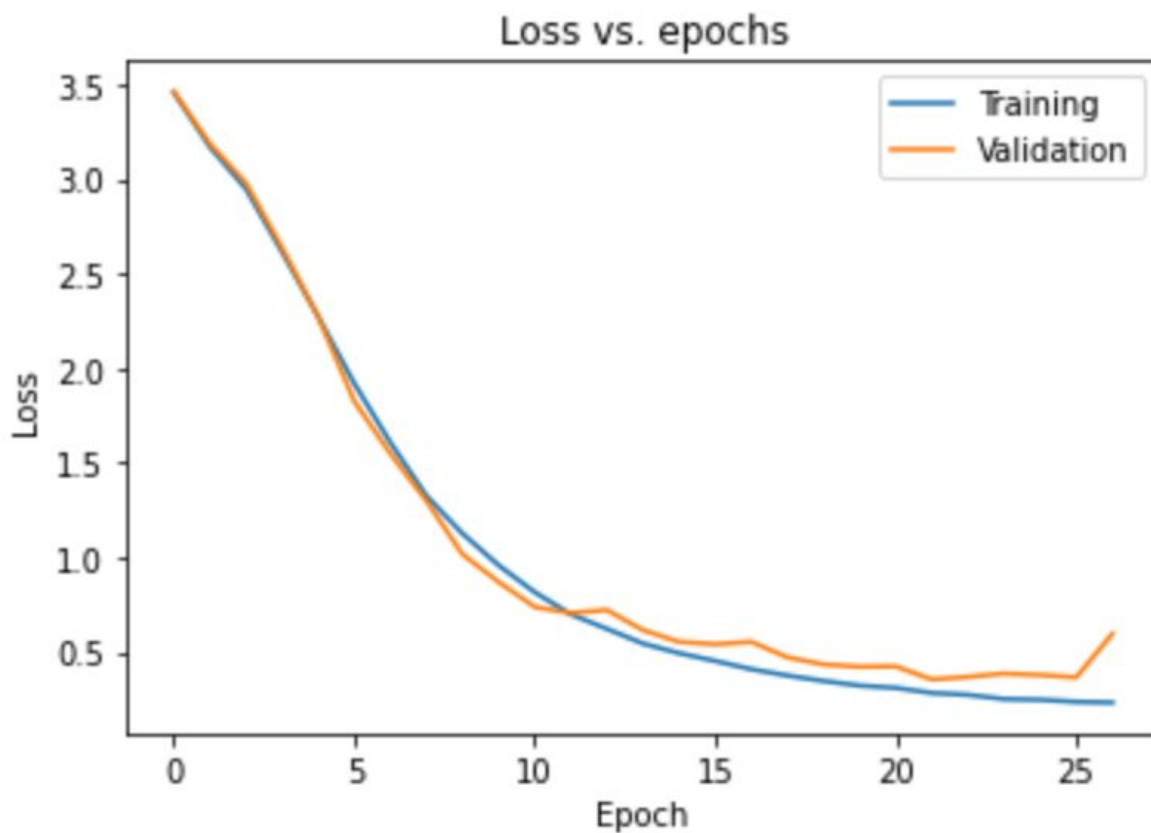
Name	Smoothed	Value	Step	Time	Relative
 run_2020_12_15-10_15_48/train	0.6316	0.7164	9	Tue Dec 15, 10:18:17	2m 8s
 run_2020_12_15-10_15_48/validation	0.6763	0.7726	9	Tue Dec 15, 10:18:17	2m 8s

epoch\_loss



Unsmoothed plots of Accuracy & Loss for CNN:





Epoch 00025: saving model to checkpoints\_best\_only/checkpoint

Epoch 26/100

779/779 [=====] - 15s 20ms/step - loss: 0.2281  
 - accuracy: 0.9276 - val\_loss: 0.3667 - val\_accuracy: 0.8964

Results at the end of the 27th epoch after which early stopping kicked in

Epoch 00026: saving model to checkpoints\_best\_only/checkpoint

Epoch 27/100

779/779 [=====] - 15s 19ms/step - loss: 0.2239  
 - **accuracy: 0.9291 - val\_loss: 0.5970 - val\_accuracy: 0.8429**

val/train: 2.56

Epoch 00027: saving model to checkpoints\_best\_only/checkpoint

Epoch 00027: early stopping

Precision: 0.857542

Recall: 0.842942

F1 score: 0.838929

Evaluation Accuracy: 0.843

Evaluation Loss: 0.597

## GRU:

**Optimizer used: tf.keras.optimizers.SGD(lr=0.001, momentum=0.9)**

### GRU Model Summary:

Model: "sequential"

