AUTOMATIC TICKET ASSIGNMENT – FINAL TECHNICAL REPORT

March 2020

- Mentor
  - Shirish
- Students
  - Akshaya Kumar Das
  - Divya S
  - Dodo Nath
  - Venkata Ramana Reddy Madire
  - Venkatarahavan Nagarajan
  - Pratik Ganesh Futane
  - Rajib Bhattacharya

CONTENTS

1. SUMMARY OF PROBLEM STATEMENT, DATA AND FINDINGS

One of the key activities of any IT function is 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.

Manual assignment of incidents is time consuming and requires human efforts. There may be mistakes due to human errors and resource consumption is carried out ineffectively because of the misaddressing. On the other hand, manual assignment increases the response and resolution times which result in user satisfaction deterioration / poor customer service.

BUSINESS DOMAIN VALUE:

In the support process, incoming incidents are analyzed and assessed by organization's support teams to fulfill 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. Incase L1 / L2 is unable to resolve, they will then escalate / assign the tickets to Functional teams from Applications and Infrastructure (L3 teams). Some portions of incidents are directly assigned to L3 teams by either Monitoring tools or Callers / Requestors. L3 teams will carry out detailed diagnosis and resolve the incidents. Around ~56% of incidents are resolved by Functional / L3 teams. Incase if vendor support is needed, they will reach out for their support towards incident closure.

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 getting assigned to wrong functional groups. Around ~25% of Incidents are wrongly assigned to functional teams. Additional effort needed for Functional teams to re-assign to right functional groups. During this process, some of the incidents are in queue and not addressed timely resulting in poor customer service.

PROBLEM STATEMENT

Incident Management process is expected to **quickly** restore services.

Majority of IT Organizations are yet to Automate Ticket assignment leading to:-

1. Increase in the response and resolution time.
2. Decrease in Customer Satisfaction.
3. Possibility of human error in Ticket Assignment. (~25%)
4. Poor Customer Service.
5. Additional Staffing (~1 FTE) requirement
6. Additional Effort (15 mins for SOP review) requirement (~25-30% of incidents)
7. Decrease in morale of L1 / L2 team.
8. Increase in Expenses.
9. Possibility of missing SLA
10. Possibility of incurring Financial Penalties associated with missed SLAs.

MILESTONES

1. Milestone 1.a: Pre-Processing, Data Visualisation and EDA
    A. Exploring the given Data files
    B. Understanding the structure of data
    C. Missing points in data
    D. Finding inconsistencies in the data
    E. Visualizing different patterns
    F. Visualizing different text features
    G. Dealing with missing values
    H. Text preprocessing
    I. Creating word vocabulary from the corpus of report text data
    J. Creating tokens as required

2. Milestone 1.b: Model Building
    A. Building a model architecture which can classify.

B. Trying different model architectures by researching state of the art for similar tasks.
C. Train the model
D. To deal with large training time, save the weights so that you can use them when training the model for the second time without starting from scratch.

3. Milestone 3: Test the Model, Fine-tuning and Repeat
    A. Test the model and report as per evaluation metrics
    B. Try different models
    C. Try different evaluation metrics
    D. Set different hyper parameters, by trying different optimizers, loss functions, epochs, learning rate, batch size, checkpointing, early stopping etc..for these models to fine-tune them
    E. Report evaluation metrics for these models along with your observation on how changing different hyper parameters leads to change in the final evaluation metric.

DATA SET

Ticket classification will be performed based on the analysis of text within the data available at following location:- https://drive.google.com/file/d/1OZNJm81JXucV3HmZroMq6qCT2m7ez7IJ

File Name:- Input Data Synthetic (created but not used in our project)

Number of Columns:- 4

Number of Rows:- 8500

Number of Rows with garbled Text:- 828

Number of Rows with non-English language:- 824

Following details / Columns of Ticket is available:-

1. Short description
   Summary of the Incident
   Blank rows:- 2
2. Description
   Incident details
   Blank rows:- 1
3. Caller

Name of the person who called to inform the incident

Unique Values:- 2950

4. Assignment group

The ticket queue details, which is our target value.

Unique Values:- 74

Findings

Number of Rows:- 8500

I. **9.74%** of Rows with garbled Text:- 828

II. **9.69%** of Rows with non-English language:- 824

III. **8 null/missing values** present in the Short description column

IV. **1 null/missing values** present in the Description column

V. **Highly imbalanced dataset** in terms of ticket distribution across Assignment Group.

Ratio of GRP_0 to all others is 47:53 and there are 37 groups having less than or equal to 25 tickets assigned each.

- Dealing with imbalanced dataset.
    - Creating distinctive clusters under GRP_0 and downsampling top clusters
    - Clubbing together all those groups into one which has 25 or less tickets assigned
- Replacing TF-IDF vectorizer technique with word embeddings for statistical ML algorithms.

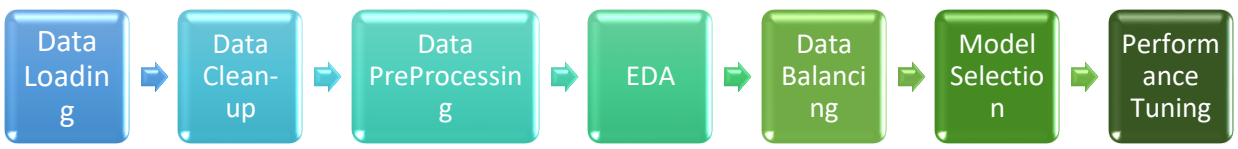
VI. **Shift timing** trend identified as Assignment Group selection criteria

VII. **Number of Lines in Description**, was determining few Assignment Group selection. [Example:- Like Filling a pre-defined Checklist]

2. OVERVIEW OF THE FINAL PROCESS

2.1. DESIGN

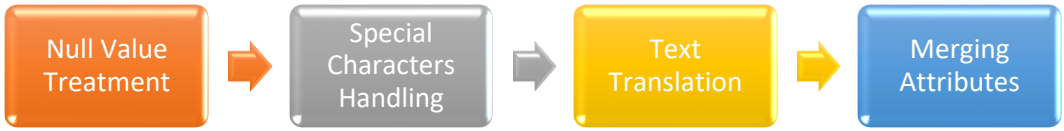
2.1.1. High Level Process flow for the solution is given below :-



- Data Loading – Loading data from the dataset
- Data Clean-up
- Data Preprocessing

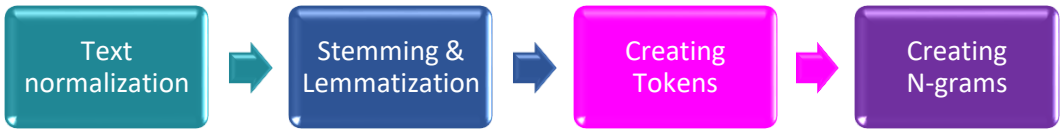
- Exploratory Data Analysis (EDA)
- Data Balancing
- Model Selection
- Performance Tuning

2.1.1.1. Data Clean-up Sub flow:



- Null Value Treatment
- Special Character Handling
- Text Translation
- Merging Attributes

2.1.1.2. Data Preprocessing Sub flow :



- Text normalization
  - converting all letters to lower or upper case
  - converting numbers into words or removing numbers
  - removing punctuations, accent marks and other diacritics
  - removing white spaces
  - removing stop words, sparse terms, and particular words
  - text canonicalization
  - Define regex patterns and function to preprocess text.
- Stemming and Lemmatization (spaCy)
- Creating Tokens
- Creating N-grams

Replacing TF-IDF vectorizer technique with word embeddings for statistical ML algorithms, owing to poor performance during Milestone 1.

2.1.1.3. Data Balancing Sub flow :

Topic Modeling is a technique to extract the hidden topics from large volumes of text. **Latent Dirichlet Allocation(LDA)** is a popular algorithm for topic modeling with excellent implementations in the Python's Gensim package.

- Prepare Stopwords
- Tokenize words and Clean-up text
- Bigram and Trigram Models
- Dictionary and Corpus needed for Topic Modeling
- Building the Topic Model
- Model Perplexity and Coherence Score
- Visualize the topics-keywords
- Topic assignment for GRP_0 tickets
- Down-sampling the majority topics under GRP_0
- Club groups with lesser tickets assigned
- Join and prepare the balanced dataset

2.1.1.4. Model Selection

Following models have been attempted to compare and choose the most optimal model:-

- Multinominal Naive Bayes Classifier
- K Nearest Neighbor
- Support Vector Machine
- Decision Tree
- Random Forest
- Deep Neural Networks
- Convolutional Neural Networks
- Recurrent Neural Networks
- Recurrent Convolutional Neural Networks
- RNN with LSTM

2.2. SALIENT FEATURES OF DATA

- **9.74%** of Rows with garbled Text:- 828
- **9.69%** of Rows with non-English language:- 824
- **8 null/missing values** present in the Short description column
- **1 null/missing values** present in the Description column
- **Highly imbalanced dataset** in terms of ticket distribution across Assignment Group.
    - Ratio of GRP_0 to all others is 47:53 and there are 37 groups having less than or equal to 25 tickets assigned each.
    - Dealing with imbalanced dataset.
        - Creating distinctive clusters under GRP_0 and downsampling top clusters
        - Clubbing together all those groups into one which has 25 or less tickets assigned
    - Replacing TF-IDF vectorizer technique with word embeddings.
- **Shift timing** trend identified as Assignment Group selection criteria
- **Number of Lines in Description**, was determining few Assignment Group selection. [Example:- Like Filling a pre-defined Checklist]

3. STEP-BY-STEP WALK THROUGH THE SOLUTION

3.1. LOADING & EXPLORING THE DATASET

3.1.1. Set the working directory

Mount the drive and set the project path to current working directory, when running in Google Colab.

No changes are required in case of running in Local PC.

3.1.2. Extract Glove Embeddings

Extract Glove 6 Billion word embeddings. We're going to use the 100D and 200d file which has 200 embedding dimensions for each word in the corpus

3.1.3. Load the dataset

We've observed poor performance in the 1st milestone, which enables us to introduce 2 more attributes, such as:

- **Shift**: Working shift of the support associate in which the ticket was received OR failure occurred
- **Lines**: Lines of text present in the ticket description column

Load the serialized dataset stored after 1st milestone's EDA and append the above attributes to them. Also drop *sd_len, sd_word_count, desc_len, desc_word_count* columns.

3.1.4. Check the first and last rows

To ensure all the Rows and Columns are loaded as expected.

3.2. STRUCTURE OF DATASET

3.2.1. Inspect the Dataset

The dataset is divided into two parts, namely, feature matrix and the response vector.

Feature matrix contains all the vectors(rows) of dataset in which each vector consists of the value of dependent features. In above dataset, features are Short description, Description and Caller.

Response vector contains the value of class variable (prediction or output) for each row of feature matrix. In above dataset, the class variable name is Assignment group.

3.2.2. Get the shape and size of the dataset

Number of Columns:- 4

Number of Rows:- 8500

3.2.3. Get more information about the data

i. Name of Columns
ii. Find the data types of each columns
iii. Look for any null/missing values

3.2.4. Describe the dataset with various summary and statistics

```
ticket.describe()
```

	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

3.2.5. Check the Short description of tickets having Description as only 'the'

```
ticket[ticket.Description == 'the'].head()
```

	Short description	Description	Caller	Assignment group
1049	reset passwords for soldfnbq uhnbsvqd using pa...	the	soldfnbq uhnbsvqd	GRP_17
1054	reset passwords for fygrwuna gomcekzi using pa...	the	fygrwuna gomcekzi	GRP_17
1144	reset passwords for wvdxnkhf jirecvta using pa...	the	wvdxnkhf jirecvta	GRP_17
1184	reset passwords for pxvjczdt kizsjfpq using pa...	the	pxvjczdt kizsjfpq	GRP_17
1292	reset passwords for cubdsrml znewqgop using pa...	the	cubdsrml znewqgop	GRP_17

3.2.6. Find out the null value counts in each column

```
ticket.isnull().sum()
```

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

Observations:-

- The dataset comprises of **8500 rows** and **4 columns**
- All columns are of type object containing textual information.
- There are **8 null/missing values** present in the Short description and **1 null/missing values** present in the description column
- **Password reset** is one of the most occurring tickets which reflects in the Short description column.
- The top occurring Description in the dataset is only the text **'the'**, which absolutely doesn't make any sense. hence by looking at the Short description of such rows reveals that these are also a category of Password reset.

3.3. DEALING WITH MISSING VALUES & TREATMENT

3.3.1. NULL treatment

i. Check the rows with null values

ii. NULL/Missing value treatment replaces the NaN cells with just empty string
iii. Verify if the replacement is as expected

Observations:-

- We have various ways of treating the NULL/Missing values in the dataset such as

    - Replacing them with empty string
    - Replacing them with some default values
    - Duplicating the Short description and Description values wherever one of them is Null
    - Dropping the records with null/missing values completely.

- We're not choosing to drop any record as we don't want to loose any information. And as we're going to concatenate the Short description and Description columns for each record while feeding them into NLP, we neither want to pollute the data by introducing any default values nor bias it by duplicating the description colomns.
- Hence our NULL/Missing value treatment replaces the NaN cells with just empty string.

3.3.2. Mojibake

Mojibake is the garbled text that is the result of text being decoded using an unintended character encoding. The result is a systematic replacement of symbols with completely unrelated ones, often from a different writing system.
This display may include the generic replacement character ("**?**") in places where the binary representation is considered invalid. A replacement can also involve multiple consecutive symbols, as viewed in one encoding, when the same binary code constitutes one symbol in the other encoding. This is either because of differing constant length encoding (as in Asian 16-bit encodings vs European 8-bit encodings), or the use of variable length encodings (notably UTF-8 and UTF-16). Few such Mojibakes are **¶**, **ç**, **å**, **€**, **æ**, **œ**, **º**, **‡**, **¼**, **¥** etc.

As we're dealing with Natural Language and the source of the data is unknown to us, let's run the encoding check to figure out if the dataset is Mojibake impacted.

The library **ftfy** (Fixes Text For You) has a greater ability to detect, fix and deal with such Mojibakes. It fixes Unicode that's broken in various ways. The goal of ftfy is to take in bad Unicode and output good Unicode.

Installation:
using pypi: **!pip install ftfy**
using conda: **conda install -c conda-forge ftfy**

i. Function applied to detect Mojibakes in the dataset

ii. 828 rows contain Garbled text.
iii. Check the list of Mojobakes in ftfy library, and test a sample fix
iv. Since the fix is successful, applying the fix to rest of the data.
v. Row 8471, which initially contained Mojibake is verified.
vi. Serialize the mojibake treated dataset.

Observations:

- badness.sequence_weirdness() determines how often a text has unexpected characters or sequences of characters. This metric is used to disambiguate when text should be re-decoded or left as is.
- We're successfully able to get the garbled characters back into their original form using **ftfy.fix_text()``**, however it is observed that the row# 8471 is not English but Mandarin.
- So the data in our hand is multilingual and it is quite not possible to derive embeddings for mix of multiple languages. We're going to translate the entire dataset into a single language of English.

3.3.3. Language Translation (Goslate: Free Google Translate API)

Goslate is an open source python library that implemented Google Translate API. This uses the Google Translate Ajax API to make calls to such methods as detect and translate. It is choosen over another library Googletrans from Google as Goslate is developed to bypass the ticketing mechanism to prevent simple crawler program to access the Ajax API. Hence Goslate with multiple service urls is able to translate the entire dataset in very few iterations without blocking the user's IP address.

Installation:
using pypi: **!pip install goslate**
using conda: **conda install -c conda-forge goslate**

Servicce Urls
used: **translate.google.com**, **translate.google.com.au**, **translate.google.com.ar**, **translate.google.co.kr**, **translate.google.co.in**, **translate.google.co.jp**, **translate.google.at**, **translate.google.de**, **translate.google.ru**, **translate.google.ch**, **translate.google.fr**, **translate.google.es**, **translate.google.ae**

i. Define and construct the service urls.
ii. Test Translation on row# 8471 of Short description.
iii. Add Column Language to capture the language present in 'Short description' & 'Description'

List of Languages detected:-

```
{'English': 7676,
 'Maltese': 14,
 'Chinese (Simplified)': 122,
 'German': 550,
 'Luxembourgish': 8,
 'Afrikaans': 14,
 'Spanish': 10,
 'Welsh': 1,
 'Portuguese': 32,
 'Italian': 6,
 'Bengali': 1,
 'Catalan': 1,
 'French': 12,
 'Shona': 1,
 'Malagasy': 1,
 'Icelandic': 1,
 'Dutch': 4,
 'Norwegian': 4,
 'Romanian': 5,
 'Vietnamese': 2,
 'Gujarati': 2,
 'Lithuanian': 1,
 'Cebuano': 3,
 'Polish': 6,
 'Slovenian': 1,
 'Hindi': 2,
 'Latin': 1,
 'Croatian': 1,
 'Filipino': 3,
 'Chichewa': 1,
 'Hungarian': 2,
 'Samoan': 1,
 'Danish': 2,
 'Frisian': 1,
 'Swedish': 1,
 'Galician': 1,
 'Turkish': 1,
 'Kurdish (Kurmanji)': 1,
 'Zulu': 1,
 'Greek': 3})
```

Observation:-

824 rows contain non-English language.

iv. Serialize the translated dataset.

v. Load the translated pickle file incase the IP gets blocked.

Observations:-

- Unless paid service is used, Google blocks repetitive hits to its Ajax API either via Googletrans or Goslate after certain iterations by clogging the IP address.
- Using these list of various domains of translation API as service urls helped the traffic being patched among themselves, in turn allowing a longer buffer before the IP gets blocked.

3.4. TEXT PRE-PROCESSING

Text preprocessing is the process of transferring text from human language to machine-readable format for further processing. After a text is obtained, we start with text normalization. **Text normalization** includes:

i. converting all letters to lower or upper case
ii. converting numbers into words or removing numbers
iii. removing punctuations, accent marks and other diacritics
iv. removing white spaces
v. removing stop words, sparse terms, and particular words
vi. text canonicalization

Define regex patterns and function to preprocess text.

i. Test preprocessing on Description for row# 32.
ii. Once successful, apply preprocessing to entire dataset.

Observations:-

- Entire dataset is converted into lower case
- Users email addresses will add NO value to our analysis, despite the fact that user id is given in the caller column. So all email adresses are removed from the dataset
- All numerals are removed because they were dominating the dataset if we were converting them into their word representation otherwise.
- All punctuation marks are removed which used to be a hindrance in lemmatization.
- All occurances of more than one blank spaces, horizontal tab spaces, new line breaks etc. have been replaced with single blank space.

Now with a nice and cleaner data in our hand let's proceed towards Lemmatization.

