2020

Automatic Ticket Assignment using Deep Learning - Natural Language Processing (NLP)

Bhabani Acharya

Karthik Ranganathan

Rajaram Chinnayan

Table of Contents

1. Executive Summary (Summary of Problem Statement, data and findings)

In the current world, support ticket systems have risen in prominence and have become ubiquitous. Assigning the ticket to the correct group is crucial to improve user experience, better allotment of support resources and quicker turnaround time. Multi-choice systems which asks the users to choose categories and sub-categories are considered better than the vanilla ones, but they face the dual challenges of catering to new users, who do not have the know how to select the required fields and user experience as most users are not keen on selecting fields by going through long dropdown or searching exhaustively for a category. In this study, an automated ticket assignment system that uses natural language processing techniques like Count Vectorization, Word2Vec along with deep learning algorithms are proposed which aims to provide seamless user experience, high quality support, reduce manual labour and human errors and improved turn-around times.

Below is summary of the results with machine learning models.

Accuracies of models like LR, RF and SVM's are decent and they have good precision and recall.

SL	MODEL_NAME	ACCURACY_SCORE	PRECISION_SCORE	RECALL_SCORE	F1_SCORE
0	Logistic Regression	82.68	0.88	0.81	0.82
1	Stochastic Gradient Descent	81.46	0.87	0.79	0.81
2	Random Forest	84.26	0.93	0.83	0.85
3	Decsision Tree	80.01	0.81	0.83	0.81
4	Support Vector Machines	83.17	0.85	0.85	0.84
5	AdaBoost	36.24	0.99	0.09	0.09
6	Gaussian Naive Bayes	68.39	0.70	0.83	0.74
7	K Nearest Neighbor	65.88	0.77	0.57	0.60

Below do several deep learning models predict summary of the results as. Results are not as encouraging as the machine learning models with only the Simple RNN and the GRU CNN delivering comparable results

Sl	Model	AUC_Score	Accuracy	Recall	Precision	F1_Score
0	RNN	93.40	73.83	63.82	89.24	74.42
1	Bi LSTM	93.24	79.76	27.33	98.49	42.78
2	GLOVE LSTM	92.94	71.07	31.28	94.46	47.00
3	GRU CNN	92.85	80.85	45.28	70.88	55.26
4	GRU GLOVE	92.65	58.16	31.77	79.28	45.36
5	GRU	92.19	72.10	26.75	74.53	39.37

2. Overview of the final process

2.1 Problem Statement

One of the key activities of any IT function is to "Keep the lights on" to ensure there is no impact to the Business operations. IT leverages Incident Management process to achieve the above Objective. An incident is something that is unplanned interruption to an IT service or reduction in the quality of an IT service that affects the Users and the Business. The main goal of Incident Management process is to provide a quick fix / workarounds or solutions that resolves the interruption and restores the service to its full capacity to ensure no business impact.In most of the organizations, incidents are created by various Business and IT Users, End Users/ Vendors if they have access to ticketing systems, and from the integrated monitoring systems and tools. Assigning the incidents to the appropriate person or unit in the support team has critical importance to provide improved user satisfaction while ensuring better allocation of support resources. The assignment of incidents to appropriate IT groups is still a manual process in many of the IT organizations. Manual assignment of incidents is time consuming and requires humanefforts. There may bemistakes due to human errors and resource consumption is carried out ineffectively because ofthe misaddressing. On the other hand, manual assignment increases the response and resolution times which result in user satisfaction deterioration / poor customer service.

2.2 Business Domain Value

In the support process, incoming incidents are analysed and assessed by organization's support teams to fulfil the request. In many organizations, better allocation and effective usage of the valuable support resources will directly result in substantial cost savings. Currently the incidents are created by various stakeholders (Business Users, IT Users and Monitoring Tools) within IT Service Management Tool and are assigned to Service Desk teams (L1/ L2 teams). This team will review the incidents for right ticket categorization, priorities and then carry out initial diagnosis to see if they can resolve. Around ~54% of the incidents are resolved by L1 / L2 teams. If L1 / L2 is unable to resolve, they will then escalate / assign the tickets to Functional teams from Applications and Infrastructure (L3 teams). Some portions of incidents are directly assigned to L3 teams by either Monitoring tools or Callers / Requestors. L3 teams will carry out detailed diagnosis and resolve the incidents. Around ~56% of incidents are resolved by Functional / L3 teams. Incase if vendor support is needed, they will reach out for their support towards incident closure.L1 / L2 needs to spend time reviewing Standard Operating Procedures (SOPs) before assigning to Functional teams (Minimum ~25-30% of incidents needs to be reviewed for SOPs before ticket assignment). 15 min is being spent for SOP review for each incident. Minimum of ~1 FTE effort needed only for incident assignment to L3 teams. During the process of incident assignments by L1 / L2 teams to functional groups, there were multiple instances of incidents being assigned to wrong functional groups. Around ~25% of Incidents are wrongly assigned to functional teams. Additional effort needed Functional teams to re-assign to right functional groups. During this process, some of the incidents are in queue and not addressed timely resulting in poor customer service. Guided by powerful AI techniques that can classify incidents to right functional groups can help organizations to reduce the resolving time of the issue and can focus on tasks that are more productive?

2.3 Steps in the Algorithm

The algorithm follows the below steps which is typically used for most NLP Classification tasks

2.3.1 Data Processing

Requisite processing to clean the input data which includes tasks like lowercasing all words, removing extra spaces, removing punctuations, correcting spellings, removing special characters and words like email address etc.

2.3.2 NLP techniques like TF-IDF, N-grams, Word Embeddings

Below is a small primer to these key concepts that has also been used by our solution

1. ***TF-IDF in NLP stands for Term Frequency*** – Inverse document frequency. It is a statistical measure that evaluates how relevant a word is to a document in a collection of documents. This is done by multiplying two metrics: how many times a word appears in a document, and the inverse document frequency of the word across a set of documents.It has many uses, most importantly in automated text analysis, and is very useful for scoring words in machine learning algorithms for Natural Language Processing (NLP). As machine learning algorithms deal with numbers. So, we need to transform text into numbers. This step is known as Text Vectorization. The Tf-IDF score can be fed into algorithms like Support vector Machines, Naïve Bayes, Random Forest[1]