Layer (type)	Output Shape	Param #
=====		
embedding (Embedding)	(None, 500, 300)	1353000
-----		
cu_dnngru (CuDNNGRU)	(None, 500, 32)	32064
-----		
dropout (Dropout)	(None, 500, 32)	0
-----		
batch_normalization (Batch Normalization)	(None, 500, 32)	128
-----		
cu_dnngru_1 (CuDNNGRU)	(None, 500, 32)	6336
-----		
dropout_1 (Dropout)	(None, 500, 32)	0
-----		



batch_normalization_1	(Batch (None, 500, 32)	128
-----------------------	------------------------	-----

---

cu_dnngru_2	(CuDNNGRU (None, 500, 32)	6336
-------------	---------------------------	------

---

dropout_2	(Dropout (None, 500, 32)	0
-----------	--------------------------	---

---

batch_normalization_2	(Batch (None, 500, 32)	128
-----------------------	------------------------	-----

---

cu_dnngru_3	(CuDNNGRU (None, 500, 32)	6336
-------------	---------------------------	------

---

dropout_3	(Dropout (None, 500, 32)	0
-----------	--------------------------	---

---

batch_normalization_3	(Batch (None, 500, 32)	128
-----------------------	------------------------	-----

---

cu_dnngru_4	(CuDNNGRU (None, 500, 32)	6336
-------------	---------------------------	------

---

dropout_4	(Dropout (None, 500, 32)	0
-----------	--------------------------	---

---

batch_normalization_4	(Batch (None, 500, 32)	128
-----------------------	------------------------	-----

---

cu_dnngru_5	(CuDNNGRU (None, 500, 32)	6336
-------------	---------------------------	------

---

dropout_5	(Dropout (None, 500, 32)	0
-----------	--------------------------	---

---

batch_normalization_5	(Batch (None, 500, 32)	128
-----------------------	------------------------	-----

---

cu_dnngru_6	(CuDNNGRU (None, 32)	6336
-------------	----------------------	------

---

dropout_6	(Dropout (None, 32)	0
-----------	---------------------	---

---

batch_normalization_6	(Batch (None, 32)	128
-----------------------	-------------------	-----

---

dense	(Dense (None, 256)	8448
-------	--------------------	------

---

batch\_normalization\_7 (Batch Normalization) 1024

dense\_1 (Dense) (None, 74) 19018

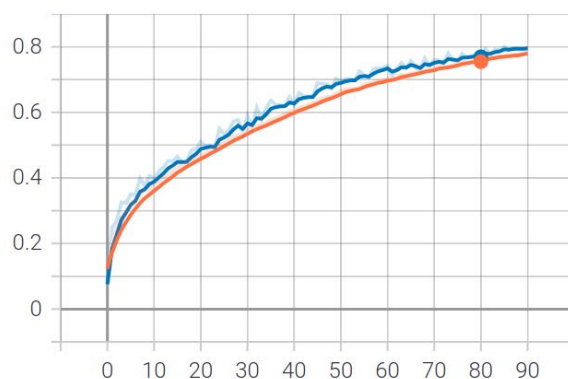
=====  
Total params: 1,452,466

Trainable params: 1,451,506

Non-trainable params: 960

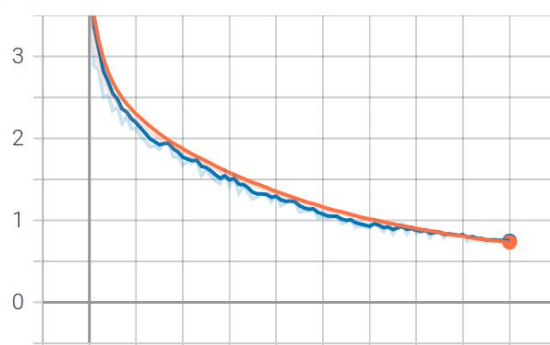
## Results for GRU:

epoch\_accuracy



Name	Smoothed	Value	Step	Time	Relative
 run_2020_12_15-10_22_43/train	0.7546	0.7552	80	Tue Dec 15, 11:57:46	1h 33m 51s
 run_2020_12_15-10_22_43/validation	0.7695	0.7665	80	Tue Dec 15, 11:57:46	1h 33m 51s

epoch\_loss



Epoch 00090: saving model to checkpoints\_best\_only/checkpoint

Epoch 91/100



779/779 [=====] - 76s 97ms/step - loss: 0.6893  
- accuracy: 0.7912 - val\_loss: 0.7361 - val\_accuracy: 0.7978

val/train: 1.03

Epoch 00091: saving model to checkpoints\_best\_only/checkpoint

Epoch 00091: early stopping

Evaluation Accuracy: 0.798

Evaluation Loss: 0.736

Mean Accuracy for the validation dataset:

0.6217235459403677

Mean Loss for the validation dataset:

1.3452635348498165

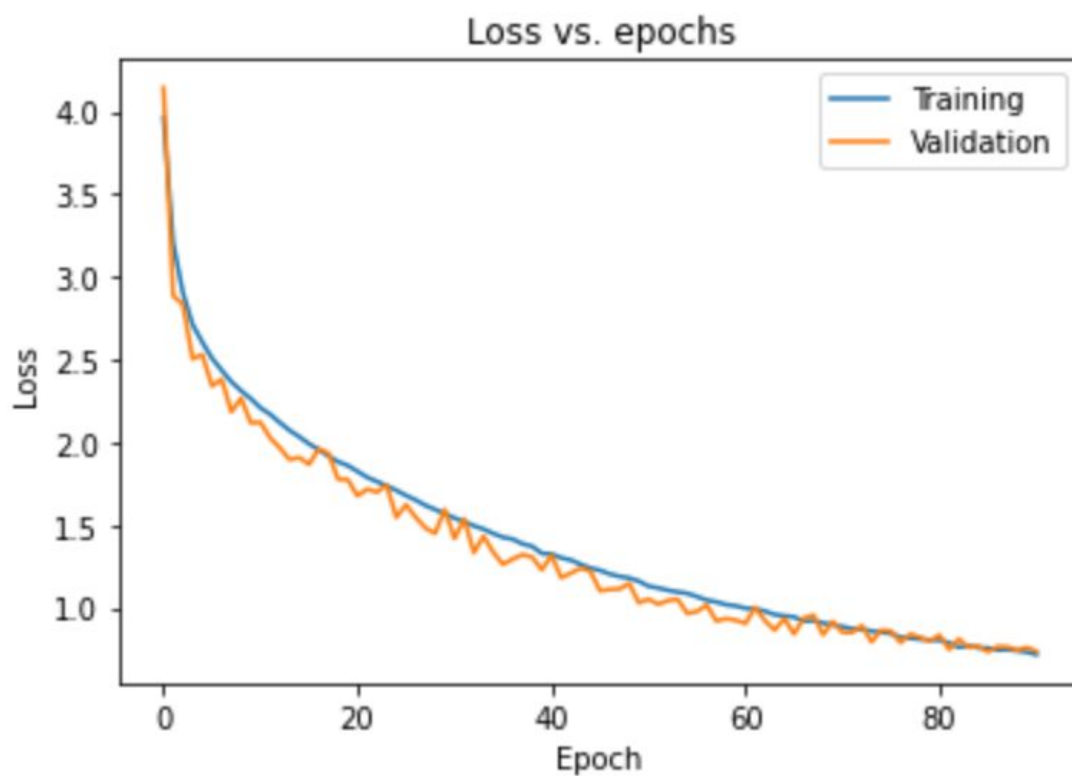
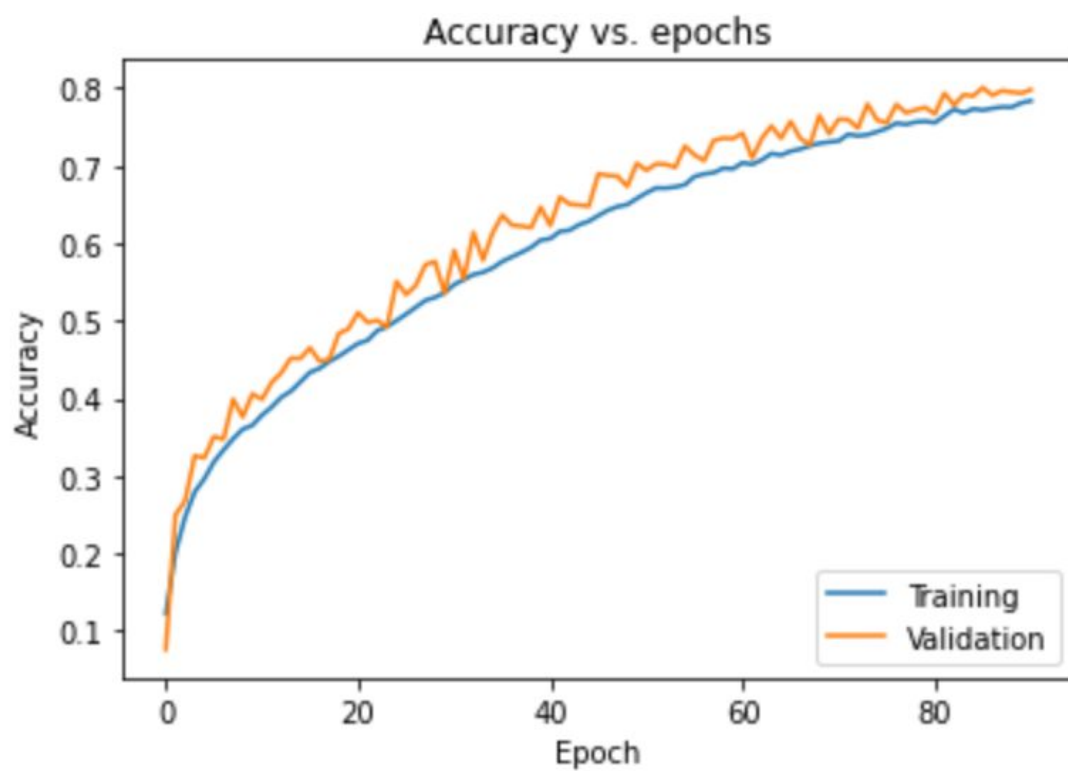
Mean Accuracy for the training dataset:

0.5949455665854307

Mean Loss for the training dataset:

1.4078370489916958

Unsmoothed Plots of Accuracy & Losses:



Transfer Learning Models - we may try one of these in the upcoming iterations

1. Transformers
2. Transformers with attention mechanism
3. BERT
4. ELMo

### How to improve your model performance?

Please note : the model runs above incorporate the results from the Data Augmentation techniques outlined below.

As noted earlier, our first model run with Logistic Regression using the preprocessed data could ONLY score 62% accuracy.

It became pretty clear immediately, we had to do something to improve the scores.

We researched on Data Augmentation techniques and some pointers provided by our mentor proved very useful. We researched these techniques:

- a. Translation based Data Augmentation - we pick each ticket , translate it into random language and translate it back to English. While this was powerful, the amount of time it took was huge and it did NOT provide the volume of data needed for our scores to make a difference.

For details about translation based Data Augmentation please refer to notebook: capstone-nlp-Data-Augmentation-by-Translation.ipynb

- b. Spacy based technique - we used spacy to find synonyms of words. The technique is to substitute random words in the ticket with their synonyms.

This again took too much time for spacy to generate synonyms, so we had to abandon it. We think the above provide very good potential, but given the time we had we had to find another technique. For details about translation based Data Augmentation please refer to notebook: [capstone-nlp-Data-Augmentation-by-spaCy.ipynb](#)

- c. We used KNN to find the synonyms - These are technically not synonyms but the words are very close based on their 'distance' from the candidate word for which we are finding the synonym.

Examples:

'reset': ['disable', 'restart', 'manually', 'disconnect'],  
'issue': ['problem', 'question', 'concern', 'concerned'],  
'unable': ['cannot', 'able', 'failing', 'however'],  
'you': ['sure', 'want', 'can', 'know'],  
'have': ['they', 'could', 'already', 'would'],  
'user': ['login', 'automatically', 'functionality', 'interface'],  
'my': ['me', 'myself', 'own', 'got'],  
'error': ['incorrect', 'invalid', 'problem', 'exception'],  
'access': ['accessible', 'allow', 'provide', 'secure'],  
'hello': ['hi', 'hey', 'yes', 'dear'],  
'be': ['should', 'being', 'not', 'will'],  
'account': ['payment', 'personal', 'same', 'however'],

- d. We also used Parts of Speech tagging in a similar way to identify synonyms.