3.5. BUILDING WORD VOCABULARY

3.5.1. Stemming and Lemmatization

Stemming and Lemmatization are Text Normalization (or sometimes called Word Normalization) techniques in the field of Natural Language Processing that are used to prepare text, words, and documents for further processing.
In grammar, inflection is known as the modification of a word to express different grammatical categories such as tense, case, voice, aspect, person, number, gender, and mood. An inflection expresses one or more grammatical categories with a prefix, suffix or infix, or another internal modification such as a vowel change.

Stemming
Stemming is the process of reducing inflection in words to their root forms such as mapping a group of words to the same stem even if the stem itself is not a valid word in the Language.

Lemmatization
Lemmatization, unlike Stemming, reduces the inflected words properly ensuring that the root word belongs to the language. In Lemmatization root word is called Lemma. A lemma (plural lemmas or lemmata) is the canonical form, dictionary form, or citation form of a set of words.

3.5.2. spaCy

The spaCy library is one of the most popular NLP libraries along with NLTK which contains only one, but the best algorithm to solve any Natural Language problem. Once it is downloaded and installed, the next step is to download the language model, which is used to perform a variety of NLP tasks.

Installation:
using pypi: **!pip install spacy**
using conda: **conda install -c conda-forge spacy**

Language Model Download:
$ python -m spacy download en_core_web_md

i. Initialize spacy 'en' medium model, keeping only tagger component needed for lemmatization.
ii. Define a function to lemmatize the descriptions.
iii. Test lemmatization on Description in row# 43.
iv. Once successful, apply Lemmatization to entire dataset and verify.
v. Serialize the preprocessed dataset.
vi. Create new features of length and word count for both of the description columns.

3.6. CREATING TOKENS

Ticket Description and Short description is tokenized into words and n-grams

3.7. EXPLORATORY DATA ANALYSIS

Exploratory Data Analysis (EDA) is an approach/philosophy for data analysis that employs a variety of techniques (mostly graphical) to

- maximize insight into a data set;
- uncover underlying structure;
- extract important variables;

- detect outliers and anomalies;
- test underlying assumptions;
- develop parsimonious models; and
- determine optimal factor settings

Visually representing the content of a text document is one of the most important tasks in the field of text mining. It helps not only to explore the content of documents from different aspects and at different levels of details, but also helps in summarizing a single document, show the words and topics, detect events, and create storylines.

We'll be using plotly library to generate the graphs and visualizations. We need cufflinks to link plotly to pandas dataframe and add the iplot method

Installation:
using pypi: **!pip install plotly cufflinks**
using conda: **conda install -c conda-forge plotly cufflinks**

Check the version of Plotly and Cufflinks packages:-

```
print('Plotly:', py.__version__)
print('Cufflinks:', cf.__version__)
```

```
Plotly: 4.5.0
Cufflinks: 0.17.0
```

EDA is covered further under Visualizations section…

3.8. DEALING WITH IMBALANCED DATASET [MILESTONE 2]

We've observed poor performance in the 1st milestone, which enables us to introduce 2 more attributes, such as:

- **Shift**: Working shift of the support associate in which the ticket was recieved OR failure occured
- **Lines**: Lines of text present in the ticket description column

Load the searialized dataset stored after 1st milestone's EDA and append the above attributes to them. Also drop *sd_len, sd_word_count, desc_len, desc_word_count* columns

Observation From Milestone-1

Out of all the models we've tried in Milestone-1, Support Vector Machine (SVM) under statistical ML algorithms and Neural Networks are performing better than all others. The models were highly overfitted and one of the obvious reason was the dataset was highly imbalanced. Ratio of GRP_0 to all others is 47:53 and there are 40 groups having less than or equal to 30 tickets assigned each.

Let's address this problem to fine tune the model accuracy by implementing

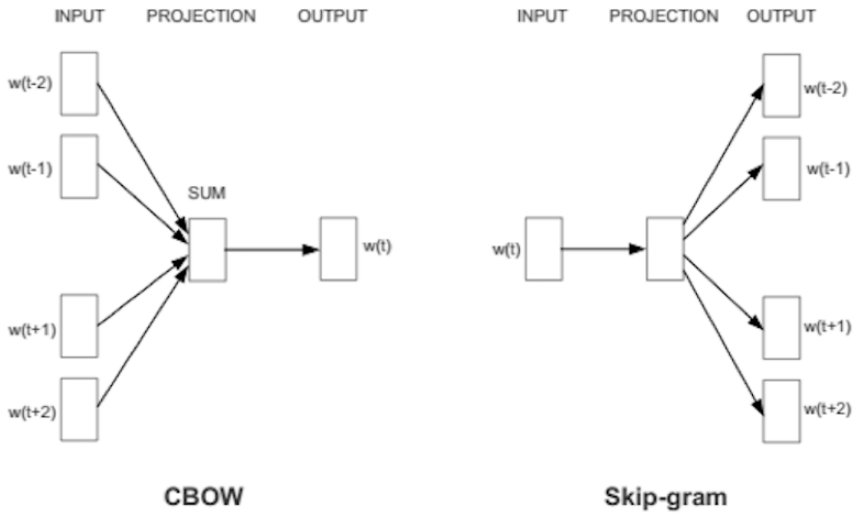
- Dealing with imbalanced dataset.
  - Creating distinctive clusters under GRP_0 and downsampling top clusters
  - Clubbing together all those groups into one which has 30 or less tickets assigned
- Replacing TF-IDF vectorizer technique with word embeddings for statistical ML algorithms.

Create Word Embeddings

We've observed poor performance in the 1st milestone, which enables us to create our own word embeddings. Let's load the preprocessed dataset and use Gensim model to create Word2Vec embeddings.

Word embedding is one of the most important techniques in natural language processing(NLP), where words are mapped to vectors of real numbers. Word embedding is capable of capturing the meaning of a word in a document, semantic and syntactic similarity, relation with other words.

The word2vec algorithms include skip-gram and CBOW models, using either hierarchical softmax or negative sampling.



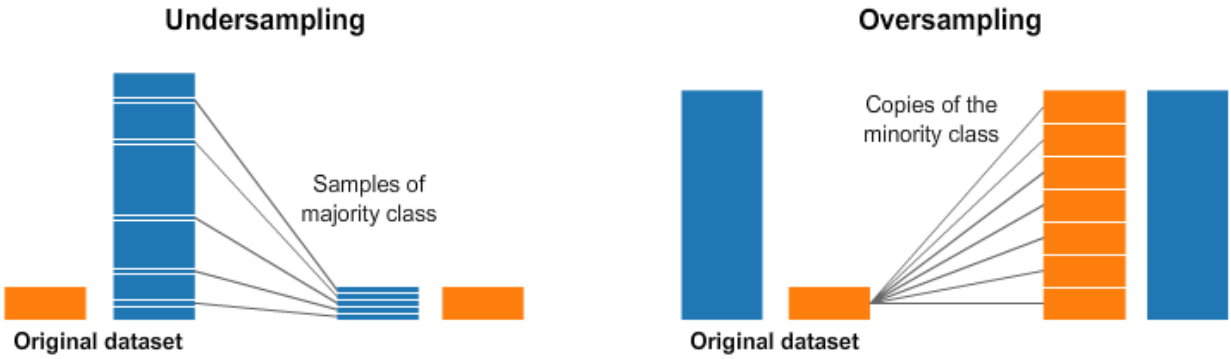
I. Create a Function to create the tokenized sentence
II. Create the Bigram and Trigram models
III. Serialize bigram and trigram for future
IV. Create the tagged documents

V. Build the Word2Vec model

Comments: Word Embbeddings are generated from the corpus of our tickets dataset and serialized for further use.

Resampling The Imbalanced Dataset

A widely adopted technique for dealing with highly unbalanced datasets is called resampling. It consists of removing samples from the majority class (under-sampling) and / or adding more examples from the minority class (over-sampling).



Topic Modeling

Topic Modeling is a technique to extract the hidden topics from large volumes of text. **Latent Dirichlet Allocation(LDA)** is a popular algorithm for topic modeling with excellent implementations in the Python's Gensim package.

Let's first use gensim to implement LDA and find out any distinctive topics among GRP_0, followed by down-sampling the top 3 topics to contain maximum number of tickets created for.

Installation:
using pypi: **!pip install gensim**
using conda: **conda install -c conda-forge gensim**

1. Prepare Stopwords

Used English stopwords from NLTK and extended it to include domain specific frequent words

2. Tokenize words and Clean-up text

Tokenize each sentence into a list of words, removing punctuations and unnecessary characters altogether.

3. Bigram and Trigram Models

Bigrams and Trigrams are two and three words frequently occurring together respectively in a document.

4. Dictionary and Corpus needed for Topic Modeling

Create the two main inputs to the LDA topic model are the dictionary(id2word) and the corpus.

5. Building the Topic Model

Build a Topic Model with top 3 different topics where each topic is a combination of keywords and each keyword contributes a certain weightage to the topic.

```
Topic: 1
Words: 0.050*"unable" + 0.034*"outlook" + 0.031*"issue" + 0.023*"work" + 0.020*"error" + 0.019*"connect" + 0.018*"open" + 0.016
*"get" + 0.016*"skype" + 0.011*"vpn"

Topic: 2
Words: 0.098*"password" + 0.066*"erp" + 0.061*"reset" + 0.054*"account" + 0.048*"user" + 0.037*"login" + 0.037*"sid" + 0.034*"l
ock" + 0.017*"unlock" + 0.016*"request"

Topic: 3
Words: 0.043*"gmail" + 0.042*"com" + 0.026*"access" + 0.020*"company" + 0.020*"update" + 0.015*"email" + 0.013*"ticket" + 0.011
*"device" + 0.009*"mobile" + 0.008*"send"
```

How to interpret this?

Topic 1 is a represented as `0.060*"company" + 0.028*"windows" + 0.026*"device" + 0.021*"vpn" + 0.021*"connect" + 0.018*"message" + 0.014*"link" + 0.013*"window" + 0.011*"follow" + 0.011*"use"`

It means the top 10 keywords that contribute to this topic are: 'company', 'windows', 'device'.. and so on and the weight of 'windows' on topic 1 is 0.028.

The weights reflect how important a keyword is to that topic.

6. Model Perplexity and Coherence Score

Model perplexity and topic coherence provide a convenient measure to judge how good a given topic model is.

Perplexity is a measure of how good the model is. Lower the better.

```
Perplexity:  -6.800106460957963

Coherence Score:  0.4568991626530355
```

7. Visualize the topics-keywords

Examine the produced topics and the associated keywords using pyLDAvis.

8. Topic assignment for GRP_0 tickets

Run LDA for each record of GRP_0 to find the associated topic based on the LDA score. As the topic modeling has been trained to accommodate only top 3 topics for entire GRP_0 data, any record scoring less than 50%, we categorize them into 4th(other) topic and such tickets are not the candidates for resampling.

```
Text: outlook receive from hmjdrvpb.komuaywn@gmail.com hello team meetingsskype meeting etc be not appear in outlook calendar c
an somebody please advise how to correct this kind
Topic: Communication Issue
Score: 0.5099999904632568
```

	Short description	Description	Language	Summary	Shift	Lines	Topic	Assignment group
0	login issue	verified user detailsemployee manager name che...	English	login issue verified user detailsemployee mana...	General	5	Account/Password Reset	GRP_0
1	outlook	receive from hmjdrvpb.komuaywn@gmail.com hello...	English	outlook receive from hmjdrvpb.komuaywn@gmail.c...	General	7	Communication Issue	GRP_0
2	can not log in to vpn	receive from eylqgodm.ybqkwiam@gmail.com hi i ...	English	can not log in to vpn receive from eylqgodm.yb...	General	7	Access Issue	GRP_0
3	unable to access hrtool page	unable to access hrtool page	English	unable to access hrtool page unable to access ...	General	1	Access Issue	GRP_0
4	error skype	error skype	English	error skype error skype	General	1	Communication Issue	GRP_0

Count of records, based on Topics:-

```
Account/Password Reset    1308
Communication Issue       1304
Access Issue              1022
Other Issues               342
```

Observations:

- From the above analysis, it's evident that the tope 3 topics are present in maximum numbers. The ratio of top 3 topics and other topic is 33:33:26:8
- Except for the Other Issues, rest 3 categories of records can be down sampled to balance the dataset

9. Down-sampling the majority topics under GRP_0

Under-sample the majority class(es) by randomly picking samples with or without replacement. We're using RandomUnderSampler class from imblearn.

```
Other Issues              342
Account/Password Reset    342
Access Issue              342
Communication Issue       342
```

Observation:
The output of the UnderSampling technique shows that all the 4 distinct topics are resampled to exactly match the records in each topic making them a perfectly balanced distribution under GRP_0.

Let's combine the Topic and Assignment group columns to maintain a single target attribute.

10. Club groups with lesser tickets assigned

Combine all groups with less than 25 tickets assigned into one separate group named Miscellaneous

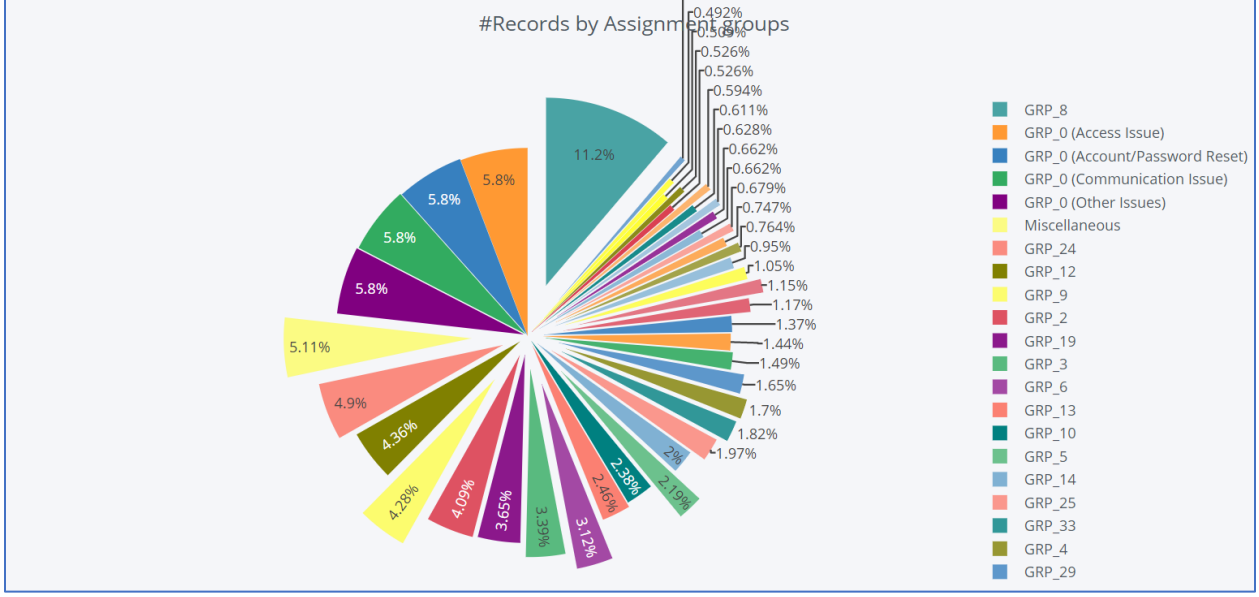
#Groups with less than equal to 25 tickets assigned: **37**

11. Join and prepare the balanced dataset

Let's club together resampled topics under GRP_0 with Miscellaneous group with less than 25 tickets with all others

#Unique groups remaining: **41**

	labels	pull	values	D	E	F	G	H	I	J	K	L	M	N	O
1	GRP_0 (Access Issue)	0	342												
2	GRP_0 (Account/Password Reset)	0.0075	342												
3	GRP_0 (Communication Issue)	0.015	342												
4	GRP_0 (Other Issues)	0.0225	342												
5	GRP_1	0.03	31												



4. MODEL EVALUATION

Since the data provided to us a few major issues like:
1. Huge imbalance of Incident distribution among Assignment Groups.
2. 37 Assignment groups were having Incident count less than or equal to 25.
3. Less variety of Incident per group
4. A few Incidents in different languages.
5. Lack of dedicated finance domain corpus

We are able to address the concerns by employing techniques like:
1. LDA - topic modelling to sub group the Group_0 which had 47% of the Incidents
2. Merging limited groups which do not have enough instance counts into single group "Miscellaneous"
3. We employed Random Oversampling to scale up instances for other groups to make data balance
4. We did try to utilize GloVe embedding but could not get satisfactory accuracy on NN
5. Identification of crucial details like "Lines"- a technical parameter and "Shift" details.

Initial analysis gave us insights that SVM worked the best with ~68% accuracy. However, in spite of the advantage of SVM it has some limitations.
i] It is applicable to only binary classification. If a multiple classification problem is given, it should be decomposed into several binary classification problems using SVM.
ii] Problem in representing documents into numerical vectors, sparse distribution, since inner products of its input vector and training examples generates zero values very frequently.

With those limitation, we embarked on Neural Network model, LSTM, with our own "word embeddings" created of the incident description.

We applied self-generated, word embeddings with SVM. But, for the limitations highlighted above, the model could not give us satisfactory accuracy.

Finally, the generated word embeddings were used as pre-calculated weights for LSTM model and we were able to get satisfactory Accuracy of 94% for RNN with LSTM.

MODEL ARCHITECTURE

Build Multi-Class classifier that can classify the tickets by analyzing text. Text present in Short description and Description Columns are concatenated prior to Model Building.

Following models have been attempted to compare and choose the most optimal model in Milestone 1.