2. ***N-grams***: Statistical language models, in its essence, are the type of models that assign probabilities to the sequences of words. NLP techniques involve assigning probabilities to sentences and sequences of words, the n-gram . N-gram is nothing but a sequence of N words, by that notion, a 2-gram (or bigram) is a two-word sequence of words like "please listen", "my house" and 3-gram (trigram) is a three word sequence like "please listen to", "my house is". The Bigram Model, as the name suggests, approximates the probability of a word given all the previous words by using only the conditional probability of one preceding word. You can further generalize the bigram model to the trigram model which looks two words into the past and can thus be further generalized to the N-gram model[2]

3. ***Word Embedding***: A distributed representation of words where different words that have a similar meaning (based on their usage) also have a similar representation. A word embedding is a learned representation for text where words that have the same meaning have a similar representation. It is this approach to representing words and documents that may be considered one of the key

[1] https://monkeylearn.com/blog/what-is-tf-idf/
[2] https://towardsdatascience.com/introduction-to-language-models-n-gram-e323081503d9

breakthroughs of deep learning on challenging natural language processing problems[3]

I. **Visualization -**

We will then use of a topic model algorithm like LDA to see distribution of words in a dynamic 2D chart using pyLDAvis. *pyLDAvis is designed to help users interpret the topics in a topic model that has been fit to a corpus of text data. The package extracts information from a fitted LDA topic model to inform an interactive web-based visualization.

The visualization is intended to be used within an IPython notebook but can also be saved to a stand-alone HTML file for easy sharing.

2.3.3 Evaluation using Machine Learning and Deep Learning Models -

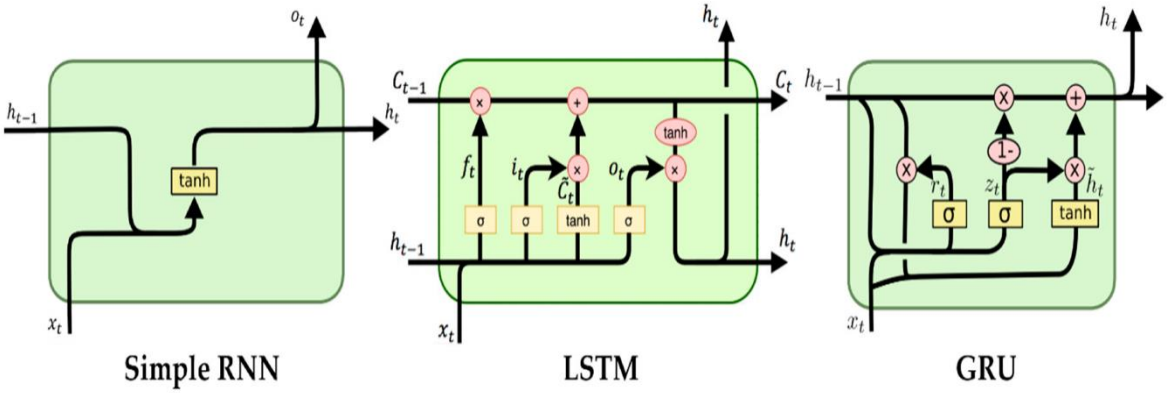
Once the input data is processed and analysed using above NLP techniques, the data is fed through different ML algorithms like

- Logistic Regression
- SVM
- Decision Tree
- Random Forest
- K nearest neighbours
- Adaboost

2.3.4 Results are summarized using metrics like Accuracy, Precision, Recall, AUC

This step is followed by generating word embeddings, post which Deep Learning Models like

- Simple Neural Network
- LSTM
- Bi-LSTM
- LSTM – Using GLOVE
- GRU
- GRU – Using GLOVE



[3] https://machinelearningmastery.com/what-are-word-embeddings/#:~:text=A%20word%20embedding%20is%20a,challenging%20natural%20language%20processing%20problems.

3. Step-by-step walk through the solution

1.1 Data Processing

Details about the data and dataset files are given in below link

https://drive.google.com/open?id=1OZNJm81JXucV3HmZroMq6qCT2m7ez7IJ

On inspection of the dataset, the data consists of 4 main columns

i. Short Description: - Shorter version of the Ticket/Complaint raised
ii. Description: Detailed Description of the Complaint/Ticket Raised
iii. Caller: Name of the person/s who raised the ticket
iv. Assignment Group: The Output Parameter corresponding to the group which is supposed to handle the ticket/complaint

The objective of this project is to automate the process of assigning a ticket/complaint to the right Assignment group based on Artificial Intelligence techniques, particularly Natural Language processing (Deep Learning) techniques.

	Short description	Description	Caller	Assignment group
0	login issue	-verified user details.(employee# & manager na...	spxjnwirpjlcoqds	GRP_0
1	outlook	\n\nreceived from: hmjdrvpb.komuaywn@gmail.com...	hmjdrvpbkomuaywn	GRP_0
2	cant log in to vpn	\n\nreceived from: eylqgodm.ybqkwiam@gmail.com...	eylqgodmybqkwiam	GRP_0
3	unable to access hr_tool page	unable to access hr_tool page	xbkucsvzgcpydteq	GRP_0
4	skype error	skype error	owlgqjmeqhcozdfx	GRP_0

Snapshot of the data set

The following steps were implemented as part of pre-processing -

i. **Dealing with Missing data** –
   a. 9 rows out of 8500 were identified as missing.
   b. 1 row of Detailed Description was missing and 8 rows of Short Description were missing.

Since, number of rows of missing data, were miniscule, an easy approach of replacing the Description with Short Description was applied. As further steps mainly involved using only the Description as the Primary Input, so replacing the Single (1) Row of Description with Short Description was the easiest and practical solution.

ii. **Identifying the different languages in the Input dataset** –

a. The data set was multi lingual with English being the most common language.