Examples:

'trying': ['try'],  
'working': ['work'],  
'home': ['house'],  
'unlocking': ['unlock'],

```

'locked': ['lock'],
'getting': ['get'],
'need': ['want'],
'dynamics': ['dynamic'],
'regarding': ['regard'],
'phone': ['telephone'],
'client': ['customer'],
'duplication': ['duplicate'],
'gentle': ['soft'],
'display': ['showing'],

```

For the detailed code for synonyms using KNN & POS tagging, please refer to the notebook:

[capstone-nlp-Data-Augmentation-by-Word-Embeddings.ipynb](#)

Because using KNN technique, we could find multiple synonyms for a word, we settled on using the synonyms from KNN to create Tickets where we subtitled a random word in the ticket with its synonym keeping the label the same.

For example, in the below sample, the word accessible was changed to reachable, so the same meaning is retained yet it helped us create a new ticket for the same group.

<b>accessible</b> hand language icon customer number telephone	GRP_3
<b>reachable</b> hand language explorer customer number telephone	GRP_3

We also excluded the majority group GRP\_0 and concentrated on increasing the number of records for the minority groups.

Fig 12 - Distribution before Augmentation



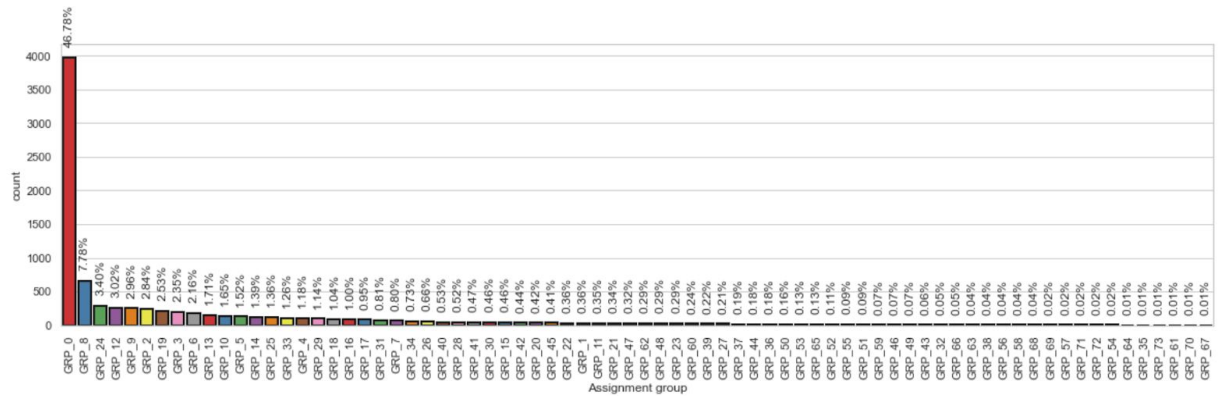
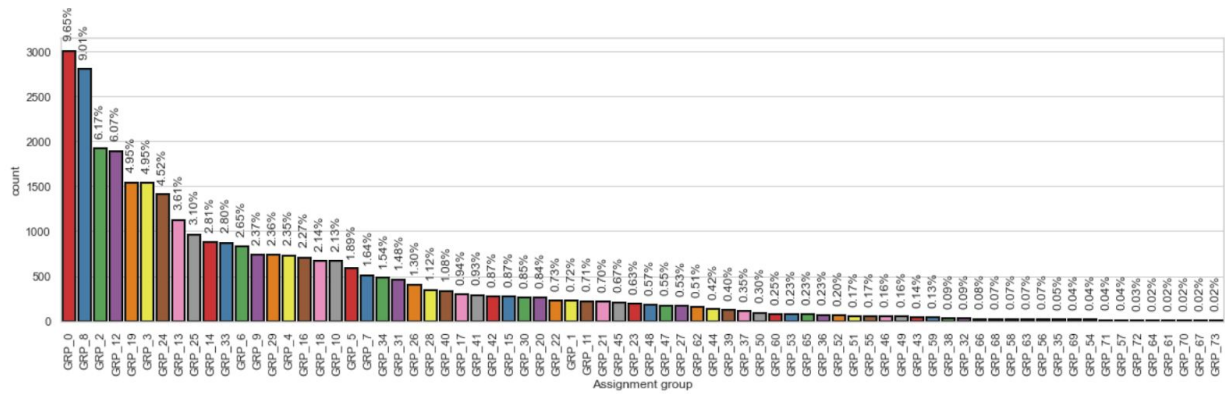