Multinominal Naive Bayes Classifier
K Nearest Neighbor
Support Vector Machine
Decision Tree
Random Forest
Deep Neural Networks
Convolutional Neural Networks
Recurrent Neural Networks
Recurrent Convolutional Neural Networks
RNN with LSTM

I. Import Model, Packages, Evaluation Metrics, Visualization tools, Utilities
II. Create Generic Class and Methods to evaluate Model Performance
III. Split Data into Training and Test Set (Training : 80% of Data, Testing : 20% of Data)

MODEL BUILDING

Multinominal Naive Bayes Classifier

Naive Bayes is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set. There is not a single algorithm for training such classifiers, but a family of algorithms based on a common principle: all naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable.
Advantages:

- It works very well with text data
- Easy to implement
- Fast in comparing to other algorithms

Disadvantages:

- A strong assumption about the shape of the data distribution
- Limited by data scarcity for which any possible value in feature space, a likelihood value must be estimated by a frequentist

```
==================================
Training accuracy: 56.26%
Testing accuracy: 52.47%
==================================
Confusion matrix:
 [[761   0   0 ...   0   0   0]
 [  3   0   0 ...   0   2   0]
 [ 15   0   0 ...   0   9   0]
 ...
 [  1   0   0 ...   0   0   0]
 [ 15   0   0 ...   0 106   0]
 [ 18   0   0 ...   0  38   0]]
==================================
```

Classification report:

	precision	recall	f1-score	support
0	0.53	1.00	0.69	761
1	0.00	0.00	0.00	8
2	0.00	0.00	0.00	24
3	0.00	0.00	0.00	5
4	0.64	0.21	0.32	42
5	0.00	0.00	0.00	26
6	0.00	0.00	0.00	20
7	0.00	0.00	0.00	8
8	0.00	0.00	0.00	20
9	0.00	0.00	0.00	17
10	0.00	0.00	0.00	18
11	0.00	0.00	0.00	58
12	0.67	0.04	0.07	51
13	0.00	0.00	0.00	5
14	0.00	0.00	0.00	7
15	0.00	0.00	0.00	5
16	0.00	0.00	0.00	3
17	1.00	0.19	0.33	72
18	0.00	0.00	0.00	20
19	0.00	0.00	0.00	13
20	0.00	0.00	0.00	2
21	0.00	0.00	0.00	8
22	0.00	0.00	0.00	20
23	0.00	0.00	0.00	42
24	0.00	0.00	0.00	6
25	0.00	0.00	0.00	24
27	0.00	0.00	0.00	14
28	0.00	0.00	0.00	12
29	0.00	0.00	0.00	1
30	0.00	0.00	0.00	2
31	0.00	0.00	0.00	2
32	0.00	0.00	0.00	2
33	0.00	0.00	0.00	6
34	0.00	0.00	0.00	26
35	0.00	0.00	0.00	8
36	0.00	0.00	0.00	10
37	0.00	0.00	0.00	8
39	0.00	0.00	0.00	1
40	0.00	0.00	0.00	12
41	0.00	0.00	0.00	2
42	0.00	0.00	0.00	9
43	0.00	0.00	0.00	6
45	0.00	0.00	0.00	28
46	0.00	0.00	0.00	2
47	0.00	0.00	0.00	1
48	0.00	0.00	0.00	1
49	0.00	0.00	0.00	3
51	0.00	0.00	0.00	1
56	0.00	0.00	0.00	46
57	0.00	0.00	0.00	5
59	0.00	0.00	0.00	5
60	0.00	0.00	0.00	1
62	0.00	0.00	0.00	2
63	0.00	0.00	0.00	1
64	0.00	0.00	0.00	1
66	0.00	0.00	0.00	1
67	0.00	0.00	0.00	18
70	0.00	0.00	0.00	1
72	0.47	0.88	0.61	121
73	0.00	0.00	0.00	56
accuracy			0.52	1700
macro avg	0.06	0.04	0.03	1700
weighted avg	0.35	0.52	0.38	1700

Time taken: 0.446161s to run the model

K Nearest Neighbor

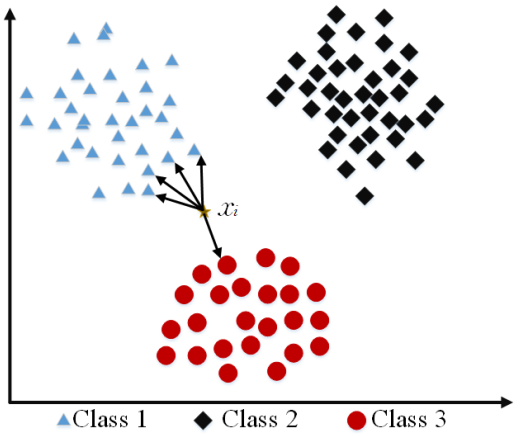
In pattern recognition, the k-nearest neighbours algorithm (k-NN) is a non-parametric method used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space. The output depends on whether k-NN is used for classification or regression:

In k-NN classification, the output is a class membership. An object is classified by a plurality vote of its neighbours, with the object being assigned to the class most common among its k nearest neighbours (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbour. In k-NN regression, the output is the property value for the object. This value is the average of the values of k nearest neighbours. k-NN is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until classification.

Both for classification and regression, a useful technique can be to assign weights to the contributions of the neighbours, so that the nearer neighbours contribute more to the average than the more distant ones. For example, a common weighting scheme consists in giving each neighbour a weight of 1/d, where d is the distance to the neighbour.

The neighbours are taken from a set of objects for which the class (for k-NN classification) or the object property value (for k-NN regression) is known. This can be thought of as the training set for the algorithm, though no explicit training step is required.

A peculiarity of the k-NN algorithm is that it is sensitive to the local structure of the data.




```
================================================================================
Training accuracy: 69.81%
Testing accuracy: 61.24%
================================================================================
Confusion matrix:
 [[746   0   0 ...   0   0   1]
 [  1   2   0 ...   0   0   0]
 [  9   0  10 ...   0   0   1]
 ...
 [  1   0   0 ...   0   0   0]
 [  4   1   6 ...   0  91   7]
 [ 15   0   0 ...   0  29  10]]
================================================================================
```

```
==========================================================
Classification report:
              precision    recall  f1-score   support

           0       0.62      0.98      0.76       761
           1       0.50      0.25      0.33         8
           2       0.48      0.42      0.44        24
           3       1.00      0.40      0.57         5
           4       0.47      0.38      0.42        42
           5       0.57      0.31      0.40        26
           6       0.45      0.25      0.32        20
           7       0.00      0.00      0.00         8
           8       0.00      0.00      0.00        20
           9       1.00      0.12      0.21        17
          10       0.42      0.28      0.33        18
          11       0.53      0.14      0.22        58
          12       0.69      0.39      0.50        51
          13       0.00      0.00      0.00         5
          14       0.00      0.00      0.00         7
          15       0.00      0.00      0.00         5
          16       0.33      0.33      0.33         3
          17       0.94      0.65      0.77        72
          18       0.75      0.30      0.43        20
          19       1.00      0.08      0.14        13
          20       0.00      0.00      0.00         2
          21       0.00      0.00      0.00         8
          22       0.40      0.10      0.16        20
          23       0.64      0.17      0.26        42
          24       0.00      0.00      0.00         6
          25       0.40      0.08      0.14        24
          27       0.50      0.07      0.12        14
          28       1.00      0.08      0.15        12
          29       0.00      0.00      0.00         1
          30       1.00      0.50      0.67         2
          31       0.00      0.00      0.00         2
          32       0.00      0.00      0.00         2
          33       0.33      0.17      0.22         6
          34       0.00      0.00      0.00        26
          35       0.50      0.12      0.20         8
          36       1.00      0.20      0.33        10
          37       0.67      0.25      0.36         8
          39       0.00      0.00      0.00         1
          40       0.00      0.00      0.00        12
          41       0.00      0.00      0.00         2
          42       0.00      0.00      0.00         9
          43       1.00      0.17      0.29         6
          45       0.80      0.43      0.56        28
          46       0.00      0.00      0.00         2
          47       0.00      0.00      0.00         1
          48       0.00      0.00      0.00         1
          49       0.00      0.00      0.00         3
          51       0.00      0.00      0.00         1
          54       0.00      0.00      0.00         0
          56       0.68      0.57      0.62        46
          57       0.00      0.00      0.00         5
          59       0.00      0.00      0.00         5
          60       0.00      0.00      0.00         1
          62       0.00      0.00      0.00         2
          63       0.00      0.00      0.00         1
          64       0.00      0.00      0.00         1
          66       0.00      0.00      0.00         1
          67       0.40      0.11      0.17        18
          70       0.00      0.00      0.00         1
          72       0.59      0.75      0.66       121
          73       0.48      0.18      0.26        56

    accuracy                           0.61      1700
   macro avg       0.33      0.15      0.19      1700
weighted avg       0.57      0.61      0.54      1700

Time taken: 2.626797s to run the model
```

Support Vector Machine

In machine learning, support-vector machines (SVMs, also support-vector networks) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier (although methods such as Platt scaling exist to use SVM in a probabilistic classification setting). An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on the side of the gap on which they fall.

In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.

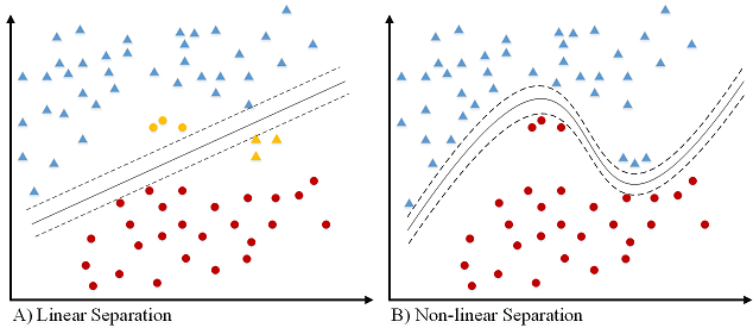
When data are unlabelled, supervised learning is not possible, and an unsupervised learning approach is required, which attempts to find natural clustering of the data to groups, and then map new data to these formed groups. The support-vector clustering algorithm, created by Hava Siegelmann and Vladimir Vapnik, applies the statistics of support vectors, developed in the support vector machines algorithm, to categorize unlabeled data, and is one of the most widely used clustering algorithms in industrial applications.

The advantages of support vector machines are based on scikit-learn page:

- Effective in high dimensional spaces.
- Still effective in cases where number of dimensions is greater than the number of samples
- Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
- Versatile: different Kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels.

The disadvantages of support vector machines include:

- If the number of features is much greater than the number of samples, avoiding over-fitting via choosing kernel functions and regularization term is crucial.
- SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation (see Scores and probabilities, below).




```
==================================
Training accuracy: 92.46%
Testing accuracy: 68.16%
==================================
Confusion matrix:
 [[522   0   0 ...   5   1   2]
 [  0   2   0 ...   0   0   0]
 [  3   0  10 ...   0   1   1]
 ...
 [  8   0   0 ...   7   0   0]
 [  1   1   2 ...   0  81   0]
 [  4   0   0 ...   0  26   8]]
==================================
```

```
==========================================================
Classification report:
              precision    recall  f1-score   support

           0       0.74      0.93      0.82       761
           1       0.00      0.00      0.00         8
           2       0.60      0.62      0.61        24
           3       0.40      0.40      0.40         5
           4       0.52      0.60      0.56        42
           5       0.54      0.58      0.56        26
           6       0.42      0.40      0.41        20
           7       1.00      0.38      0.55         8
           8       0.25      0.10      0.14        20
           9       0.88      0.88      0.88        17
          10       0.60      0.33      0.43        18
          11       0.54      0.24      0.33        58
          12       0.59      0.45      0.51        51
          13       1.00      0.20      0.33         5
          14       0.67      0.29      0.40         7
          15       0.33      0.20      0.25         5
          16       0.43      1.00      0.60         3
          17       0.90      0.89      0.90        72
          18       0.79      0.55      0.65        20
          19       1.00      0.08      0.14        13
          20       0.00      0.00      0.00         2
          21       0.50      0.12      0.20         8
          22       0.58      0.55      0.56        20
          23       0.33      0.29      0.31        42
          24       0.33      0.17      0.22         6
          25       0.44      0.17      0.24        24
          27       0.42      0.36      0.38        14
          28       0.75      0.25      0.38        12
          29       0.00      0.00      0.00         1
          30       0.67      1.00      0.80         2
          31       0.00      0.00      0.00         2
          32       1.00      0.50      0.67         2
          33       0.67      0.33      0.44         6
          34       0.25      0.08      0.12        26
          35       0.40      0.25      0.31         8
          36       0.75      0.60      0.67        10
          37       0.67      0.50      0.57         8
          39       0.00      0.00      0.00         1
          40       1.00      0.08      0.15        12
          41       0.00      0.00      0.00         2
          42       1.00      0.11      0.20         9
          43       0.50      0.17      0.25         6
          45       0.76      0.46      0.58        28
          46       0.00      0.00      0.00         2
          47       1.00      1.00      1.00         1
          48       0.00      0.00      0.00         1
          49       0.00      0.00      0.00         3
          51       1.00      1.00      1.00         1
          56       0.79      0.59      0.68        46
          57       1.00      0.20      0.33         5
          59       0.00      0.00      0.00         5
          60       0.00      0.00      0.00         1
          62       0.00      0.00      0.00         2
          63       0.00      0.00      0.00         1
          64       0.00      0.00      0.00         1
          66       0.00      0.00      0.00         1
          67       0.50      0.39      0.44        18
          70       0.00      0.00      0.00         1
          72       0.56      0.88      0.69       121
          73       0.50      0.18      0.26        56

    accuracy                           0.67      1700
   macro avg       0.48      0.32      0.35      1700
weighted avg       0.65      0.67      0.63      1700

Time taken: 1.074287s to run the model
```

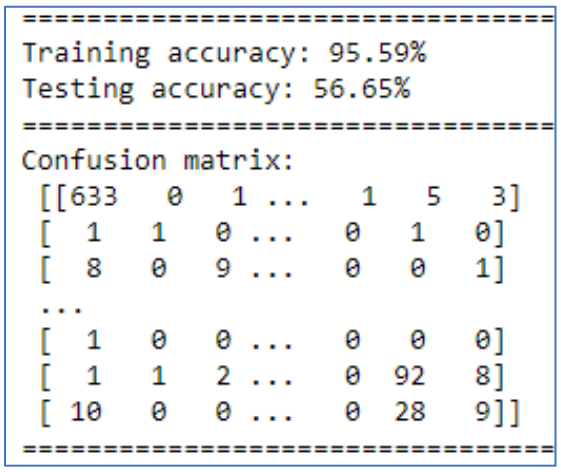
```
==================================
Training accuracy: 81.91%
Testing accuracy: 62.12%
==================================
Confusion matrix:
 [[751   0   0 ...   0   0   0]
 [  2   0   0 ...   0   1   0]
 [ 14   0   9 ...   0   1   0]
 ...
 [  1   0   0 ...   0   0   0]
 [  4   0   2 ...   0 108   1]
 [ 16   0   0 ...   0  35   3]]
==================================
```

```
============================================================
Classification report:
              precision    recall  f1-score   support

           0       0.60      0.99      0.75       761
           1       0.00      0.00      0.00         8
           2       0.82      0.38      0.51        24
           3       0.00      0.00      0.00         5
           4       0.49      0.43      0.46        42
           5       0.60      0.23      0.33        26
           6       1.00      0.25      0.40        20
           7       0.00      0.00      0.00         8
           8       0.00      0.00      0.00        20
           9       0.89      1.00      0.94        17
          10       0.80      0.22      0.35        18
          11       0.50      0.03      0.06        58
          12       0.74      0.39      0.51        51
          13       0.00      0.00      0.00         5
          14       1.00      0.14      0.25         7
          15       0.00      0.00      0.00         5
          16       0.33      0.33      0.33         3
          17       0.92      0.76      0.83        72
          18       0.86      0.30      0.44        20
          19       0.00      0.00      0.00        13
          20       0.00      0.00      0.00         2
          21       0.00      0.00      0.00         8
          22       0.33      0.10      0.15        20
          23       0.67      0.05      0.09        42
          24       0.00      0.00      0.00         6
          25       0.00      0.00      0.00        24
          27       1.00      0.14      0.25        14
          28       0.00      0.00      0.00        12
          29       0.00      0.00      0.00         1
          30       1.00      0.50      0.67         2
          31       0.00      0.00      0.00         2
          32       0.00      0.00      0.00         2
          33       1.00      0.17      0.29         6
          34       0.50      0.04      0.07        26
          35       0.00      0.00      0.00         8
          36       1.00      0.20      0.33        10
          37       0.00      0.00      0.00         8
          39       0.00      0.00      0.00         1
          40       0.00      0.00      0.00        12
          41       0.00      0.00      0.00         2
          42       0.00      0.00      0.00         9
          43       0.00      0.00      0.00         6
          45       0.76      0.46      0.58        28
          46       0.00      0.00      0.00         2
          47       0.00      0.00      0.00         1
          48       0.00      0.00      0.00         1
          49       0.00      0.00      0.00         3
          51       0.00      0.00      0.00         1
          56       0.85      0.50      0.63        46
          57       0.00      0.00      0.00         5
          59       0.00      0.00      0.00         5
          60       0.00      0.00      0.00         1
          62       0.00      0.00      0.00         2
          63       0.00      0.00      0.00         1
          64       0.00      0.00      0.00         1
          66       0.00      0.00      0.00         1
          67       0.50      0.17      0.25        18
          70       0.00      0.00      0.00         1
          72       0.58      0.89      0.70       121
          73       0.60      0.05      0.10        56

    accuracy                           0.62      1700
   macro avg       0.31      0.15      0.17      1700
weighted avg       0.57      0.62      0.53      1700

Time taken: 46.921049s to run the model
```

Decision Tree

Decision tree classifiers are utilized as a well known classification technique in different pattern recognition issues, for example, image classification and character recognition (Safavian & Landgrebe, 1991). Decision tree classifiers perform more successfully, specifically for complex classification problems, due to their high adaptability and computationally effective features. Besides, decision tree classifiers exceed expectations over numerous typical supervised classification methods (Friedl & Brodley, 1997).