Since English is the most common language by a large margin, we decided to translate the non-English parts to English and then pass the data through the model. Although we did considered using different models for different languages and using mixed language word embedding, owning to the fact that more than 84% of the data was in English, we decided to translate the non-English ones. We used google translate for the translation.

```
# how many unique language labels were applied?
print("Number of tagged languages (estimated):")
print(len(langs.unique()))
# percent of the total dataset in English
print("Percent of data in English (estimated):")
print((sum(langs=="en")/len(langs))*100)
```

```
Number of tagged languages (estimated):
43
Percent of data in English (estimated):
84.65882352941176
```

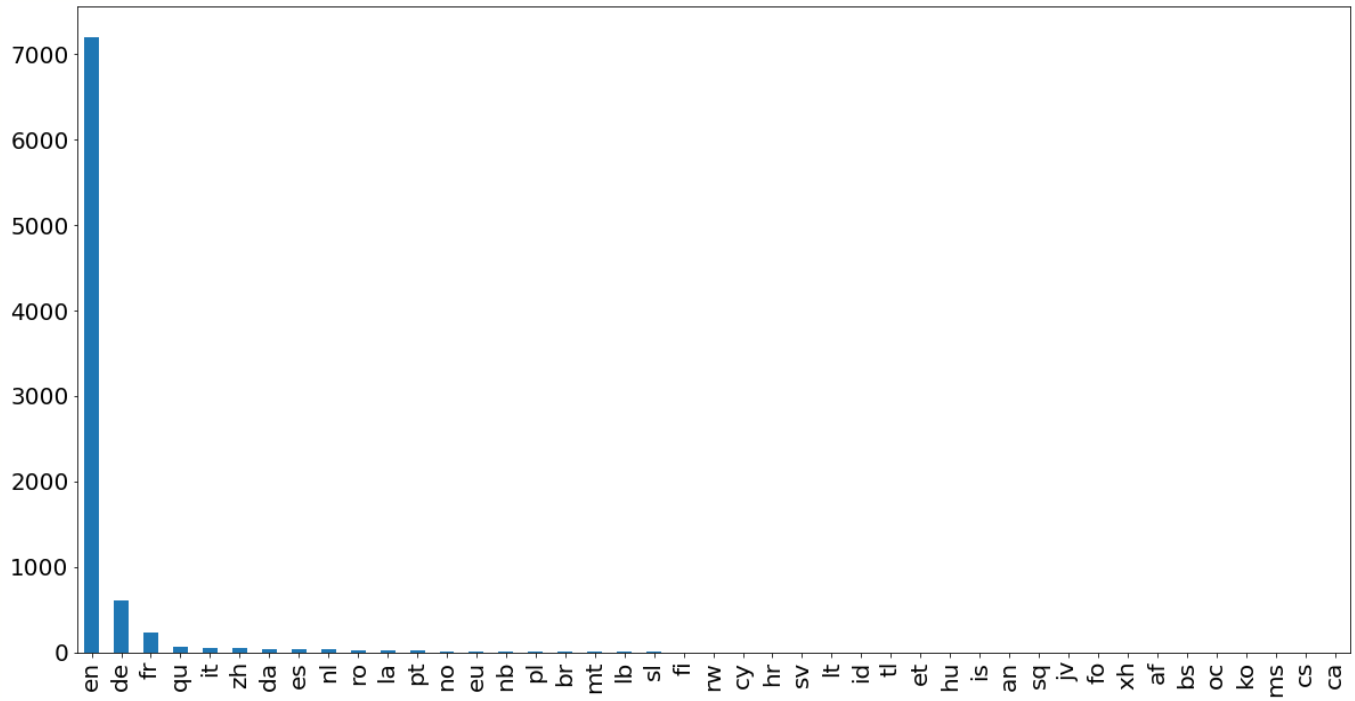


Figure 1: Distribution of different languages

iii. **Identifying the different groups in the Input dataset** –

a. There were 74 assignment groups with group 0 being the group to which maximum number of tickets were assigned.

Since more than half of the tickets assigned to group 0, we need to keep in mind that this is an imbalanced data set.

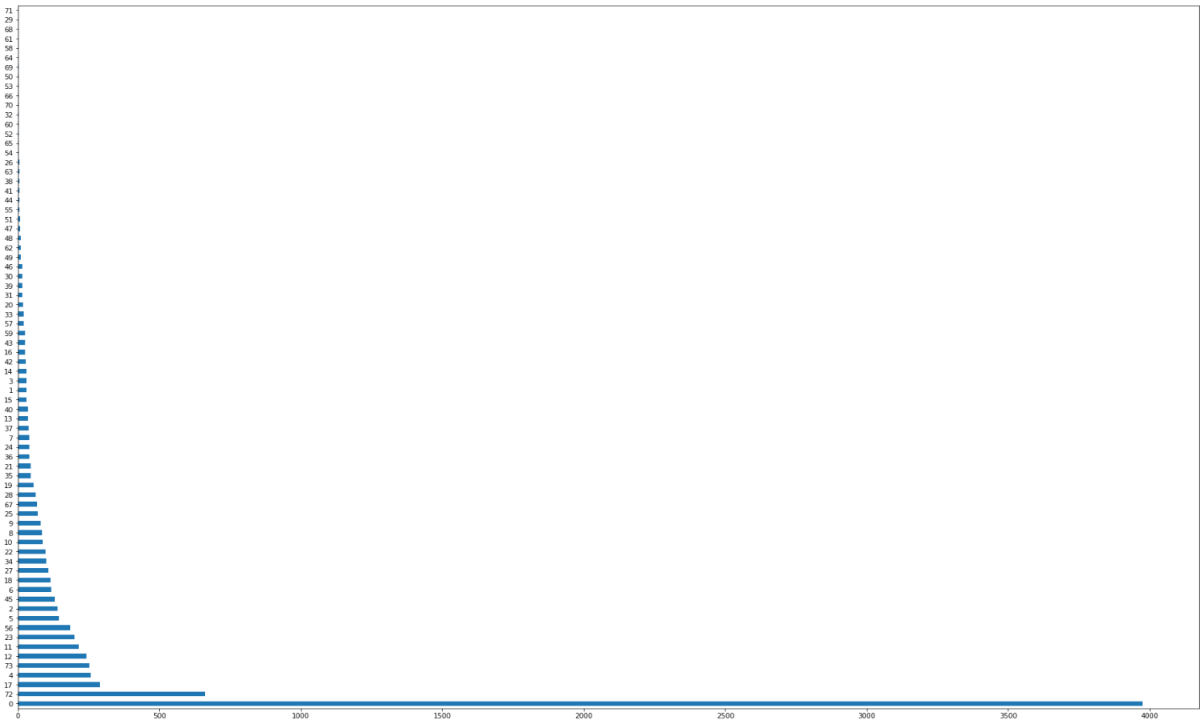


Figure 2: Number of tickets assigned to each group

Post this the following pre-processing/clean-up of the dataset

i. Converting all characters to lower case
ii. Converting Unicode characters to ascii
iii. Removal of disclaimers
iv. Removing duplicate words in sentence
v. Removing newline and return characters
vi. Removing date and time stamp
vii. Removing punctuations
viii. Removing email
ix. Removing special characters, symbols and whitespace
x. Lemmatize and Removing stop words
xi. Also removal of First Name and Last name
xii. Removal of Noise and spelling mistakes like aa* etc.