Fig 13 - Distribution after Augmentation




As the graphs indicate, we were able to significantly increase the representation of the minority groups. The initial results are promising. A sample classification report below shows that there is test data in every group:

	precision	recall	f1-score	support
0	0.63	0.71	0.67	919
1	0.85	0.78	0.81	64
2	0.94	0.88	0.91	193
3	0.96	0.87	0.91	78
4	0.84	0.84	0.84	594
5	0.95	0.93	0.94	327
6	0.91	0.84	0.87	242
7	1.00	0.87	0.93	84



8	0.92	0.93	0.92	224
9	0.99	0.95	0.97	73
10	0.97	0.94	0.96	186
11	0.84	0.84	0.84	447
12	0.86	0.90	0.88	585
13	1.00	0.95	0.97	78
14	0.95	0.93	0.94	61
15	0.99	0.88	0.93	75
16	0.88	0.91	0.89	55
17	0.80	0.90	0.85	433
18	0.92	0.97	0.95	265
19	0.94	0.81	0.87	113
20	0.90	0.84	0.87	45
21	0.96	0.78	0.86	105
22	0.97	0.89	0.93	236
23	0.83	0.86	0.84	469
24	0.94	0.82	0.88	83
25	0.83	0.74	0.78	148
26	1.00	0.80	0.89	10
27	0.92	0.87	0.90	269
28	0.85	0.85	0.85	142
29	1.00	1.00	1.00	2
30	0.89	0.63	0.74	27
31	1.00	0.91	0.96	35
32	1.00	1.00	1.00	10
33	0.67	0.59	0.62	34
34	0.92	0.86	0.89	237
35	0.96	0.93	0.94	97
36	0.96	0.95	0.96	79



37	0.95	0.93	0.94	87
38	1.00	0.92	0.96	13
39	1.00	0.92	0.96	26
40	0.96	0.82	0.89	66
41	1.00	1.00	1.00	16
42	0.69	0.48	0.56	42
43	0.96	0.89	0.93	57
44	1.00	0.92	0.96	12
45	0.62	0.61	0.61	155
46	0.85	0.74	0.79	31
47	1.00	0.94	0.97	16
48	1.00	0.88	0.93	16
49	0.89	0.70	0.78	23
50	1.00	1.00	1.00	2
51	0.81	1.00	0.90	13
52	0.75	0.60	0.67	5
53	1.00	0.40	0.57	5
54	1.00	0.75	0.86	4
55	1.00	0.93	0.96	14
56	0.68	0.81	0.74	241
57	0.58	0.32	0.41	22
58	1.00	1.00	1.00	1
59	0.95	0.76	0.84	46
60	1.00	0.75	0.86	4
61	1.00	1.00	1.00	2
62	0.90	0.95	0.93	20
63	0.80	1.00	0.89	4
64	1.00	1.00	1.00	1
65	1.00	0.40	0.57	5

66	1.00	0.50	0.67	6
67	0.96	0.95	0.96	151
68	1.00	1.00	1.00	2
69	1.00	0.67	0.80	3
70	0.00	0.00	0.00	1
71	1.00	1.00	1.00	2
72	0.78	0.85	0.82	852
73	0.80	0.64	0.71	250
accuracy			0.84	9340
macro avg	0.90	0.82	0.85	9340
weighted avg	0.85	0.84	0.84	9340

	Algorithm	Accuracy	f1	Precision Score	Recall Score
1	SVC 30% Test Data after Data Augmentation Word Embedding Round 2	91.43%	91.44%	94.37%	89.24%
2	Stacker 30% Test Data after Data Augmentation Word Embedding Round 2	90.50%	90.46%	93.74%	88.29%
3	KNN 30% Test Data after Data Augmentation Word Embedding Round 2	88.81%	88.70%	89.36%	90.07%
4	XGBoost Classifier 30% Test Data after Data Augmentation Word Embedding Round 2	84.06%	84.16%	90.65%	83.31%
5	Bagging Classifier 30% Test Data after Data Augmentation Word Embedding Round 2	83.94%	83.94%	92.17%	84.95%
6	DT 30% Test Data after Data Augmentation Word Embedding Round 2	78.37%	78.35%	82.03%	79.56%
7	LR 30% Test Data after Data Augmentation Word Embedding Round 2	76.48%	76.10%	78.98%	76.77%

We were able to increase the Accuracy of Logistics Regression from 62% to 76.48% after a couple of rounds of Data Augmentation. SVC topped the charts with 91.43 %.

We also ran the ensembles but their results were not too impressive. We will need to look into the reasons. Attached box plot shows the different Machine Learning Algorithms. The SVC & LR score the highest