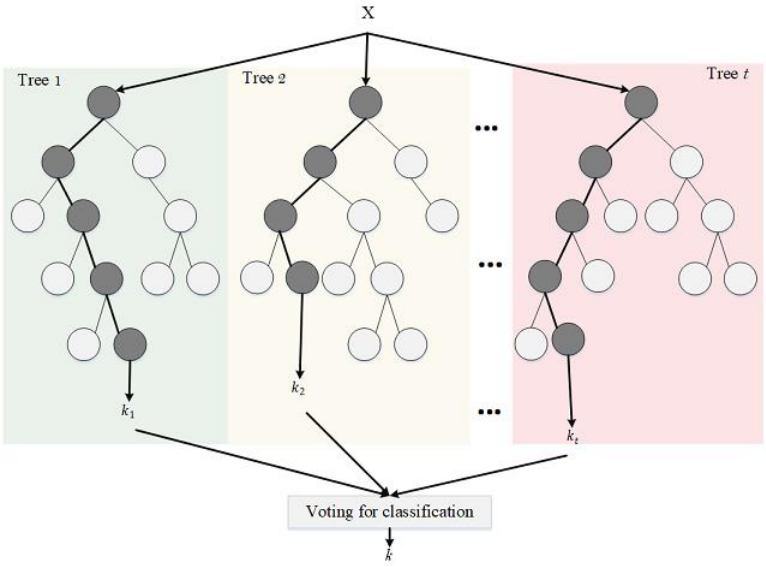
In particular, no distribution assumption is needed by decision tree classifiers regarding the input data. This particular feature gives to the Decision Tree Classifiers a higher adaptability to deal with different datasets, whether numeric or categorical, even with missing data. Also, decision tree classifiers are basically nonparametric. Also, decision trees are ideal for dealing with nonlinear relations among features and classes. At long last, the classification procedure through a tree-like structure is constantly natural and interpretable.



```
------------------------------------------------------------
Classification report:
              precision    recall  f1-score   support

           0       0.72      0.83      0.77       761
           1       0.20      0.12      0.15         8
           2       0.39      0.38      0.38        24
           3       0.14      0.20      0.17         5
           4       0.34      0.40      0.37        42
           5       0.22      0.19      0.20        26
           6       0.26      0.30      0.28        20
           7       0.11      0.12      0.12         8
           8       0.46      0.30      0.36        20
           9       1.00      1.00      1.00        17
          10       0.30      0.17      0.21        18
          11       0.20      0.14      0.16        58
          12       0.48      0.39      0.43        51
          13       0.00      0.00      0.00         5
          14       0.25      0.14      0.18         7
          15       0.00      0.00      0.00         5
          16       0.00      0.00      0.00         3
          17       0.77      0.68      0.72        72
          18       0.27      0.30      0.29        20
          19       0.33      0.15      0.21        13
          20       0.00      0.00      0.00         2
          21       0.00      0.00      0.00         8
          22       0.56      0.25      0.34        20
          23       0.28      0.33      0.30        42
          24       0.00      0.00      0.00         6
          25       0.08      0.04      0.06        24
          27       0.22      0.14      0.17        14
          28       0.08      0.08      0.08        12
          29       0.00      0.00      0.00         1
          30       1.00      0.50      0.67         2
          31       0.00      0.00      0.00         2
          32       0.00      0.00      0.00         2
          33       0.50      0.33      0.40         6
          34       0.07      0.04      0.05        26
          35       0.00      0.00      0.00         8
          36       0.60      0.30      0.40        10
          37       0.00      0.00      0.00         8
          39       0.00      0.00      0.00         1
          40       0.12      0.08      0.10        12
          41       0.00      0.00      0.00         2
          42       0.00      0.00      0.00         9
          43       0.14      0.17      0.15         6
          45       0.68      0.54      0.60        28
          46       0.00      0.00      0.00         2
          47       0.00      0.00      0.00         1
          48       0.00      0.00      0.00         1
          49       0.50      0.33      0.40         3
          51       0.00      0.00      0.00         1
          52       0.00      0.00      0.00         0
          55       0.00      0.00      0.00         0
          56       0.66      0.46      0.54        46
          57       1.00      0.20      0.33         5
          59       0.17      0.20      0.18         5
          60       0.00      0.00      0.00         1
          62       0.00      0.00      0.00         2
          63       0.00      0.00      0.00         1
          64       0.00      0.00      0.00         1
          66       0.00      0.00      0.00         1
          67       0.40      0.33      0.36        18
          68       0.00      0.00      0.00         0
          70       0.00      0.00      0.00         1
          72       0.56      0.76      0.64       121
          73       0.29      0.16      0.21        56

    accuracy                           0.57      1700
   macro avg       0.23      0.18      0.19      1700
weighted avg       0.54      0.57      0.54      1700

Time taken: 2.114424s to run the model
```

Random Forest

Random forests or random decision forests technique is an ensemble learning method for text classification. This method was introduced by T. Kam Ho in 1995 for first time which used t trees in parallel. This technique was later developed by L. Breiman in 1999 that they found converged for RF as a margin measure




```
==================================
Training accuracy: 95.59%
Testing accuracy: 61.18%
==================================
Confusion matrix:
 [[758   0   0 ...   0   0   0]
 [  2   0   0 ...   0   1   0]
 [ 15   0   8 ...   0   0   1]
 ...
 [  1   0   0 ...   0   0   0]
 [  2   1   2 ...   0 100   7]
 [ 16   0   0 ...   0  28   9]]
==================================
```

```
==========================================================
Classification report:
              precision    recall  f1-score   support

           0       0.59      1.00      0.74       761
           1       0.00      0.00      0.00         8
           2       0.80      0.33      0.47        24
           3       0.00      0.00      0.00         5
           4       0.61      0.40      0.49        42
           5       0.40      0.08      0.13        26
           6       1.00      0.15      0.26        20
           7       0.00      0.00      0.00         8
           8       0.00      0.00      0.00        20
           9       1.00      0.82      0.90        17
          10       1.00      0.06      0.11        18
          11       1.00      0.02      0.03        58
          12       0.77      0.39      0.52        51
          13       0.00      0.00      0.00         5
          14       1.00      0.14      0.25         7
          15       0.00      0.00      0.00         5
          16       1.00      0.33      0.50         3
          17       0.94      0.71      0.81        72
          18       0.83      0.25      0.38        20
          19       0.00      0.00      0.00        13
          20       0.00      0.00      0.00         2
          21       0.00      0.00      0.00         8
          22       0.50      0.05      0.09        20
          23       1.00      0.10      0.17        42
          24       0.00      0.00      0.00         6
          25       0.00      0.00      0.00        24
          27       0.25      0.07      0.11        14
          28       0.50      0.08      0.14        12
          29       0.00      0.00      0.00         1
          30       0.00      0.00      0.00         2
          31       0.00      0.00      0.00         2
          32       0.00      0.00      0.00         2
          33       0.33      0.17      0.22         6
          34       0.67      0.08      0.14        26
          35       0.00      0.00      0.00         8
          36       1.00      0.20      0.33        10
          37       0.00      0.00      0.00         8
          39       0.00      0.00      0.00         1
          40       0.00      0.00      0.00        12
          41       0.00      0.00      0.00         2
          42       0.00      0.00      0.00         9
          43       0.00      0.00      0.00         6
          45       0.76      0.46      0.58        28
          46       0.00      0.00      0.00         2
          47       0.00      0.00      0.00         1
          48       0.00      0.00      0.00         1
          49       0.00      0.00      0.00         3
          51       0.00      0.00      0.00         1
          56       0.79      0.50      0.61        46
          57       1.00      0.20      0.33         5
          59       0.00      0.00      0.00         5
          60       0.00      0.00      0.00         1
          62       0.00      0.00      0.00         2
          63       0.00      0.00      0.00         1
          64       0.00      0.00      0.00         1
          66       0.00      0.00      0.00         1
          67       0.00      0.00      0.00        18
          70       0.00      0.00      0.00         1
          72       0.59      0.83      0.69       121
          73       0.47      0.16      0.24        56

    accuracy                           0.61      1700
   macro avg       0.31      0.13      0.15      1700
weighted avg       0.59      0.61      0.52      1700

Time taken: 11.263145s to run the model
```

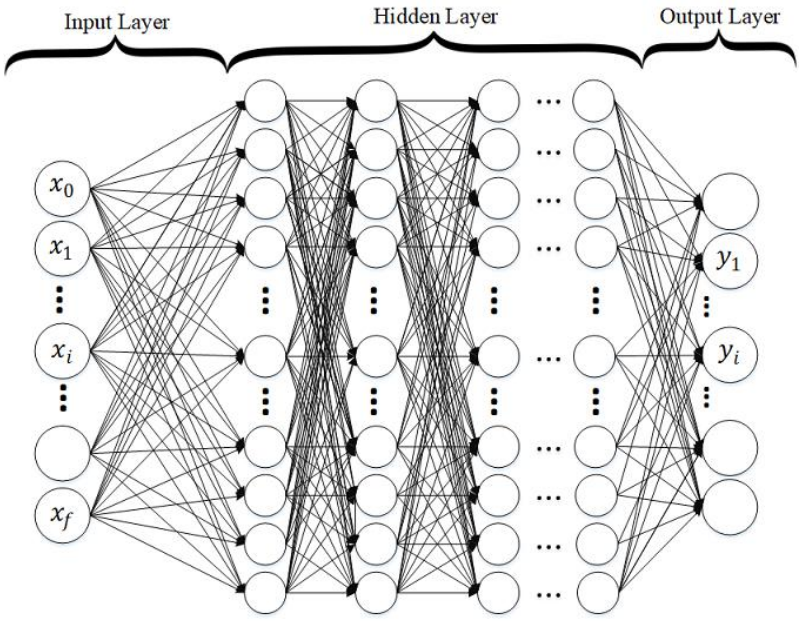
Neural Network

A Neural Network, unlike statistical ML algorithms, is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. Neural networks can adapt to changing input; so the network generates the best possible result without needing to redesign the output criteria.

I. Define a function for checkpoints

Deep Neural Networks

Deep Neural Networks architectures are designed to learn through multiple connection of layers where each single layer only receives connection from previous and provides connections only to the next layer in hidden part. The input is a connection of feature space (As discussed in Section Feature_extraction with first hidden layer. For Deep Neural Networks (DNN), input layer could be tf-ifd, word embedding, or etc. as shown in standard DNN in Figure. The output layer houses neurons equal to the number of classes for multi-class classification and only one neuron for binary classification. But our main contribution in this paper is that we have many trained DNNs to serve different purposes. Here, we have multi-class DNNs where each learning model is generated randomly (number of nodes in each layer as well as the number of layers are randomly assigned). Our implementation of Deep Neural Network (DNN) is basically a discriminatively trained model that uses standard back-propagation algorithm and sigmoid or ReLU as activation functions. The output layer for multi-class classification should use Softmax



```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 512)               2560512
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 512)               262656
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
batch_normalization_1 (Batch (None, 512)               2048
_________________________________________________________________
dense_3 (Dense)              (None, 512)               262656
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0
_________________________________________________________________
batch_normalization_2 (Batch (None, 512)               2048
_________________________________________________________________
dense_4 (Dense)              (None, 512)               262656
_________________________________________________________________
dropout_4 (Dropout)          (None, 512)               0
_________________________________________________________________
batch_normalization_3 (Batch (None, 512)               2048
_________________________________________________________________
dense_5 (Dense)              (None, 512)               262656
_________________________________________________________________
dropout_5 (Dropout)          (None, 512)               0
_________________________________________________________________
batch_normalization_4 (Batch (None, 512)               2048
_________________________________________________________________
dense_6 (Dense)              (None, 75)                38475
=================================================================
Total params: 3,657,803
Trainable params: 3,653,707
Non-trainable params: 4,096
_________________________________________________________________
None
```

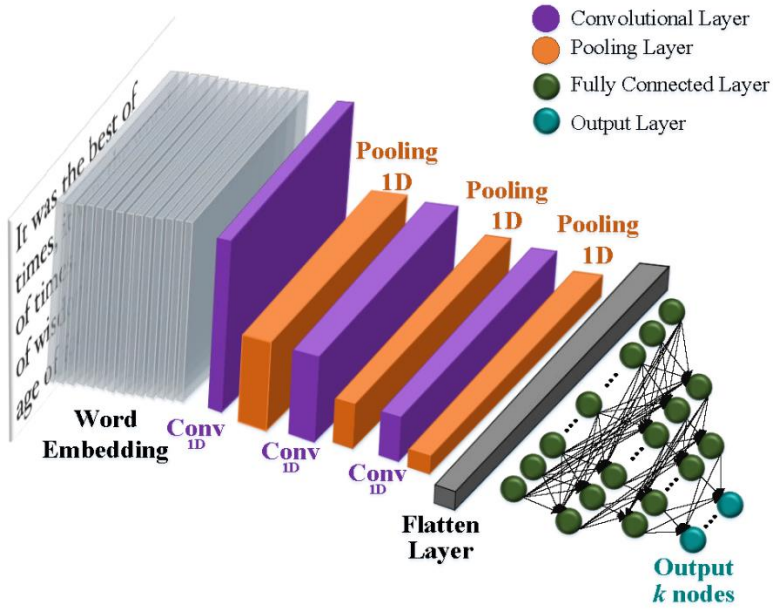
```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:223: The name tf.variables_i
nitializer is deprecated. Please use tf.compat.v1.variables_initializer instead.

 - 10s - loss: 2.9371 - acc: 0.4151 - val_loss: 2.0986 - val_acc: 0.5506

Epoch 00001: val_loss improved from inf to 2.09862, saving model to Weights/NN_epoch01_loss2.0986.h5
Epoch 2/10
 - 1s - loss: 1.8714 - acc: 0.5768 - val_loss: 1.7426 - val_acc: 0.5935

Epoch 00002: val_loss improved from 2.09862 to 1.74256, saving model to Weights/NN_epoch02_loss1.7426.h5
Epoch 3/10
 - 1s - loss: 1.5311 - acc: 0.6276 - val_loss: 1.6726 - val_acc: 0.6082

Epoch 00003: val_loss improved from 1.74256 to 1.67264, saving model to Weights/NN_epoch03_loss1.6726.h5
Epoch 4/10
 - 1s - loss: 1.2625 - acc: 0.6744 - val_loss: 1.8418 - val_acc: 0.6253

Epoch 00004: val_loss did not improve from 1.67264
Epoch 5/10
 - 1s - loss: 1.0332 - acc: 0.7281 - val_loss: 1.6876 - val_acc: 0.6324

Epoch 00005: val_loss did not improve from 1.67264
Epoch 6/10
 - 1s - loss: 0.8449 - acc: 0.7713 - val_loss: 1.8516 - val_acc: 0.6494

Epoch 00006: val_loss did not improve from 1.67264
Epoch 7/10
 - 1s - loss: 0.6933 - acc: 0.8125 - val_loss: 1.8501 - val_acc: 0.6465

Epoch 00007: val_loss did not improve from 1.67264
Epoch 8/10
 - 1s - loss: 0.5928 - acc: 0.8362 - val_loss: 1.9391 - val_acc: 0.6465

Epoch 00008: val_loss did not improve from 1.67264
Epoch 9/10
 - 1s - loss: 0.5128 - acc: 0.8559 - val_loss: 2.0079 - val_acc: 0.6441

Epoch 00009: val_loss did not improve from 1.67264
Epoch 10/10
 - 1s - loss: 0.4458 - acc: 0.8766 - val_loss: 2.0172 - val_acc: 0.6565

Epoch 00010: val_loss did not improve from 1.67264
```

Convolutional Neural Networks

Another deep learning architecture that is employed for hierarchical document classification is Convolutional Neural Networks (CNN) . Although originally built for image processing with architecture similar to the visual cortex, CNNs have also been effectively used for text classification. In a basic CNN for image processing, an image tensor is convolved with a set of kernels of size d by d. These convolution layers are called feature maps and can be stacked to provide multiple filters on the input. To reduce the computational complexity, CNNs use pooling which reduces the size of the output from one layer to the next in the network. Different pooling techniques are used to reduce outputs while preserving important features.

The most common pooling method is max pooling where the maximum element is selected from the pooling window. In order to feed the pooled output from stacked featured maps to the next layer, the maps are flattened into one column. The final layers in a CNN are typically fully connected dense layers. In general, during the back-propagation step of a convolutional neural network not only the weights are adjusted but also the feature detector filters. A potential problem of CNN used for text is the number of ‘channels’, Sigma (size of the feature

space). This might be very large (e.g. 50K), for text but for images this is less of a problem (e.g. only 3 channels of RGB). This means the dimensionality of the CNN for text is very high.




```
Found 15207 unique tokens.
(8500, 500)
Total 400000 word vectors.
```

```
Model: "model_1"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            (None, 500)          0
__________________________________________________________________________________________________
embedding_1 (Embedding)         (None, 500, 200)     3041600     input_1[0][0]
__________________________________________________________________________________________________
conv1d_1 (Conv1D)               (None, 499, 128)     51328       embedding_1[0][0]
__________________________________________________________________________________________________
conv1d_2 (Conv1D)               (None, 498, 128)     76928       embedding_1[0][0]
__________________________________________________________________________________________________
conv1d_3 (Conv1D)               (None, 497, 128)     102528      embedding_1[0][0]
__________________________________________________________________________________________________
conv1d_4 (Conv1D)               (None, 496, 128)     128128      embedding_1[0][0]
__________________________________________________________________________________________________
conv1d_5 (Conv1D)               (None, 495, 128)     153728      embedding_1[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]
__________________________________________________________________________________________________
max_pooling1d_5 (MaxPooling1D)  (None, 99, 128)      0           conv1d_5[0][0]
__________________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 495, 128)     0           max_pooling1d_1[0][0]
                                                                 max_pooling1d_2[0][0]
                                                                 max_pooling1d_3[0][0]
                                                                 max_pooling1d_4[0][0]
                                                                 max_pooling1d_5[0][0]
__________________________________________________________________________________________________
conv1d_6 (Conv1D)               (None, 491, 128)     82048       concatenate_1[0][0]
__________________________________________________________________________________________________
dropout_6 (Dropout)             (None, 491, 128)     0           conv1d_6[0][0]
__________________________________________________________________________________________________
batch_normalization_5 (BatchNor (None, 491, 128)     512         dropout_6[0][0]
__________________________________________________________________________________________________
max_pooling1d_6 (MaxPooling1D)  (None, 98, 128)      0           batch_normalization_5[0][0]
__________________________________________________________________________________________________
conv1d_7 (Conv1D)               (None, 94, 128)      82048       max_pooling1d_6[0][0]
__________________________________________________________________________________________________
dropout_7 (Dropout)             (None, 94, 128)      0           conv1d_7[0][0]
__________________________________________________________________________________________________
batch_normalization_6 (BatchNor (None, 94, 128)      512         dropout_7[0][0]
__________________________________________________________________________________________________
max_pooling1d_7 (MaxPooling1D)  (None, 3, 128)       0           batch_normalization_6[0][0]
__________________________________________________________________________________________________
flatten_1 (Flatten)             (None, 384)          0           max_pooling1d_7[0][0]
__________________________________________________________________________________________________
dense_7 (Dense)                 (None, 1024)         394240      flatten_1[0][0]
__________________________________________________________________________________________________
dropout_8 (Dropout)             (None, 1024)         0           dense_7[0][0]
__________________________________________________________________________________________________
dense_8 (Dense)                 (None, 512)          524800      dropout_8[0][0]
__________________________________________________________________________________________________
dropout_9 (Dropout)             (None, 512)          0           dense_8[0][0]
__________________________________________________________________________________________________
dense_9 (Dense)                 (None, 75)           38475       dropout_9[0][0]
==================================================================================================
Total params: 4,676,875
Trainable params: 4,676,363
Non-trainable params: 512
```

```
_________________________________________________________________
None
Train on 6800 samples, validate on 1700 samples
Epoch 1/10
6800/6800 [==============================] - 10s 1ms/step - loss: 3.1009 - acc: 0.4401 - val_loss: 3.7124 - val_acc: 0.4476

Epoch 00001: val_loss improved from inf to 3.71238, saving model to Weights/CNN_epoch01_loss3.7124.h5
Epoch 2/10
6800/6800 [==============================] - 4s 524us/step - loss: 2.5220 - acc: 0.4722 - val_loss: 3.3454 - val_acc: 0.4488

Epoch 00002: val_loss improved from 3.71238 to 3.34540, saving model to Weights/CNN_epoch02_loss3.3454.h5
Epoch 3/10
6800/6800 [==============================] - 4s 523us/step - loss: 2.4466 - acc: 0.4862 - val_loss: 3.4509 - val_acc: 0.4253

Epoch 00003: val_loss did not improve from 3.34540
Epoch 4/10
6800/6800 [==============================] - 4s 529us/step - loss: 2.2708 - acc: 0.5231 - val_loss: 3.1966 - val_acc: 0.3576

Epoch 00004: val_loss improved from 3.34540 to 3.19658, saving model to Weights/CNN_epoch04_loss3.1966.h5
Epoch 5/10
6800/6800 [==============================] - 4s 530us/step - loss: 2.1526 - acc: 0.5413 - val_loss: 3.0367 - val_acc: 0.3553

Epoch 00005: val_loss improved from 3.19658 to 3.03670, saving model to Weights/CNN_epoch05_loss3.0367.h5
Epoch 6/10
6800/6800 [==============================] - 4s 527us/step - loss: 2.0668 - acc: 0.5456 - val_loss: 2.9317 - val_acc: 0.4971

Epoch 00006: val_loss improved from 3.03670 to 2.93171, saving model to Weights/CNN_epoch06_loss2.9317.h5
Epoch 7/10
6800/6800 [==============================] - 4s 521us/step - loss: 1.9926 - acc: 0.5463 - val_loss: 2.9903 - val_acc: 0.4418

Epoch 00007: val_loss did not improve from 2.93171
Epoch 8/10
6800/6800 [==============================] - 4s 528us/step - loss: 1.9284 - acc: 0.5538 - val_loss: 2.9125 - val_acc: 0.4635

Epoch 00008: val_loss improved from 2.93171 to 2.91249, saving model to Weights/CNN_epoch08_loss2.9125.h5
Epoch 9/10
6800/6800 [==============================] - 4s 527us/step - loss: 1.8370 - acc: 0.5668 - val_loss: 2.7214 - val_acc: 0.4724

Epoch 00009: val_loss improved from 2.91249 to 2.72139, saving model to Weights/CNN_epoch09_loss2.7214.h5
Epoch 10/10
6800/6800 [==============================] - 4s 528us/step - loss: 1.7882 - acc: 0.5703 - val_loss: 2.6176 - val_acc: 0.4559

Epoch 00010: val_loss improved from 2.72139 to 2.61758, saving model to Weights/CNN_epoch10_loss2.6176.h5
Estimator: <keras.engine.training.Model object at 0x7f5284b937b8>
================================================================================
Training accuracy: 50.28%
Testing accuracy: 45.59%
================================================================================
Confusion matrix:
 [[721   0   0 ...   0   0   0]
 [  0   0   0 ...   0   0   0]
 [  5   0   0 ...   0   0   0]
 ...
 [  1   0   0 ...   0   0   0]
 [  4   0   0 ...   0  24   0]
 [ 10   0   0 ...   0   0   0]]
================================================================================
```

```
------------------------------------------------------------
Classification report:
              precision    recall  f1-score   support

           0       0.65      0.95      0.77       761
           1       0.00      0.00      0.00         8
           2       0.00      0.00      0.00        24
           3       0.00      0.00      0.00         5
           4       0.03      0.26      0.06        42
           5       0.00      0.00      0.00        26
           6       0.00      0.00      0.00        20
           7       0.00      0.00      0.00         8
           8       0.00      0.00      0.00        20
           9       0.00      0.00      0.00        17
          10       0.00      0.00      0.00        18
          11       0.00      0.00      0.00        58
          12       0.08      0.37      0.13        51
          13       0.00      0.00      0.00         5
          14       0.00      0.00      0.00         7
          15       0.00      0.00      0.00         5
          16       0.00      0.00      0.00         3
          17       0.00      0.00      0.00        72
          18       0.00      0.00      0.00        20
          19       0.00      0.00      0.00        13
          20       0.00      0.00      0.00         2
          21       0.00      0.00      0.00         8
          22       0.00      0.00      0.00        20
          23       0.00      0.00      0.00        42
          24       0.00      0.00      0.00         6
          25       0.00      0.00      0.00        24
          27       0.00      0.00      0.00        14
          28       0.00      0.00      0.00        12
          29       0.00      0.00      0.00         1
          30       0.00      0.00      0.00         2
          31       0.00      0.00      0.00         2
          32       0.00      0.00      0.00         2
          33       0.00      0.00      0.00         6
          34       0.00      0.00      0.00        26
          35       0.00      0.00      0.00         8
          36       0.00      0.00      0.00        10
          37       0.00      0.00      0.00         8
          39       0.00      0.00      0.00         1
          40       0.00      0.00      0.00        12
          41       0.00      0.00      0.00         2
          42       0.00      0.00      0.00         9
          43       0.00      0.00      0.00         6
          45       0.00      0.00      0.00        28
          46       0.00      0.00      0.00         2
          47       0.00      0.00      0.00         1
          48       0.00      0.00      0.00         1
          49       0.00      0.00      0.00         3
          51       0.00      0.00      0.00         1
          56       0.00      0.00      0.00        46
          57       0.00      0.00      0.00         5
          59       0.00      0.00      0.00         5
          60       0.00      0.00      0.00         1
          62       0.00      0.00      0.00         2
          63       0.00      0.00      0.00         1
          64       0.00      0.00      0.00         1
          66       0.00      0.00      0.00         1
          67       0.00      0.00      0.00        18
          70       0.00      0.00      0.00         1
          72       0.96      0.20      0.33       121
          73       0.00      0.00      0.00        56

    accuracy                           0.46      1700
   macro avg       0.03      0.03      0.02      1700
weighted avg       0.36      0.46      0.37      1700

Time taken: 47.869170s to run the model
```