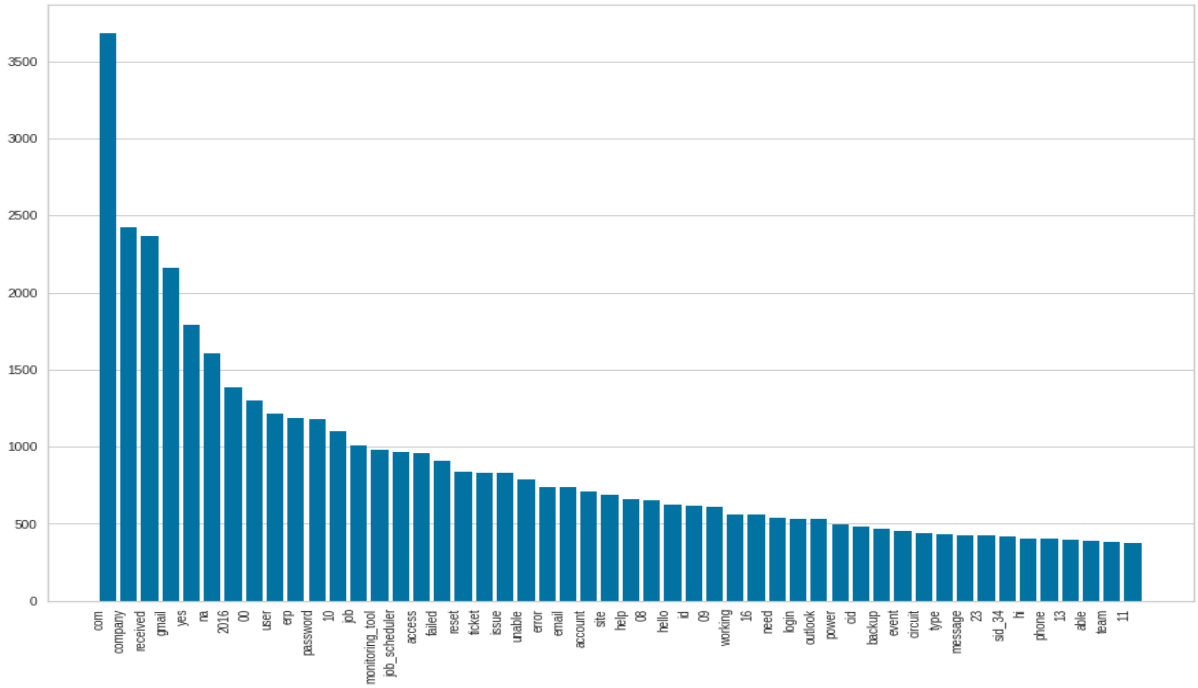


Figure 3: Most commonly used words before pre-processing

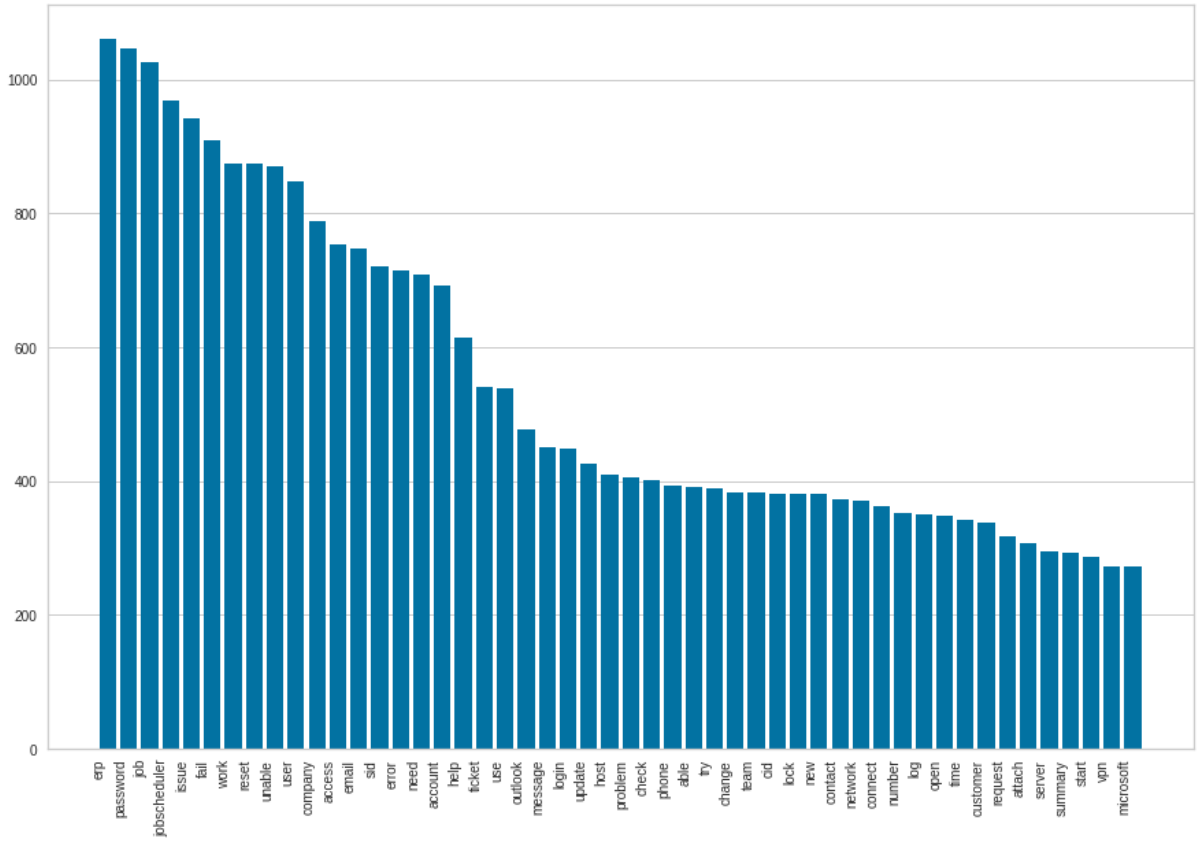


Figure 4: Most commonly used words after pre-processing

1.2 NLP techniques like TF-IDF, N-grams, Word Embeddings

Count Vectorizer is used to generate vectors on our clean dataframe using the below code

```
from sklearn.feature_extraction.text import CountVectorizer

from yellowbrick.text import FreqDistVisualizer

from gensim import corpora

import warnings
warnings.filterwarnings("ignore", category=FutureWarning)

text_data = tic_df['clean_desc'].tolist()

vectorizer = CountVectorizer(stop_words='english')
docs       = vectorizer.fit_transform(text_data)
features   = vectorizer.get_feature_names()

visualizer = FreqDistVisualizer(features=features, orient='v', size=
(1080, 720))
visualizer.fit(docs)
```