Recurrent Neural Networks

RNN assigns more weights to the previous data points of sequence. Therefore, this technique is a powerful method for text, string and sequential data classification. Moreover, this technique could be used for image classification as we did in this work. In RNN, the neural net considers the information of previous nodes in a very sophisticated method which allows for better semantic analysis of the structures in the dataset.

Gated Recurrent Unit (GRU) is a gating mechanism for RNN which was introduced by J. Chung et al. and K.Cho et al.. GRU is a simplified variant of the LSTM architecture, but there are differences as follows: GRU contains two gates and does not possess any internal memory (as shown in Figure; and finally, a second non-linearity is not applied (tanh in Figure).

Layer (type)	Output Shape	Param #
embedding_2 (Embedding)	(None, 500, 100)	1162700
gru_1 (GRU)	(None, 500, 32)	12768
dropout_10 (Dropout)	(None, 500, 32)	0
batch_normalization_7 (Batch	(None, 500, 32)	128
gru_2 (GRU)	(None, 500, 32)	6240
dropout_11 (Dropout)	(None, 500, 32)	0
batch_normalization_8 (Batch	(None, 500, 32)	128
gru_3 (GRU)	(None, 500, 32)	6240
dropout_12 (Dropout)	(None, 500, 32)	0
batch_normalization_9 (Batch	(None, 500, 32)	128
gru_4 (GRU)	(None, 32)	6240
dropout_13 (Dropout)	(None, 32)	0
batch_normalization_10 (Batc	(None, 32)	128
dense_10 (Dense)	(None, 256)	8448
batch_normalization_11 (Batc	(None, 256)	1024
dense_11 (Dense)	(None, 75)	19275

Total params: 1,223,447
Trainable params: 1,222,679
Non-trainable params: 768

```
_________________________________________________________________
None
Train on 6800 samples, validate on 1700 samples
Epoch 1/10
6800/6800 [==============================] - 162s 24ms/step - loss: 4.7133 - acc: 0.0418 - val_loss: 3.9959 - val_acc: 0.1565

Epoch 00001: val_loss improved from inf to 3.99592, saving model to Weights/RNN_epoch01_loss3.9959.h5
Epoch 2/10
6800/6800 [==============================] - 158s 23ms/step - loss: 3.8224 - acc: 0.2471 - val_loss: 3.0233 - val_acc: 0.4476

Epoch 00002: val_loss improved from 3.99592 to 3.02328, saving model to Weights/RNN_epoch02_loss3.0233.h5
Epoch 3/10
6800/6800 [==============================] - 158s 23ms/step - loss: 3.1724 - acc: 0.4147 - val_loss: 2.8416 - val_acc: 0.4988

Epoch 00003: val_loss improved from 3.02328 to 2.84162, saving model to Weights/RNN_epoch03_loss2.8416.h5
Epoch 4/10
6800/6800 [==============================] - 157s 23ms/step - loss: 2.8593 - acc: 0.4822 - val_loss: 2.5870 - val_acc: 0.5029

Epoch 00004: val_loss improved from 2.84162 to 2.58699, saving model to Weights/RNN_epoch04_loss2.5870.h5
Epoch 5/10
6800/6800 [==============================] - 168s 25ms/step - loss: 2.7069 - acc: 0.5056 - val_loss: 2.5713 - val_acc: 0.5076

Epoch 00005: val_loss improved from 2.58699 to 2.57132, saving model to Weights/RNN_epoch05_loss2.5713.h5
Epoch 6/10
6800/6800 [==============================] - 156s 23ms/step - loss: 2.6275 - acc: 0.5163 - val_loss: 2.5308 - val_acc: 0.5094

Epoch 00006: val_loss improved from 2.57132 to 2.53079, saving model to Weights/RNN_epoch06_loss2.5308.h5
Epoch 7/10
6800/6800 [==============================] - 154s 23ms/step - loss: 2.5612 - acc: 0.5250 - val_loss: 2.6692 - val_acc: 0.4894

Epoch 00007: val_loss did not improve from 2.53079
Epoch 8/10
6800/6800 [==============================] - 154s 23ms/step - loss: 2.5204 - acc: 0.5237 - val_loss: 2.5029 - val_acc: 0.5094

Epoch 00008: val_loss improved from 2.53079 to 2.50291, saving model to Weights/RNN_epoch08_loss2.5029.h5
Epoch 9/10
6800/6800 [==============================] - 154s 23ms/step - loss: 2.4551 - acc: 0.5316 - val_loss: 2.3792 - val_acc: 0.5135

Epoch 00009: val_loss improved from 2.50291 to 2.37919, saving model to Weights/RNN_epoch09_loss2.3792.h5
Epoch 10/10
6800/6800 [==============================] - 152s 22ms/step - loss: 2.4053 - acc: 0.5306 - val_loss: 2.4194 - val_acc: 0.5118

Epoch 00010: val_loss did not improve from 2.37919
Estimator: <keras.engine.sequential.Sequential object at 0x7f528481cc18>
================================================================================
Training accuracy: 54.65%
Testing accuracy: 51.18%
================================================================================
Confusion matrix:
 [[724   0   4 ...   0  19   0]
 [  0   0   0 ...   0   6   0]
 [ 11   0   0 ...   0  12   0]
 ...
 [  1   0   0 ...   0   0   0]
 [  8   0   0 ...   0 112   0]
 [ 16   0   0 ...   0  38   0]]
================================================================================
```

```
Classification report:
              precision    recall  f1-score   support

           0       0.57      0.95      0.71       761
           1       0.00      0.00      0.00         8
           2       0.00      0.00      0.00        24
           3       0.00      0.00      0.00         5
           4       0.24      0.19      0.21        42
           5       0.00      0.00      0.00        26
           6       0.00      0.00      0.00        20
           7       0.00      0.00      0.00         8
           8       0.00      0.00      0.00        20
           9       0.00      0.00      0.00        17
          10       0.00      0.00      0.00        18
          11       0.00      0.00      0.00        58
          12       0.00      0.00      0.00        51
          13       0.00      0.00      0.00         5
          14       0.00      0.00      0.00         7
          15       0.00      0.00      0.00         5
          16       0.00      0.00      0.00         3
          17       0.58      0.36      0.44        72
          18       0.00      0.00      0.00        20
          19       0.00      0.00      0.00        13
          20       0.00      0.00      0.00         2
          21       0.00      0.00      0.00         8
          22       0.00      0.00      0.00        20
          23       0.00      0.00      0.00        42
          24       0.00      0.00      0.00         6
          25       0.00      0.00      0.00        24
          27       0.00      0.00      0.00        14
          28       0.00      0.00      0.00        12
          29       0.00      0.00      0.00         1
          30       0.00      0.00      0.00         2
          31       0.00      0.00      0.00         2
          32       0.00      0.00      0.00         2
          33       0.00      0.00      0.00         6
          34       0.00      0.00      0.00        26
          35       0.00      0.00      0.00         8
          36       0.00      0.00      0.00        10
          37       0.00      0.00      0.00         8
          39       0.00      0.00      0.00         1
          40       0.00      0.00      0.00        12
          41       0.00      0.00      0.00         2
          42       0.00      0.00      0.00         9
          43       0.00      0.00      0.00         6
          45       0.00      0.00      0.00        28
          46       0.00      0.00      0.00         2
          47       0.00      0.00      0.00         1
          48       0.00      0.00      0.00         1
          49       0.00      0.00      0.00         3
          51       0.00      0.00      0.00         1
          56       0.00      0.00      0.00        46
          57       0.00      0.00      0.00         5
          59       0.00      0.00      0.00         5
          60       0.00      0.00      0.00         1
          62       0.00      0.00      0.00         2
          63       0.00      0.00      0.00         1
          64       0.00      0.00      0.00         1
          66       0.00      0.00      0.00         1
          67       0.00      0.00      0.00        18
          70       0.00      0.00      0.00         1
          72       0.35      0.93      0.51       121
          73       0.00      0.00      0.00        56

    accuracy                           0.51      1700
   macro avg       0.03      0.04      0.03      1700
weighted avg       0.31      0.51      0.38      1700

Time taken: 1842.884726s to run the model
```

Recurrent Convolutional Neural Networks

Recurrent Convolutional Neural Networks (RCNN) is also used for text classification. The main idea of this technique is capturing contextual information with the recurrent structure and constructing the representation of text using a convolutional neural network. This architecture is a combination of RNN and CNN to use advantages of both technique in a model.

Layer (type)	Output Shape	Param #	Connected to
input_2 (InputLayer)	(None, 500)	0	
embedding_3 (Embedding)	(None, 500, 100)	1162700	input_2[0][0]
conv1d_8 (Conv1D)	(None, 499, 128)	25728	embedding_3[0][0]
conv1d_9 (Conv1D)	(None, 498, 128)	38528	embedding_3[0][0]
conv1d_10 (Conv1D)	(None, 497, 128)	51328	embedding_3[0][0]
conv1d_11 (Conv1D)	(None, 496, 128)	64128	embedding_3[0][0]
conv1d_12 (Conv1D)	(None, 495, 128)	76928	embedding_3[0][0]
max_pooling1d_8 (MaxPooling1D)	(None, 99, 128)	0	conv1d_8[0][0]
max_pooling1d_9 (MaxPooling1D)	(None, 99, 128)	0	conv1d_9[0][0]
max_pooling1d_10 (MaxPooling1D)	(None, 99, 128)	0	conv1d_10[0][0]
max_pooling1d_11 (MaxPooling1D)	(None, 99, 128)	0	conv1d_11[0][0]
max_pooling1d_12 (MaxPooling1D)	(None, 99, 128)	0	conv1d_12[0][0]
concatenate_2 (Concatenate)	(None, 495, 128)	0	max_pooling1d_8[0][0] max_pooling1d_9[0][0] max_pooling1d_10[0][0] max_pooling1d_11[0][0] max_pooling1d_12[0][0]
conv1d_13 (Conv1D)	(None, 491, 128)	82048	concatenate_2[0][0]
dropout_14 (Dropout)	(None, 491, 128)	0	conv1d_13[0][0]
batch_normalization_12 (BatchNo	(None, 491, 128)	512	dropout_14[0][0]
max_pooling1d_13 (MaxPooling1D)	(None, 98, 128)	0	batch_normalization_12[0][0]
conv1d_14 (Conv1D)	(None, 94, 128)	82048	max_pooling1d_13[0][0]
dropout_15 (Dropout)	(None, 94, 128)	0	conv1d_14[0][0]
batch_normalization_13 (BatchNo	(None, 94, 128)	512	dropout_15[0][0]
max_pooling1d_14 (MaxPooling1D)	(None, 3, 128)	0	batch_normalization_13[0][0]
flatten_2 (Flatten)	(None, 384)	0	max_pooling1d_14[0][0]
dense_12 (Dense)	(None, 1024)	394240	flatten_2[0][0]
dropout_16 (Dropout)	(None, 1024)	0	dense_12[0][0]
dense_13 (Dense)	(None, 512)	524800	dropout_16[0][0]
dropout_17 (Dropout)	(None, 512)	0	dense_13[0][0]
dense_14 (Dense)	(None, 75)	38475	dropout_17[0][0]

Total params: 2,541,975
Trainable params: 2,541,463
Non-trainable params: 512

]

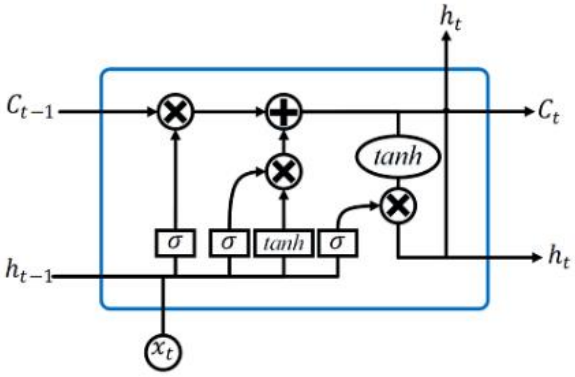
```
_________________________________________________________________________________________________
None
Train on 6800 samples, validate on 1700 samples
Epoch 1/10
6800/6800 [==============================] - 6s 881us/step - loss: 3.0501 - acc: 0.4438 - val_loss: 3.5348 - val_acc: 0.4476

Epoch 00001: val_loss improved from inf to 3.53482, saving model to Weights/RCNN_epoch01_loss3.5348.h5
Epoch 2/10
6800/6800 [==============================] - 4s 528us/step - loss: 2.4811 - acc: 0.4813 - val_loss: 3.4631 - val_acc: 0.4494

Epoch 00002: val_loss improved from 3.53482 to 3.46310, saving model to Weights/RCNN_epoch02_loss3.4631.h5
Epoch 3/10
6800/6800 [==============================] - 4s 531us/step - loss: 2.3434 - acc: 0.5160 - val_loss: 2.8409 - val_acc: 0.4647

Epoch 00003: val_loss improved from 3.46310 to 2.84091, saving model to Weights/RCNN_epoch03_loss2.8409.h5
Epoch 4/10
6800/6800 [==============================] - 4s 529us/step - loss: 2.1537 - acc: 0.5393 - val_loss: 2.7829 - val_acc: 0.4641

Epoch 00004: val_loss improved from 2.84091 to 2.78292, saving model to Weights/RCNN_epoch04_loss2.7829.h5
Epoch 5/10
6800/6800 [==============================] - 4s 528us/step - loss: 2.0946 - acc: 0.5381 - val_loss: 2.5877 - val_acc: 0.4647

Epoch 00005: val_loss improved from 2.78292 to 2.58768, saving model to Weights/RCNN_epoch05_loss2.5877.h5
Epoch 6/10
6800/6800 [==============================] - 4s 518us/step - loss: 2.0062 - acc: 0.5453 - val_loss: 2.7399 - val_acc: 0.4865

Epoch 00006: val_loss did not improve from 2.58768
Epoch 7/10
6800/6800 [==============================] - 4s 526us/step - loss: 1.9348 - acc: 0.5512 - val_loss: 2.8283 - val_acc: 0.4571

Epoch 00007: val_loss did not improve from 2.58768
Epoch 8/10
6800/6800 [==============================] - 4s 529us/step - loss: 1.8873 - acc: 0.5496 - val_loss: 2.7495 - val_acc: 0.4553

Epoch 00008: val_loss did not improve from 2.58768
Epoch 9/10
6800/6800 [==============================] - 4s 530us/step - loss: 1.7921 - acc: 0.5647 - val_loss: 2.5322 - val_acc: 0.4488

Epoch 00009: val_loss improved from 2.58768 to 2.53224, saving model to Weights/RCNN_epoch09_loss2.5322.h5
Epoch 10/10
6800/6800 [==============================] - 4s 519us/step - loss: 1.7208 - acc: 0.5749 - val_loss: 2.7362 - val_acc: 0.4541

Epoch 00010: val_loss did not improve from 2.53224
Estimator: <keras.engine.training.Model object at 0x7f512d70df98>
================================================================================
Training accuracy: 49.34%
Testing accuracy: 45.41%
================================================================================
Confusion matrix:
 [[730   0   0 ...   0   0   0]
 [  0   0   0 ...   0   0   0]
 [  4   0   0 ...   0   0   0]
 ...
 [  1   0   0 ...   0   0   0]
 [  1   0   0 ...   0  24   0]
 [  7   0   0 ...   0   0   0]]
================================================================================
```

```
============================================================
Classification report:
              precision    recall  f1-score   support

           0       0.65      0.96      0.77       761
           1       0.00      0.00      0.00         8
           2       0.00      0.00      0.00        24
           3       0.00      0.00      0.00         5
           4       0.00      0.00      0.00        42
           5       0.00      0.00      0.00        26
           6       0.00      0.00      0.00        20
           7       0.00      0.00      0.00         8
           8       0.00      0.00      0.00        20
           9       0.00      0.00      0.00        17
          10       0.00      0.00      0.00        18
          11       0.00      0.00      0.00        58
          12       0.03      0.35      0.06        51
          13       0.00      0.00      0.00         5
          14       0.00      0.00      0.00         7
          15       0.00      0.00      0.00         5
          16       0.00      0.00      0.00         3
          17       0.00      0.00      0.00        72
          18       0.00      0.00      0.00        20
          19       0.00      0.00      0.00        13
          20       0.00      0.00      0.00         2
          21       0.00      0.00      0.00         8
          22       0.00      0.00      0.00        20
          23       0.00      0.00      0.00        42
          24       0.00      0.00      0.00         6
          25       0.00      0.00      0.00        24
          27       0.00      0.00      0.00        14
          28       0.00      0.00      0.00        12
          29       0.00      0.00      0.00         1
          30       0.00      0.00      0.00         2
          31       0.00      0.00      0.00         2
          32       0.00      0.00      0.00         2
          33       0.00      0.00      0.00         6
          34       0.00      0.00      0.00        26
          35       0.00      0.00      0.00         8
          36       0.00      0.00      0.00        10
          37       0.00      0.00      0.00         8
          39       0.00      0.00      0.00         1
          40       0.00      0.00      0.00        12
          41       0.00      0.00      0.00         2
          42       0.00      0.00      0.00         9
          43       0.00      0.00      0.00         6
          45       0.00      0.00      0.00        28
          46       0.00      0.00      0.00         2
          47       0.00      0.00      0.00         1
          48       0.00      0.00      0.00         1
          49       0.00      0.00      0.00         3
          51       0.00      0.00      0.00         1
          56       0.00      0.00      0.00        46
          57       0.00      0.00      0.00         5
          59       0.00      0.00      0.00         5
          60       0.00      0.00      0.00         1
          62       0.00      0.00      0.00         2
          63       0.00      0.00      0.00         1
          64       0.00      0.00      0.00         1
          66       0.00      0.00      0.00         1
          67       0.00      0.00      0.00        18
          70       0.00      0.00      0.00         1
          72       0.96      0.20      0.33       121
          73       0.00      0.00      0.00        56

    accuracy                           0.45      1700
   macro avg       0.03      0.03      0.02      1700
weighted avg       0.36      0.45      0.37      1700

Time taken: 46.433537s to run the model
```

RNN with LSTM

Long Short-Term Memory~(LSTM) was introduced by S. Hochreiter and J. Schmidhuber and developed by many research scientists.

To deal with these problems Long Short-Term Memory (LSTM) is a special type of RNN that preserves long term dependency in a more effective way compared to the basic RNNs. This is particularly useful to overcome vanishing gradient problem as LSTM uses multiple gates to carefully regulate the amount of information that will be allowed into each node state. The figure shows the basic cell of a LSTM model.




```
Found 15207 unique tokens.
(8500, 500)
Total 400000 word vectors.
```

Layer (type)	Output Shape	Param #
embedding_4 (Embedding)	(None, 500, 50)	581350
dropout_18 (Dropout)	(None, 500, 50)	0
conv1d_15 (Conv1D)	(None, 499, 256)	25856
max_pooling1d_15 (MaxPooling	(None, 249, 256)	0
conv1d_16 (Conv1D)	(None, 248, 256)	131328
max_pooling1d_16 (MaxPooling	(None, 124, 256)	0
conv1d_17 (Conv1D)	(None, 123, 256)	131328
max_pooling1d_17 (MaxPooling	(None, 61, 256)	0
conv1d_18 (Conv1D)	(None, 60, 256)	131328
max_pooling1d_18 (MaxPooling	(None, 30, 256)	0
bidirectional_1 (Bidirection	(None, 30, 512)	1050624
bidirectional_2 (Bidirection	(None, 30, 512)	1574912
bidirectional_3 (Bidirection	(None, 30, 512)	1574912
bidirectional_4 (Bidirection	(None, 512)	1574912
dense_15 (Dense)	(None, 1024)	525312
dense_16 (Dense)	(None, 75)	76875
activation_1 (Activation)	(None, 75)	0

Total params: 7,378,737
Trainable params: 7,378,737
Non-trainable params: 0

```
_________________________________________________________________
None
Train on 6800 samples, validate on 1700 samples
Epoch 1/10
6800/6800 [==============================] - 33s 5ms/step - loss: 2.6414 - acc: 0.4649 - val_loss: 2.6477 - val_acc: 0.4476

Epoch 00001: val_loss improved from inf to 2.64768, saving model to Weights/LSTM_epoch01_loss2.6477.h5
Epoch 2/10
6800/6800 [==============================] - 26s 4ms/step - loss: 2.4783 - acc: 0.4728 - val_loss: 2.4821 - val_acc: 0.4476

Epoch 00002: val_loss improved from 2.64768 to 2.48211, saving model to Weights/LSTM_epoch02_loss2.4821.h5
Epoch 3/10
6800/6800 [==============================] - 25s 4ms/step - loss: 2.2372 - acc: 0.5129 - val_loss: 2.1757 - val_acc: 0.5059

Epoch 00003: val_loss improved from 2.48211 to 2.17570, saving model to Weights/LSTM_epoch03_loss2.1757.h5
Epoch 4/10
6800/6800 [==============================] - 25s 4ms/step - loss: 2.0503 - acc: 0.5425 - val_loss: 2.1780 - val_acc: 0.5082

Epoch 00004: val_loss did not improve from 2.17570
Epoch 5/10
6800/6800 [==============================] - 24s 4ms/step - loss: 1.9565 - acc: 0.5496 - val_loss: 2.0206 - val_acc: 0.5165

Epoch 00005: val_loss improved from 2.17570 to 2.02057, saving model to Weights/LSTM_epoch05_loss2.0206.h5
Epoch 6/10
6800/6800 [==============================] - 26s 4ms/step - loss: 1.9333 - acc: 0.5529 - val_loss: 2.0130 - val_acc: 0.5347

Epoch 00006: val_loss improved from 2.02057 to 2.01305, saving model to Weights/LSTM_epoch06_loss2.0130.h5
Epoch 7/10
6800/6800 [==============================] - 25s 4ms/step - loss: 1.8153 - acc: 0.5771 - val_loss: 2.0054 - val_acc: 0.5188

Epoch 00007: val_loss improved from 2.01305 to 2.00541, saving model to Weights/LSTM_epoch07_loss2.0054.h5
Epoch 8/10
6800/6800 [==============================] - 25s 4ms/step - loss: 1.7480 - acc: 0.5832 - val_loss: 2.0194 - val_acc: 0.5359

Epoch 00008: val_loss did not improve from 2.00541
Epoch 9/10
6800/6800 [==============================] - 25s 4ms/step - loss: 1.7086 - acc: 0.5907 - val_loss: 2.0240 - val_acc: 0.5335

Epoch 00009: val_loss did not improve from 2.00541
Epoch 10/10
6800/6800 [==============================] - 25s 4ms/step - loss: 1.6627 - acc: 0.5931 - val_loss: 1.9896 - val_acc: 0.5418

Epoch 00010: val_loss improved from 2.00541 to 1.98956, saving model to Weights/LSTM_epoch10_loss1.9896.h5
Estimator: <keras.engine.sequential.Sequential object at 0x7f5284bfd438>
================================================================================
Training accuracy: 59.97%
Testing accuracy: 54.18%
================================================================================
Confusion matrix:
 [[736   0   0 ...   0   0   0]
 [  0   0   0 ...   0   2   0]
 [  6   0   0 ...   0   9   0]
 ...
 [  1   0   0 ...   0   0   0]
 [  2   0   0 ...   0 103   0]
 [ 11   0   0 ...   0  38   0]]
================================================================================
```

Classification report:

	precision	recall	f1-score	support
0	0.66	0.97	0.78	761
1	0.00	0.00	0.00	8
2	0.00	0.00	0.00	24
3	0.00	0.00	0.00	5
4	0.13	0.29	0.18	42
5	0.09	0.12	0.10	26
6	0.00	0.00	0.00	20
7	0.00	0.00	0.00	8
8	0.00	0.00	0.00	20
9	1.00	0.24	0.38	17
10	0.00	0.00	0.00	18
11	0.00	0.00	0.00	58
12	0.11	0.27	0.15	51
13	0.00	0.00	0.00	5
14	0.00	0.00	0.00	7
15	0.00	0.00	0.00	5
16	0.00	0.00	0.00	3
17	0.46	0.68	0.55	72
18	0.00	0.00	0.00	20
19	0.00	0.00	0.00	13
20	0.00	0.00	0.00	2
21	0.00	0.00	0.00	8
22	0.00	0.00	0.00	20
23	0.00	0.00	0.00	42
24	0.00	0.00	0.00	6
25	0.00	0.00	0.00	24
27	0.00	0.00	0.00	14
28	0.00	0.00	0.00	12
29	0.00	0.00	0.00	1
30	0.00	0.00	0.00	2
31	0.00	0.00	0.00	2
32	0.00	0.00	0.00	2
33	0.00	0.00	0.00	6
34	0.00	0.00	0.00	26
35	0.00	0.00	0.00	8
36	0.00	0.00	0.00	10
37	0.00	0.00	0.00	8
39	0.00	0.00	0.00	1
40	0.00	0.00	0.00	12
41	0.00	0.00	0.00	2
42	0.00	0.00	0.00	9
43	0.00	0.00	0.00	6
45	0.00	0.00	0.00	28
46	0.00	0.00	0.00	2
47	0.00	0.00	0.00	1
48	0.00	0.00	0.00	1
49	0.00	0.00	0.00	3
51	0.00	0.00	0.00	1
56	0.00	0.00	0.00	46
57	0.00	0.00	0.00	5
59	0.00	0.00	0.00	5
60	0.00	0.00	0.00	1
62	0.00	0.00	0.00	2
63	0.00	0.00	0.00	1
64	0.00	0.00	0.00	1
66	0.00	0.00	0.00	1
67	0.00	0.00	0.00	18
70	0.00	0.00	0.00	1
72	0.47	0.85	0.60	121
73	0.00	0.00	0.00	56
accuracy			0.54	1700
macro avg	0.05	0.06	0.05	1700
weighted avg	0.37	0.54	0.43	1700

Time taken: 313.876035s to run the model

Milestone 2 (Continued… RNN with LSTM)

I. Load the balanced dataset
II. Load the Word2Vec model
III. Sequences will be padded or truncated to length 75
IV. Prepare the embbedings with 0's padding to max sequence length
V. Divide the original dataset into train and test split (70: 30)
VI. Run RNN with LSTM again

```
Train on 17306 samples, validate on 1923 samples
Epoch 1/20
17306/17306 [==============================] - 71s 4ms/step - loss: 2.7391 - acc: 0.2600 - val_loss: 1.8067 - val_acc: 0.4893
Epoch 2/20
17306/17306 [==============================] - 68s 4ms/step - loss: 1.6989 - acc: 0.5259 - val_loss: 1.2630 - val_acc: 0.6453
Epoch 3/20
17306/17306 [==============================] - 68s 4ms/step - loss: 1.2872 - acc: 0.6424 - val_loss: 1.0008 - val_acc: 0.7150
Epoch 4/20
17306/17306 [==============================] - 67s 4ms/step - loss: 1.0489 - acc: 0.7076 - val_loss: 0.8288 - val_acc: 0.7577
Epoch 5/20
17306/17306 [==============================] - 68s 4ms/step - loss: 0.8877 - acc: 0.7556 - val_loss: 0.7168 - val_acc: 0.7894
Epoch 6/20
17306/17306 [==============================] - 67s 4ms/step - loss: 0.7869 - acc: 0.7783 - val_loss: 0.6305 - val_acc: 0.8216
Epoch 7/20
17306/17306 [==============================] - 68s 4ms/step - loss: 0.7030 - acc: 0.8004 - val_loss: 0.5709 - val_acc: 0.8263
Epoch 8/20
17306/17306 [==============================] - 68s 4ms/step - loss: 0.6372 - acc: 0.8160 - val_loss: 0.5208 - val_acc: 0.8461
Epoch 9/20
17306/17306 [==============================] - 69s 4ms/step - loss: 0.5823 - acc: 0.8320 - val_loss: 0.4781 - val_acc: 0.8549
Epoch 10/20
17306/17306 [==============================] - 68s 4ms/step - loss: 0.5367 - acc: 0.8436 - val_loss: 0.4431 - val_acc: 0.8684
Epoch 11/20
17306/17306 [==============================] - 66s 4ms/step - loss: 0.5025 - acc: 0.8521 - val_loss: 0.4179 - val_acc: 0.8783
Epoch 12/20
17306/17306 [==============================] - 66s 4ms/step - loss: 0.4677 - acc: 0.8601 - val_loss: 0.3887 - val_acc: 0.8856
Epoch 13/20
17306/17306 [==============================] - 67s 4ms/step - loss: 0.4353 - acc: 0.8709 - val_loss: 0.3816 - val_acc: 0.8747
Epoch 14/20
17306/17306 [==============================] - 68s 4ms/step - loss: 0.4210 - acc: 0.8737 - val_loss: 0.3483 - val_acc: 0.8965
Epoch 15/20
17306/17306 [==============================] - 67s 4ms/step - loss: 0.4004 - acc: 0.8802 - val_loss: 0.3453 - val_acc: 0.9012
Epoch 16/20
17306/17306 [==============================] - 67s 4ms/step - loss: 0.3892 - acc: 0.8844 - val_loss: 0.3309 - val_acc: 0.8981
Epoch 17/20
17306/17306 [==============================] - 68s 4ms/step - loss: 0.3626 - acc: 0.8892 - val_loss: 0.3243 - val_acc: 0.9028
Epoch 18/20
17306/17306 [==============================] - 68s 4ms/step - loss: 0.3525 - acc: 0.8916 - val_loss: 0.3096 - val_acc: 0.9100
Epoch 19/20
17306/17306 [==============================] - 67s 4ms/step - loss: 0.3394 - acc: 0.8948 - val_loss: 0.3087 - val_acc: 0.9085
Epoch 20/20
17306/17306 [==============================] - 66s 4ms/step - loss: 0.3217 - acc: 0.9036 - val_loss: 0.3074 - val_acc: 0.9043
```

Iteration 1 ...changing the dropout value from 0.2 to 0.1

```
Train on 17306 samples, validate on 1923 samples
Epoch 1/20
17306/17306 [==============================] - 70s 4ms/step - loss: 2.4506 - acc: 0.3459 - val_loss: 1.5717 - val_acc: 0.5757
Epoch 2/20
17306/17306 [==============================] - 69s 4ms/step - loss: 1.3551 - acc: 0.6298 - val_loss: 1.0543 - val_acc: 0.7150
Epoch 3/20
17306/17306 [==============================] - 68s 4ms/step - loss: 0.9840 - acc: 0.7384 - val_loss: 0.7932 - val_acc: 0.7920
Epoch 4/20
17306/17306 [==============================] - 69s 4ms/step - loss: 0.7751 - acc: 0.7847 - val_loss: 0.6645 - val_acc: 0.8185
Epoch 5/20
17306/17306 [==============================] - 68s 4ms/step - loss: 0.6358 - acc: 0.8242 - val_loss: 0.5489 - val_acc: 0.8393
Epoch 6/20
17306/17306 [==============================] - 69s 4ms/step - loss: 0.5429 - acc: 0.8473 - val_loss: 0.5011 - val_acc: 0.8601
Epoch 7/20
17306/17306 [==============================] - 69s 4ms/step - loss: 0.4804 - acc: 0.8630 - val_loss: 0.4168 - val_acc: 0.8799
Epoch 8/20
17306/17306 [==============================] - 69s 4ms/step - loss: 0.4222 - acc: 0.8803 - val_loss: 0.3849 - val_acc: 0.8856
Epoch 9/20
17306/17306 [==============================] - 69s 4ms/step - loss: 0.3868 - acc: 0.8877 - val_loss: 0.3563 - val_acc: 0.8913
Epoch 10/20
17306/17306 [==============================] - 69s 4ms/step - loss: 0.3547 - acc: 0.8960 - val_loss: 0.3427 - val_acc: 0.8991
Epoch 11/20
17306/17306 [==============================] - 68s 4ms/step - loss: 0.3214 - acc: 0.9077 - val_loss: 0.3220 - val_acc: 0.9048
Epoch 12/20
17306/17306 [==============================] - 67s 4ms/step - loss: 0.3049 - acc: 0.9098 - val_loss: 0.2978 - val_acc: 0.9059
Epoch 13/20
17306/17306 [==============================] - 68s 4ms/step - loss: 0.2796 - acc: 0.9177 - val_loss: 0.2876 - val_acc: 0.9137
Epoch 14/20
17306/17306 [==============================] - 67s 4ms/step - loss: 0.2648 - acc: 0.9236 - val_loss: 0.2853 - val_acc: 0.9059
Epoch 15/20
17306/17306 [==============================] - 67s 4ms/step - loss: 0.2531 - acc: 0.9246 - val_loss: 0.2784 - val_acc: 0.9069
Epoch 16/20
17306/17306 [==============================] - 67s 4ms/step - loss: 0.2337 - acc: 0.9291 - val_loss: 0.2719 - val_acc: 0.9100
Epoch 17/20
17306/17306 [==============================] - 67s 4ms/step - loss: 0.2279 - acc: 0.9298 - val_loss: 0.2550 - val_acc: 0.9210
Epoch 18/20
17306/17306 [==============================] - 67s 4ms/step - loss: 0.2209 - acc: 0.9332 - val_loss: 0.2649 - val_acc: 0.9210
Epoch 19/20
17306/17306 [==============================] - 67s 4ms/step - loss: 0.2115 - acc: 0.9341 - val_loss: 0.2522 - val_acc: 0.9256
Epoch 20/20
17306/17306 [==============================] - 66s 4ms/step - loss: 0.2023 - acc: 0.9350 - val_loss: 0.2434 - val_acc: 0.9308
```