The features are then extracted from the vectorised data and the sorted TF-IDF score is printed.

```
feature_names = cv.get_feature_names()

#get tfidf vector for first document
first_document_vector=tf_idf_vector[20]

#print the scores
df = pd.DataFrame(first_document_vector.T.todense(), index=feature_n
ames, columns=["tfidf"])
df.sort_values(by=["tfidf"],ascending=False).tail(20)
```

	Tfidf
help contact	0.0
help connect	0.0
help company	0.0
help clear	0.0
height	0.0
help	0.0
help access	0.0
help account	0.0

And we build vocabulary using below code

```
from tqdm import tqdm

tqdm.pandas()

def build_vocab(sentences, verbose =  True):
    """
    :param sentences: list of list of words
    :return: dictionary of words and their count
    """
    vocab = {}
    for sentence in tqdm(sentences, disable = (not verbose)):
        for word in sentence:
            try:
                vocab[word] += 1
            except KeyError:
                vocab[word] = 1
    return vocab

sentences = tic_df['clean_desc'].apply(lambda x: x.split()).values
vocab = build_vocab(sentences)
print({k: vocab[k] for k in list(vocab)[:5]})
```

We also check coverage of the same

```
sort_orders = sorted(vocab.items(), key=lambda x: x[1], reverse=True)

import operator

def check_coverage(vocab,embeddings_index):
    a = {}
    oov = {}
    k = 0
    i = 0
    for word in tqdm(vocab):
        try:
            a[word] = embeddings_index[word]
            k += vocab[word]
        except:

            oov[word] = vocab[word]
            i += vocab[word]
            pass

    print('Found embeddings for {:.2%} of vocab'.format(len(a) / len(vocab)
))
    print('Found embeddings for  {:.2%} of all text'.format(k / (k + i)))
    sorted_x = sorted(oov.items(), key=operator.itemgetter(1))[::-1]

    return sorted_x
oov = check_coverage(vocab,embeddings_index)
```

```
100%|██████████| 10125/10125 [00:00<00:00, 317012.62it/s]
Found embeddings for 54.70% of vocab
Found embeddings for  95.29% of all text
```

Result is encouraging as we have embeddings for 95% of all text

We will now try using another algorithm to increase the coverage. For this, we will use the fasttext algorithm

```
from gensim.models import Word2Vec
from gensim.models import FastText
import pandas as pd
from tqdm import tqdm
tqdm.pandas()

docs = tic_df['clean_desc']
token_doc = [[text for text in doc.split()] for doc in docs]
embedding_size = 300
window_size = 5
min_word = 1


ft_model = FastText(token_doc,
                    size=embedding_size,
                    window=window_size,
                    min_count=min_word,
                    sg=1,negative=15,
                    iter=10)

oov = check_coverage(vocab,ft_model)

100%|██████████| 10122/10122 [00:00<00:00, 165869.30it/s]
Found embeddings for 100.00% of vocab
Found embeddings for  100.00% of all text
```

As you can see, with fasttext, coverage has increased to 100%of all text

Post this, we generate the corpus using

```
from gensim import corpora

text_data = tic_df['clean_desc'].apply(word_tokenize)

dictionary = corpora.Dictionary(text_data)
corpus = [dictionary.doc2bow(text) for text in text_data]

import pickle
pickle.dump(corpus, open('corpus.pkl','wb'))
dictionary.save('dictionary.gensim')
```

1.3 Visualization

We then used the popular Topic Model Algorithm LDA to generate topics and show visualization of the topics generated.

We also generate the perplexity and coherence score

```
from gensim.utils import simple_preprocess
from gensim.models import CoherenceModel

# Enable logging for gensim - optional
import logging
logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)
s', level=logging.ERROR)


# Compute Perplexity
print('\nPerplexity: ', result.log_perplexity(corpus))  # a measure
of how good the model is. lower the better.

# Compute Coherence Score
coherence_model_lda = CoherenceModel(model=result, texts=text_data,
dictionary=dictionary, coherence='c_v')
coherence_lda = coherence_model_lda.get_coherence()
print('\nCoherence Score: ', coherence_lda)
```

```
Perplexity:  -7.9514839163341335

Coherence Score:  0.4960934668645882
```