Iteration 2 ..adding more core to LTSM

```
Train on 17306 samples, validate on 1923 samples
Epoch 1/20
17306/17306 [==============================] - 68s 4ms/step - loss: 2.2320 - acc: 0.3925 - val_loss: 1.3308 - val_acc: 0.6365
Epoch 2/20
17306/17306 [==============================] - 68s 4ms/step - loss: 1.1154 - acc: 0.6917 - val_loss: 0.7884 - val_acc: 0.7826
Epoch 3/20
17306/17306 [==============================] - 68s 4ms/step - loss: 0.7616 - acc: 0.7887 - val_loss: 0.5929 - val_acc: 0.8310
Epoch 4/20
17306/17306 [==============================] - 68s 4ms/step - loss: 0.5980 - acc: 0.8329 - val_loss: 0.4852 - val_acc: 0.8492
Epoch 5/20
17306/17306 [==============================] - 68s 4ms/step - loss: 0.4662 - acc: 0.8706 - val_loss: 0.4057 - val_acc: 0.8721
Epoch 6/20
17306/17306 [==============================] - 68s 4ms/step - loss: 0.3935 - acc: 0.8880 - val_loss: 0.3604 - val_acc: 0.8970
Epoch 7/20
17306/17306 [==============================] - 68s 4ms/step - loss: 0.3435 - acc: 0.9000 - val_loss: 0.3185 - val_acc: 0.8965
Epoch 8/20
17306/17306 [==============================] - 68s 4ms/step - loss: 0.3005 - acc: 0.9125 - val_loss: 0.3007 - val_acc: 0.9038
Epoch 9/20
17306/17306 [==============================] - 69s 4ms/step - loss: 0.2659 - acc: 0.9227 - val_loss: 0.2854 - val_acc: 0.9090
Epoch 10/20
17306/17306 [==============================] - 70s 4ms/step - loss: 0.2436 - acc: 0.9284 - val_loss: 0.2743 - val_acc: 0.9095
Epoch 11/20
17306/17306 [==============================] - 71s 4ms/step - loss: 0.2261 - acc: 0.9303 - val_loss: 0.2500 - val_acc: 0.9256
Epoch 12/20
17306/17306 [==============================] - 72s 4ms/step - loss: 0.2108 - acc: 0.9365 - val_loss: 0.2508 - val_acc: 0.9272
Epoch 13/20
17306/17306 [==============================] - 71s 4ms/step - loss: 0.1933 - acc: 0.9410 - val_loss: 0.2474 - val_acc: 0.9225
Epoch 14/20
17306/17306 [==============================] - 69s 4ms/step - loss: 0.1887 - acc: 0.9401 - val_loss: 0.2359 - val_acc: 0.9288
Epoch 15/20
17306/17306 [==============================] - 69s 4ms/step - loss: 0.1724 - acc: 0.9449 - val_loss: 0.2413 - val_acc: 0.9324
Epoch 16/20
17306/17306 [==============================] - 70s 4ms/step - loss: 0.1670 - acc: 0.9473 - val_loss: 0.2311 - val_acc: 0.9324
Epoch 17/20
17306/17306 [==============================] - 68s 4ms/step - loss: 0.1642 - acc: 0.9466 - val_loss: 0.2288 - val_acc: 0.9303
Epoch 18/20
17306/17306 [==============================] - 68s 4ms/step - loss: 0.1619 - acc: 0.9476 - val_loss: 0.2313 - val_acc: 0.9314
Epoch 19/20
17306/17306 [==============================] - 69s 4ms/step - loss: 0.1528 - acc: 0.9497 - val_loss: 0.2259 - val_acc: 0.9314
Epoch 20/20
17306/17306 [==============================] - 69s 4ms/step - loss: 0.1509 - acc: 0.9493 - val_loss: 0.2269 - val_acc: 0.9246
```

Iteration 3 ....adding a dense and dropout and batchNormalistaion layer

```
Train on 17306 samples, validate on 1923 samples
Epoch 1/20
17306/17306 [==============================] - 71s 4ms/step - loss: 2.3474 - acc: 0.3646 - val_loss: 1.3097 - val_acc: 0.6495
Epoch 2/20
17306/17306 [==============================] - 69s 4ms/step - loss: 1.1205 - acc: 0.6874 - val_loss: 0.7741 - val_acc: 0.7780
Epoch 3/20
17306/17306 [==============================] - 68s 4ms/step - loss: 0.7102 - acc: 0.7980 - val_loss: 0.5141 - val_acc: 0.8461
Epoch 4/20
17306/17306 [==============================] - 69s 4ms/step - loss: 0.5083 - acc: 0.8520 - val_loss: 0.4037 - val_acc: 0.8820
Epoch 5/20
17306/17306 [==============================] - 68s 4ms/step - loss: 0.3922 - acc: 0.8822 - val_loss: 0.3437 - val_acc: 0.9017
Epoch 6/20
17306/17306 [==============================] - 69s 4ms/step - loss: 0.3192 - acc: 0.9032 - val_loss: 0.3259 - val_acc: 0.8970
Epoch 7/20
17306/17306 [==============================] - 68s 4ms/step - loss: 0.2799 - acc: 0.9118 - val_loss: 0.2821 - val_acc: 0.9100
Epoch 8/20
17306/17306 [==============================] - 69s 4ms/step - loss: 0.2446 - acc: 0.9223 - val_loss: 0.2820 - val_acc: 0.9111
Epoch 9/20
17306/17306 [==============================] - 69s 4ms/step - loss: 0.2292 - acc: 0.9255 - val_loss: 0.2788 - val_acc: 0.9225
Epoch 10/20
17306/17306 [==============================] - 70s 4ms/step - loss: 0.2104 - acc: 0.9309 - val_loss: 0.2573 - val_acc: 0.9262
Epoch 11/20
17306/17306 [==============================] - 69s 4ms/step - loss: 0.1971 - acc: 0.9352 - val_loss: 0.2676 - val_acc: 0.9225
Epoch 12/20
17306/17306 [==============================] - 69s 4ms/step - loss: 0.1875 - acc: 0.9379 - val_loss: 0.2547 - val_acc: 0.9204
Epoch 13/20
17306/17306 [==============================] - 68s 4ms/step - loss: 0.1803 - acc: 0.9411 - val_loss: 0.2624 - val_acc: 0.9272
Epoch 14/20
17306/17306 [==============================] - 69s 4ms/step - loss: 0.1726 - acc: 0.9444 - val_loss: 0.2500 - val_acc: 0.9272
Epoch 15/20
17306/17306 [==============================] - 70s 4ms/step - loss: 0.1689 - acc: 0.9434 - val_loss: 0.2570 - val_acc: 0.9288
Epoch 16/20
17306/17306 [==============================] - 68s 4ms/step - loss: 0.1643 - acc: 0.9444 - val_loss: 0.2843 - val_acc: 0.9199
Epoch 17/20
17306/17306 [==============================] - 69s 4ms/step - loss: 0.1722 - acc: 0.9422 - val_loss: 0.2800 - val_acc: 0.9241
```

iteration 4 ...optimizing Adam

```
Train on 17306 samples, validate on 1923 samples
Epoch 1/20
17306/17306 [==============================] - 72s 4ms/step - loss: 2.3593 - acc: 0.3639 - val_loss: 1.3768 - val_acc: 0.6355
Epoch 2/20
17306/17306 [==============================] - 67s 4ms/step - loss: 1.1354 - acc: 0.6848 - val_loss: 0.7751 - val_acc: 0.7842
Epoch 3/20
17306/17306 [==============================] - 68s 4ms/step - loss: 0.7054 - acc: 0.7992 - val_loss: 0.5247 - val_acc: 0.8482
Epoch 4/20
17306/17306 [==============================] - 69s 4ms/step - loss: 0.5034 - acc: 0.8542 - val_loss: 0.4234 - val_acc: 0.8762
Epoch 5/20
17306/17306 [==============================] - 70s 4ms/step - loss: 0.3971 - acc: 0.8824 - val_loss: 0.3549 - val_acc: 0.8934
Epoch 6/20
17306/17306 [==============================] - 71s 4ms/step - loss: 0.3281 - acc: 0.8978 - val_loss: 0.3206 - val_acc: 0.9100
Epoch 7/20
17306/17306 [==============================] - 70s 4ms/step - loss: 0.2791 - acc: 0.9123 - val_loss: 0.2898 - val_acc: 0.9126
Epoch 8/20
17306/17306 [==============================] - 70s 4ms/step - loss: 0.2475 - acc: 0.9207 - val_loss: 0.3115 - val_acc: 0.9142
Epoch 9/20
17306/17306 [==============================] - 70s 4ms/step - loss: 0.2281 - acc: 0.9264 - val_loss: 0.2908 - val_acc: 0.9210
Epoch 10/20
17306/17306 [==============================] - 69s 4ms/step - loss: 0.2093 - acc: 0.9316 - val_loss: 0.2688 - val_acc: 0.9262
Epoch 11/20
17306/17306 [==============================] - 69s 4ms/step - loss: 0.1924 - acc: 0.9361 - val_loss: 0.2657 - val_acc: 0.9236
Epoch 12/20
17306/17306 [==============================] - 70s 4ms/step - loss: 0.1948 - acc: 0.9356 - val_loss: 0.2895 - val_acc: 0.9173
Epoch 13/20
17306/17306 [==============================] - 71s 4ms/step - loss: 0.1829 - acc: 0.9401 - val_loss: 0.2876 - val_acc: 0.9298
Epoch 14/20
17306/17306 [==============================] - 69s 4ms/step - loss: 0.1766 - acc: 0.9415 - val_loss: 0.2804 - val_acc: 0.9288
```

Summary

The accuracy of each flavors of LSTM model is as follows in the table. This is clear indicative of how LSTM, in the family of RNN is efficient of dealing with textual data.

- We've been able to bump up the model performance upto 94%
- Making the dataset balanced, helped the model to be trained more accurately.
- Creating our own word embbeddings helped finding better representation of keywords of our corpus.
- Hyperparameter tuning resulted in finding the model with more accuracy without overfitting, which is evident from the train vs. validation accuracy curve.

5. COMPARISON TO BENCHMARK

Milestone 1 Benchmark:-

Model	Accuracy Score
Support Vector Machine - Linear	0.68
Deep Neural Network	0.64
Support Vector Machine - RBF	0.62
Random Forest	0.62
K Nearest Neighbor	0.61
Recurrent Neural Network with LSTM	0.57
Decision Tree	0.56
Recurrent Convolutional Neural Network	0.54
Convolutional Neural Network	0.53
Multinominal Naïve Bayes Classifier	0.52
Recurrent Neural Network	0.5

Based on comparison of Accuracy score of Multiple Models, Support Vector Machine was giving the highest Accuracy of 68% followed by Deep Neural Network with an accuracy of 64%.

Summary

Out of all the model architectures we've tried, the accuracy of each of the model is as follows in the table. Statistical models are overfitted to a higher degree. One obvious reason is the dataset is highly imbalanced. And Neural networks need to be fine tuned to increase accuracy. Following are some of the techniques we'll be trying in Milestone-2 as part of fine tuning.

- Dealing with imbalanced dataset.
    - Creating distinctive clusters under GRP_0 and downsampling top clusters
    - Clubbing together all those groups into one which has 30 or less tickets assigned

- Replacing TF-IDF vectorizer technique with word embeddings for statistical ML algorithms.
- Running GridSearchCV to perform hyper-parameter tuning.
- Altering intermediate layers in case of Newural networks
- Using Transfer learning

Milestone 2 Benchmark:-

Model - Parameter Tuning		
Model	Accuracy	Validaton Accuracy
Iteration 2 adding more core to LTSM	94.93%	92.46%
Iteration 3 adding a dense and dropout and batchNormalistaion layer	94.22%	92.41%
iteration 4 optimizing adam	94.15%	92.88%
Iteration 1 changing the dropout value	93.50%	93.08%
Base	90.36%	90.43%

We improved the benchmark from 68% to 95%, by addressing:-

- Dealing with imbalanced dataset.
    - Creating distinctive clusters under GRP_0 and down sampling top clusters
    - Clubbing together all those groups into one which has 25 or less tickets assigned
- Replacing TF-IDF vectorizer technique with word embeddings.

6. VISUALIZATIONS

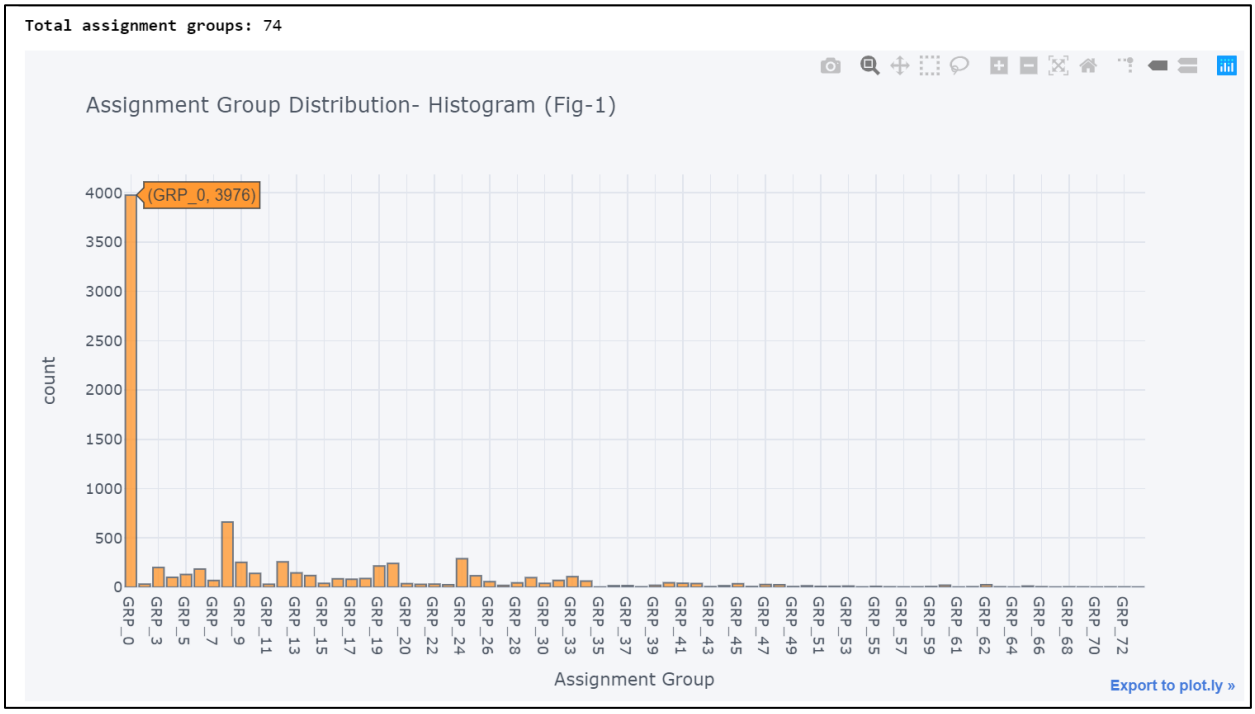
6.1. FINDING INCONSISTENCIES IN ASSIGNMENT GROUP

Checking the distribution of 8500 Incidents across 74 Assignment groups (Target Feature).

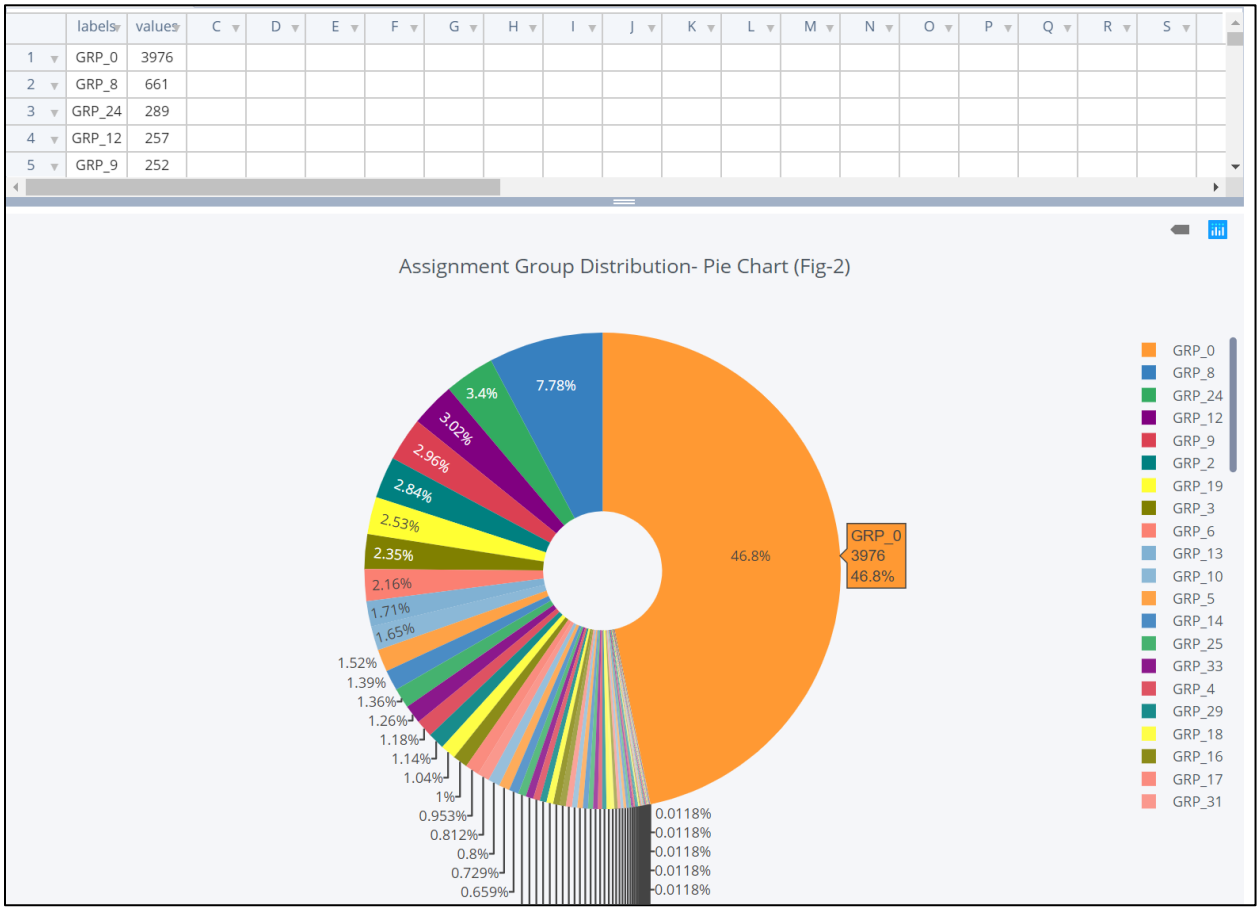
Following Plots display how the Assignments Groups are scattered across the dataset.