Result is encouraging as perplexity is low and coherence is ~50%

Visualization of same gives

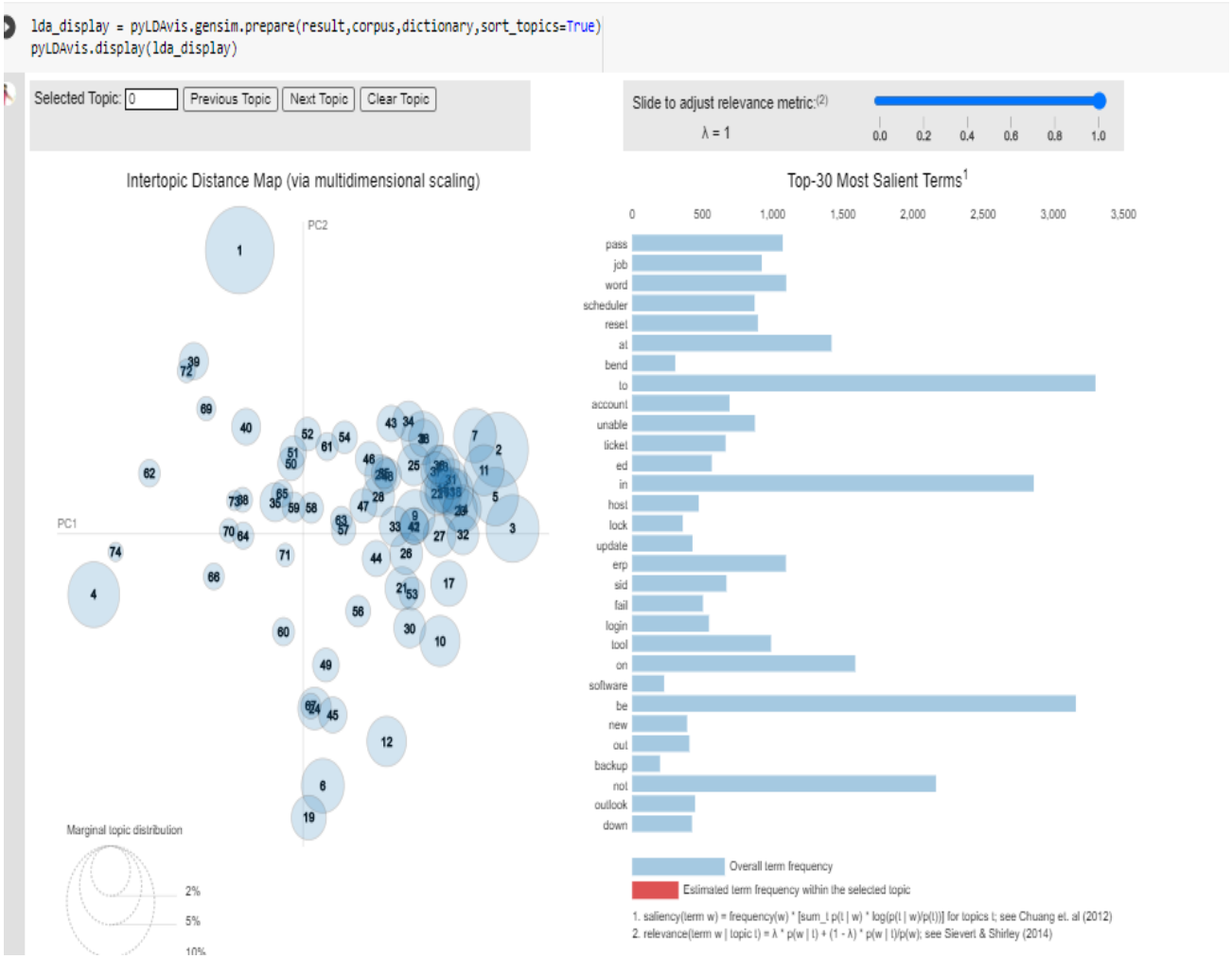


Figure 5: Most salient words per group

We also generated bigrams and trigrams. The below visualization shows the relationships between commonly used bigrams and groups -

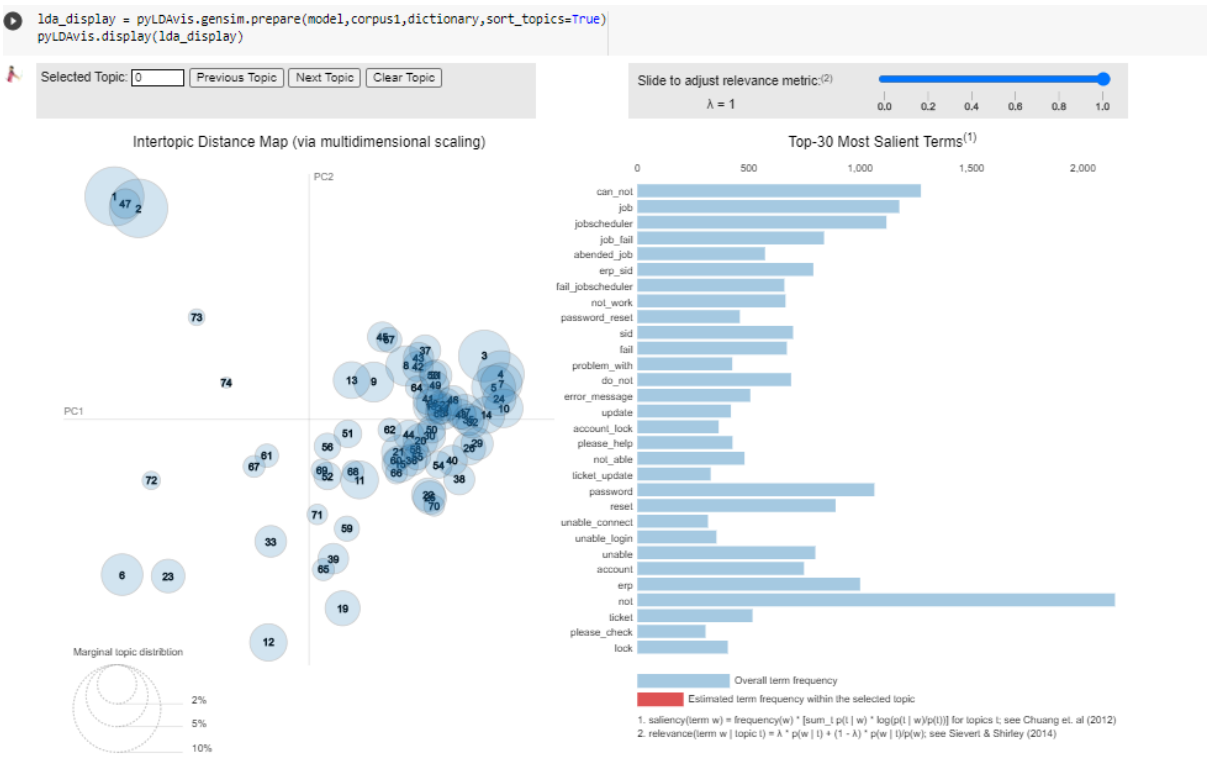


Figure 6: Most salient bigrams per group

The clustering of output varies in unigram versus bi-gram as seen in visualization indicating potential of different results using the different algorithms

1.4 Reducing the class Imbalance

We will try simple technique to reduce class imbalance. We tried using over sampling techniques like SMOTE. However this was crashing the system as memory was not sufficient. So we will manually replicate the samples with lower count

```
crit = tic_df.groupby(['Assignment_group']).size() < 10
crit[crit==True]
Assignment_group
32    True
52    True
54    True
60    True
65    True
dtype: bool
```

```
replica = tic_df[tic_df['Assignment_group'].isin([32,52,54,60,65])]
crit = tic_df.groupby(['Assignment_group']).size() < 50
replica1 = tic_df[tic_df['Assignment_group'].isin([26,31,32,38,39,41
,44,46,47,48,49,51,52,54,55,60,62,63,65])]
df = tic_df.append(replica)
tic_df = df.append(replica1)

tic_df.to_csv('ReplicatedFinal.csv')
tic_df.shape
(15533, 9)
```

As can be seen, now we have approximately double the samples

4. Model Evaluation

Machine learning for NLP and text analytics involves a set of statistical techniques for identifying parts of speech, entities, sentiment, and other aspects of text. The techniques can be expressed as a model that is then applied to other text, also known as supervised machine learning. Text data requires a special approach to machine learning. This is because text data can have hundreds of thousands of dimensions (words and phrases) but tends to be very sparse. For example, the English language has around 100,000 words in common use. In supervised machine learning, a batch of text documents are tagged or annotated with examples of what the machine should look for and how it should interpret that aspect. These documents are used to "train" a statistical model, which is then given un-tagged text to analyze. The most popular supervised NLP machine learning algorithms are:

- Support Vector Machines
- Bayesian Networks
- Neural Networks/Deep Learning

Before running the text input, tokens need to be generated. Tokenization involves breaking a text document into pieces that a machine can understand, such as words. Now, you're probably pretty

good at figuring out what's a word and what's jibbersih. English is especially easy. See all this white space between the letters and paragraphs? That makes it easy to tokenize. Therefore, NLP rules are sufficient for English tokenization. Further, we categorize and classify. Categorization means sorting content into buckets to get a quick, high-level overview of what is in the data. To train a text classification model, data scientists use pre-sorted content and gently shepherd their model until it has reached the desired level of accuracy. The result is accurate, reliable categorization of text documents that takes far less time and energy than human analysis.

Post pre-processing and visualization, the clean data was analysed using various algorithms and the code and result is given below –

```
model_dict = {'Logistic Regression' : LogisticRegression(max_iter=1000),
        'Stochastic Gradient Descent' : SGDClassifier(random_state=3, loss='log'),
        'Random Forest': RandomForestClassifier(random_state=3),
        'Decsision Tree': DecisionTreeClassifier(random_state=3),
        'Support Vector Machines': LinearSVC(max_iter=10000),
        'AdaBoost': AdaBoostClassifier(random_state=3),
        'Gaussian Naive Bayes': GaussianNB(),
        'K Nearest Neighbor': KNeighborsClassifier()
        }
```

The result of all the tested non-neural network models

SL	MODEL_NAME	ACCURACY_SCORE	PRECISION_SCORE	RECALL_SCORE	F1_SCORE
0	LOGISTIC REGRESSION	82.68	0.88	0.81	0.82
1	STOCHASTIC GRADIENT DESCENT	81.46	0.87	0.79	0.81
2	RANDOM FOREST	84.26	0.93	0.83	0.85
3	DECSISION TREE	80.01	0.81	0.83	0.81
4	SUPPORT VECTOR MACHINES	83.17	0.85	0.85	0.84
5	ADABOOST	36.24	0.99	0.09	0.09
6	GAUSSIAN NAIVE BAYES	68.39	0.70	0.83	0.74
7	K NEAREST NEIGHBOR	65.88	0.77	0.57	0.60

The Machine Learning models like Random Forest and Support vector machines perform very well with Random Forest yielding the best results

1.5 Deep Learning Models

When we are using Natural Language Processing to deal with text data, we have to keep in mind that the data is the form of a sequence and hence the order matters. For a sentence, if the words are arranged differently the meaning might be very different. In summary, the data has two categories of information, one the words themselves and second, the order is which the words appear. Linear models can only capture the information provided by the words; however, they cannot extract the information present in the word ordering. To be able to do so we will use models that are able to capture the information from both. We will start with a simple RNN and then compare its performance to a basic LSTM and GRU.

Beyond this, few deep learning models are run on the same, result after running these algorithms are as below

1) First Model is a Simple RNN. Model Summary is below

```
Model: "sequential_14"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_14 (Embedding)     (None, 255, 300)          3030900
_________________________________________________________________
simple_rnn_6 (SimpleRNN)     (None, 100)               40100
_________________________________________________________________
dense_17 (Dense)             (None, 74)                7474
=================================================================
Total params: 3,078,474
Trainable params: 3,078,474
Non-trainable params: 0
```

The Model generated an AUC Score of 93.4 and accuracy of 73.8

The Confusion Matrix is below

Assignment Group	precision	recall	f1-score	support
2	0.35	0.23	0.27	31
4	0.36	0.23	0.28	62
5	0.09	0.03	0.05	33
6	0.17	0.11	0.13	19
11	0.12	0.08	0.09	38
12	0.35	0.18	0.24	62
18	0	0	0	28
22	0.56	0.8	0.66	55
23	0.11	0.07	0.09	41
27	0.36	0.24	0.29	17
34	0	0	0	25
45	0.46	0.54	0.5	24
47	0.25	1	0.4	2
53	0.48	1	0.65	90
56	0.24	0.2	0.22	35
73	0.12	0.02	0.04	47
accuracy			0.74	3107
macro avg	0.73	0.72	0.71	3107
weighted avg	0.71	0.74	0.71	3107

We would then use GloVE to generate the embeddings and then use a LSTM model

2) Second Model was a LSTM using embeddings from GloVE

```
Model: "sequential_18"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_18 (Embedding)     (None, 255, 300)          3030900
_________________________________________________________________
spatial_dropout1d_1 (Spatial (None, 255, 300)          0
_________________________________________________________________
lstm_6 (LSTM)                (None, 100)               160400
_________________________________________________________________
dense_21 (Dense)             (None, 74)                7474
=================================================================
Total params: 3,198,774
Trainable params: 167,874
Non-trainable params: 3,030,900
```

AUC_Score was 92.9 & Accuracy was at to 71

3) Third Model was a BI-LSTM using embeddings from GloVE

```
Model: "sequential_19"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_19 (Embedding)     (None, 255, 300)          3030900
_________________________________________________________________
bidirectional_1 (Bidirection (None, 200)               320800
_________________________________________________________________
dense_22 (Dense)             (None, 74)                14874
=================================================================
Total params: 3,366,574
Trainable params: 335,674
Non-trainable params: 3,030,900
```

AUC_Score was 93.2 & Accuracy was at to 79.7

4) Fourth was a GRU - CNN

```
Model: "sequential_20"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_20 (Embedding)     (None, 255, 300)          3030900
_________________________________________________________________
conv1d_6 (Conv1D)            (None, 255, 300)          270300
_________________________________________________________________
max_pooling1d_6 (MaxPooling1 (None, 85, 300)           0
_________________________________________________________________
dropout_8 (Dropout)          (None, 85, 300)           0
_________________________________________________________________
gru_2 (GRU)                  (None, 300)               541800
_________________________________________________________________
dropout_9 (Dropout)          (None, 300)               0
_________________________________________________________________
dense_23 (Dense)             (None, 74)                22274
=================================================================
Total params: 3,865,274
Trainable params: 834,374
Non-trainable params: 3,030,900
```

AUC_Score was 92.8 & Accuracy was at to 80.8

5) Fifth was a GRU

```
Model: "sequential_23"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_23 (Embedding)     (None, 255, 300)          3030900
_________________________________________________________________
gru_5 (GRU)                  (None, 100)               120600
_________________________________________________________________
dense_27 (Dense)             (None, 74)                7474
=================================================================
Total params: 3,158,974
Trainable params: 3,158,974
Non-trainable params: 0
```

AUC_Score was 92.1 & Accuracy was at to 72.1

6) Last Model was a GRU using GloVE

```
Model: "sequential_23"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_23 (Embedding)     (None, 255, 300)          3030900
_________________________________________________________________
gru_5 (GRU)                  (None, 100)               120600
_________________________________________________________________
dense_27 (Dense)             (None, 74)                7474
=================================================================
Total params: 3,158,974
Trainable params: 3,158,974
Non-trainable params: 0
```

AUC_Score was 92.6 & Accuracy was at to 58.1

As seen from results, last 2 models don't perform very well

5. Comparison to Benchmarks and Implications

Below is summary of the results, as predicted by several deep learning models tested.

SL	MODEL	AUC_SCORE	ACCURACY	RECALL	PRECISION	F1_SCORE
0	RNN	93.40	73.83	63.82	89.24	74.42
1	BI LSTM	93.24	79.76	27.33	98.49	42.78
2	GLOVE LSTM	92.94	71.07	31.28	94.46	47.00
3	GRU CNN	92.85	80.85	45.28	70.88	55.26
4	GRU GLOVE	92.65	58.16	31.77	79.28	45.36
5	GRU	92.19	72.10	26.75	74.53	39.37

If we compare the same to machine learning models, they do not perform very well as can be seen from the lower accuracy and poorer Recall and Precision

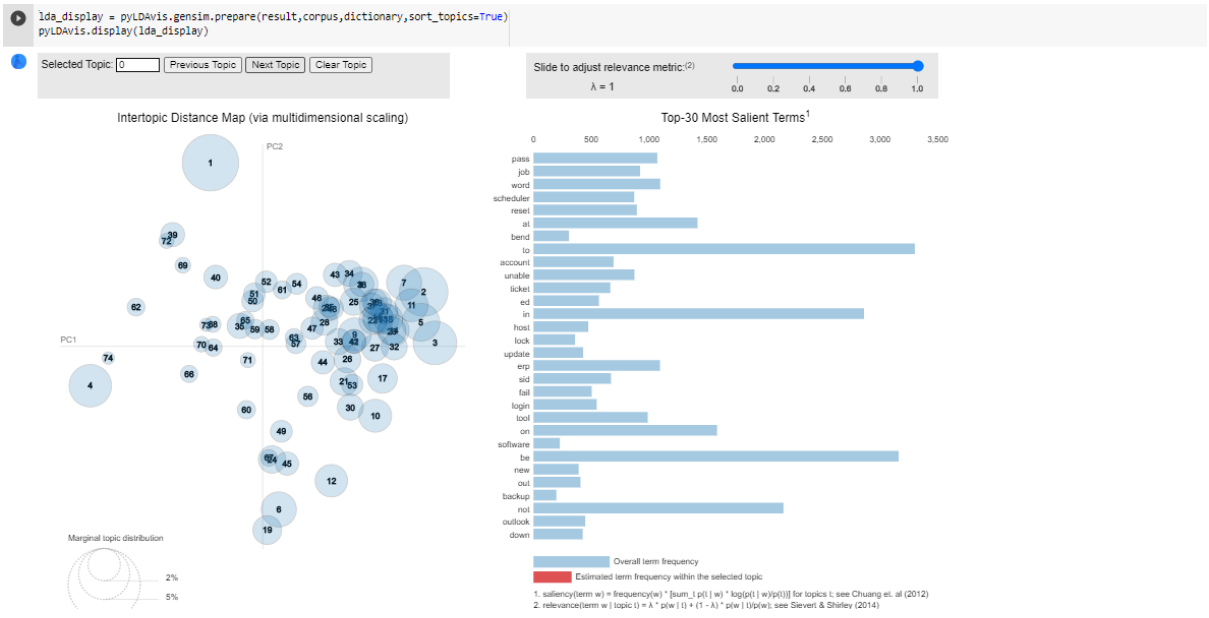
Sl	model_name	accuracy_score	precision_score	recall_score	f1_score
0	Logistic Regression	82.68	0.88	0.81	0.82
1	Stochastic Gradient Descent	81.46	0.87	0.79	0.81
2	Random Forest	84.26	0.93	0.83	0.85
3	Decsision Tree	80.01	0.81	0.83	0.81
4	Support Vector Machines	83.17	0.85	0.85	0.84
5	AdaBoost	36.24	0.99	0.09	0.09
6	Gaussian Naive Bayes	68.39	0.70	0.83	0.74
7	K Nearest Neighbor	65.88	0.77	0.57	0.60

Therefore, Implication are two

- **The Deep learning models can be tuned for better performance**
- **The Machine Learning algorithms generate decent results and can be hyper tuned further for better performance and then commercialized**

6. Visualizations

Visualizations were already included as part of detailed code walkthrough. The LDA display as shown below is one such very good visualization and very interactive if tried on the Jupiter Notebook



7. Implications

Implications are two fold

- The Deep learning models can be tuned for better performance
- The Machine Learning algorithms generate decent results and can be hyper tuned further for better performance and then commercialized

8. Limitations

All the deep learning models in this solution typically have low recall. This is undesirable if the costs of false negatives are very critical. Accuracy levels could have been a bit better.

.

9. Closing Reflections

It was interesting to work on a real-world problem that is faced by almost every tech institution. With the cost of labor increasing, the need for an automated solution based on Artificial Intelligence is the need of the hour.

The solution that we have proposed would definitely help organizations focus on more productive work like resolving the customer complaint in reduced time. Like Amazon has shown that companies that focus on customers and resolve customer complaints quickly and efficiently typically have high customer retention and build a good brand. With the help of AI, the solution gives companies a quick way to achieve faster turnaround times.

While we have tested few of the popular models, we could have used Google's word2vec and that could probably have improved most metrics. In addition, the newer modern techniques like BERT was not tested. These could have really enhanced the solution and metrics to real commercial level usage. Despite this, if Customers prefer a simple and easy to understand the solution and can compromise on the recall metrics, the solution could be commercially viable

In terms of what we would do differently, we would perhaps have spent more time on hyper tuning the models to improve performance of the deep learning models. We did manage to tune the Simple RNN using hyper parameter tuning.