The bar chart, histogram and pie chart tells the frequency of any ticket assigned to any group OR the tickets count for each group.

6.1.1. Assignment Group Distribution- Histogram:-

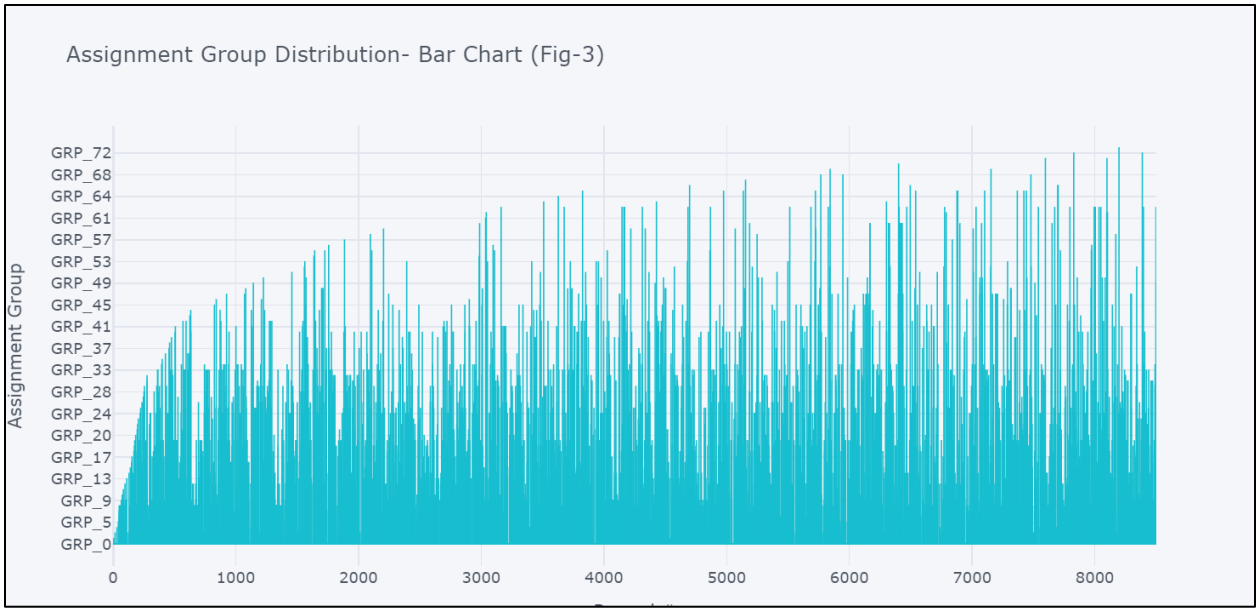


6.1.2. Assignment Group Distribution- Pie Chart:-



	labels	values	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	GRP_0	3976																	
2	GRP_8	661																	
3	GRP_24	289																	
4	GRP_12	257																	
5	GRP_9	252																	

6.1.3. Assignment Group Distribution- Bar Chart:-



Central Limit Theorem

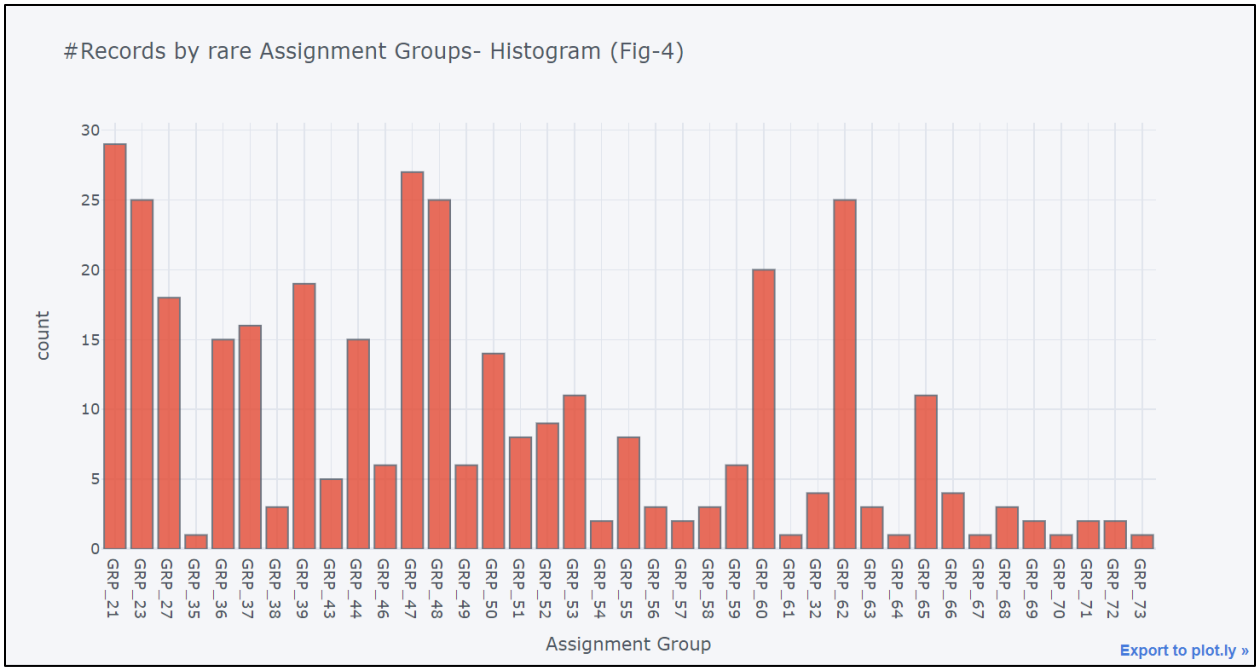
The Central Limit Theorem states that the sampling distribution of the sample means approaches a normal distribution as the sample size gets larger — no matter what the shape of the population distribution. This fact holds especially true and practitioners have been presumably considered it for sample sizes over 30. All this is saying is that as you take more samples, especially large ones, your graph of the sample means will look more like a normal distribution.

Hence let's identify such Assignment groups which doesn't have atleast 30 tickets assigned to them and categorize them as rare group.

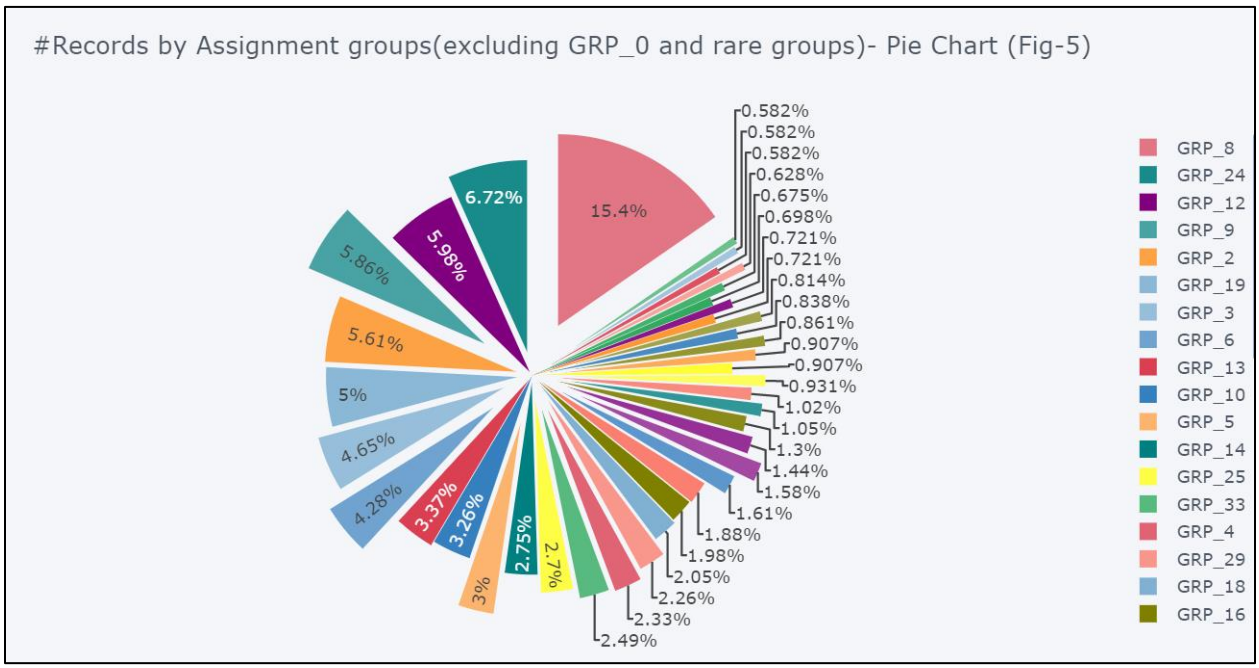
Find out the Assignment Groups with less than 30 tickets assigned:-

6.1.4. **Incident Distribution across 40 Assignment Groups:-**

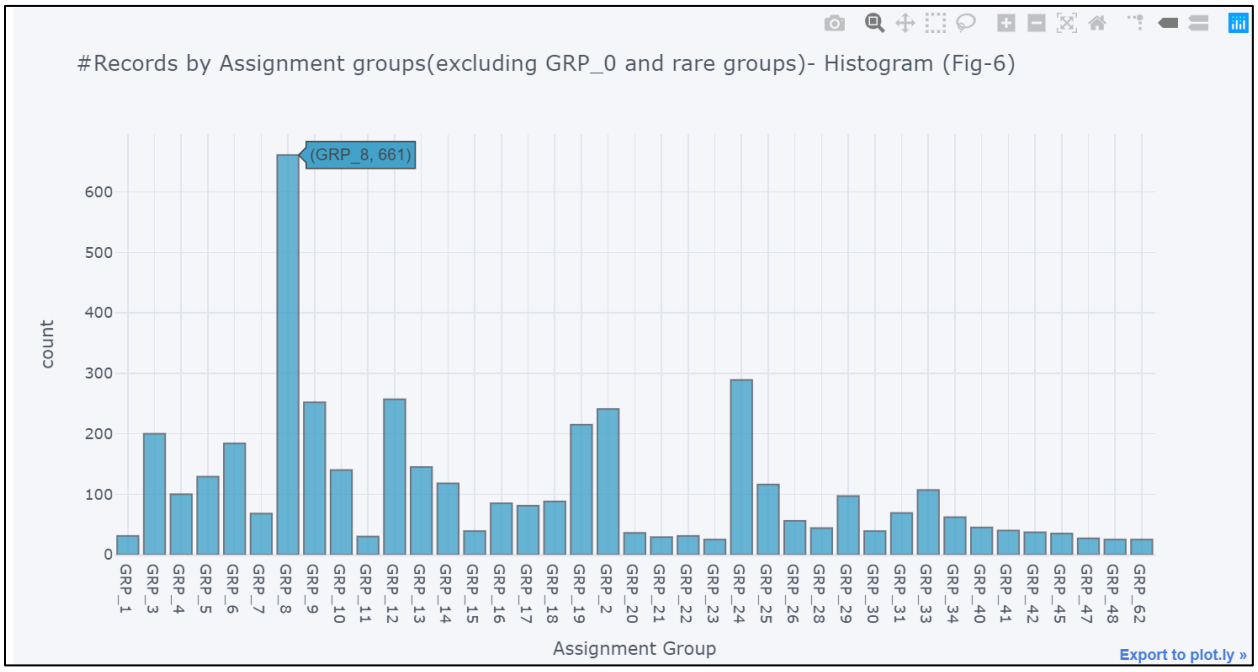
Distribution of Assignment groups excluding GRP_0 & rare groups (groups with 30 or less tickets assigned):-



6.1.5. Ticket Distribution Pie Chart (Excluding GRP_0 and 40 rare Assignment Groups):-



6.1.6. Ticket Distribution Bar Chart (Excluding GRP_0 and 40 rare Assignment Groups):-

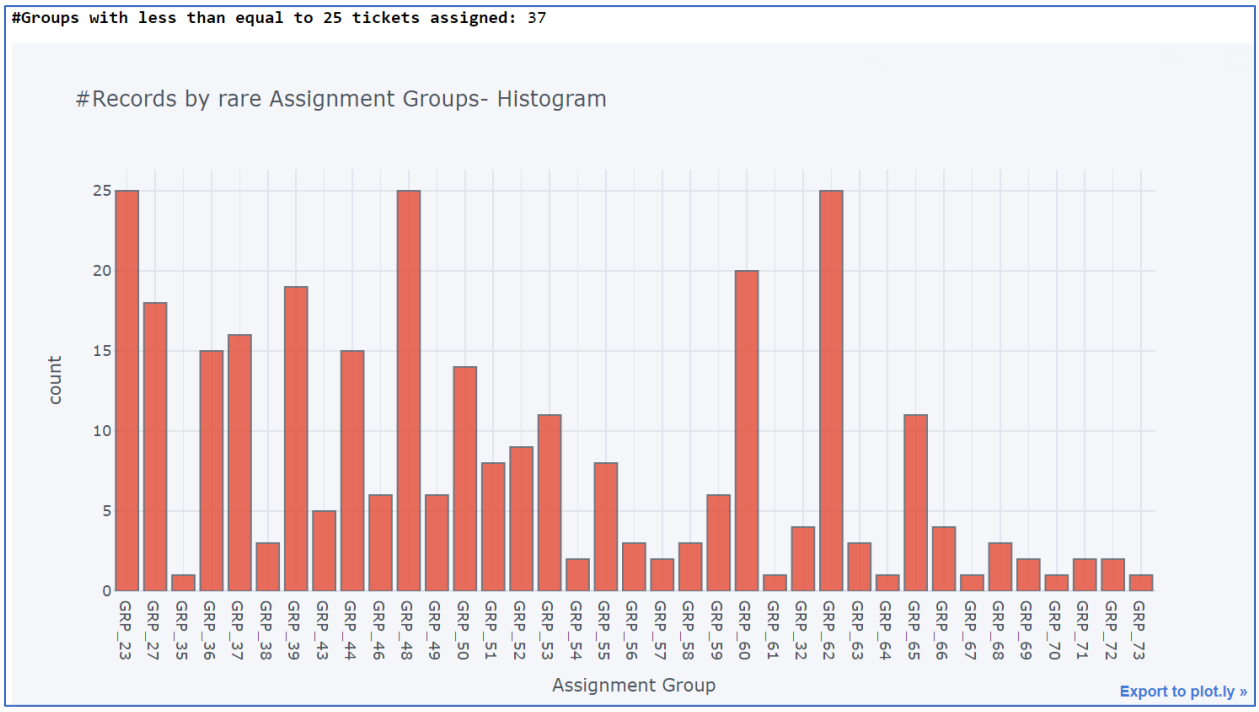


Observations:

- The bottom section of bar chart (Fig-3) is dense enough to indicate the GRP_0 is the most assigned group. So is also proved from the Histogram (Fig-1) and the pie chart (Fig-2) that tickets assigned to GRP_0 is close to 4000 out of 8500 records, which is 46.8% in volume.
- The 2nd highest assignment group is GRP_8, which is just 7.78% of the total dataset and [1/6]th of the GRP_0
- There are 40 groups with just 30 or less tickets assigned (Fig-4), amongst which 6 groups happened to be assigned with just 1 ticket and 4 groups with just 2 tickets each.
- As Assignment Group attribute is the target column in our dataset, these tickets distribution shows the dataset is highly imbalanced.

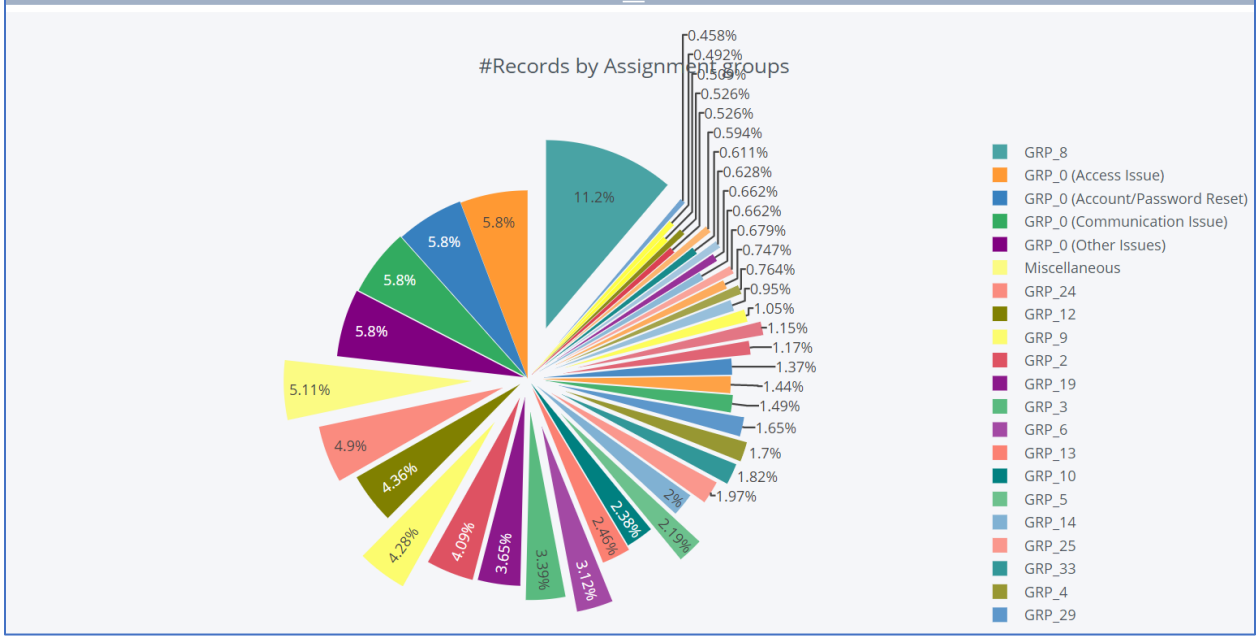
6.1.7. Ticket Distribution Histogram (For Assignment Groups with 25 or less tickets):-

```
#Groups with less than equal to 25 tickets assigned: 37
```



6.1.8. Ticket Distribution Pie Chart (Balanced Dataset of Assignment Groups):-

	labels	pull	values	D	E	F	G	H	I	J	K	L	M	N	O
1	GRP_0 (Access Issue)	0	342												
2	GRP_0 (Account/Password Reset)	0.0075	342												
3	GRP_0 (Communication Issue)	0.015	342												
4	GRP_0 (Other Issues)	0.0225	342												
5	GRP_1	0.03	31												

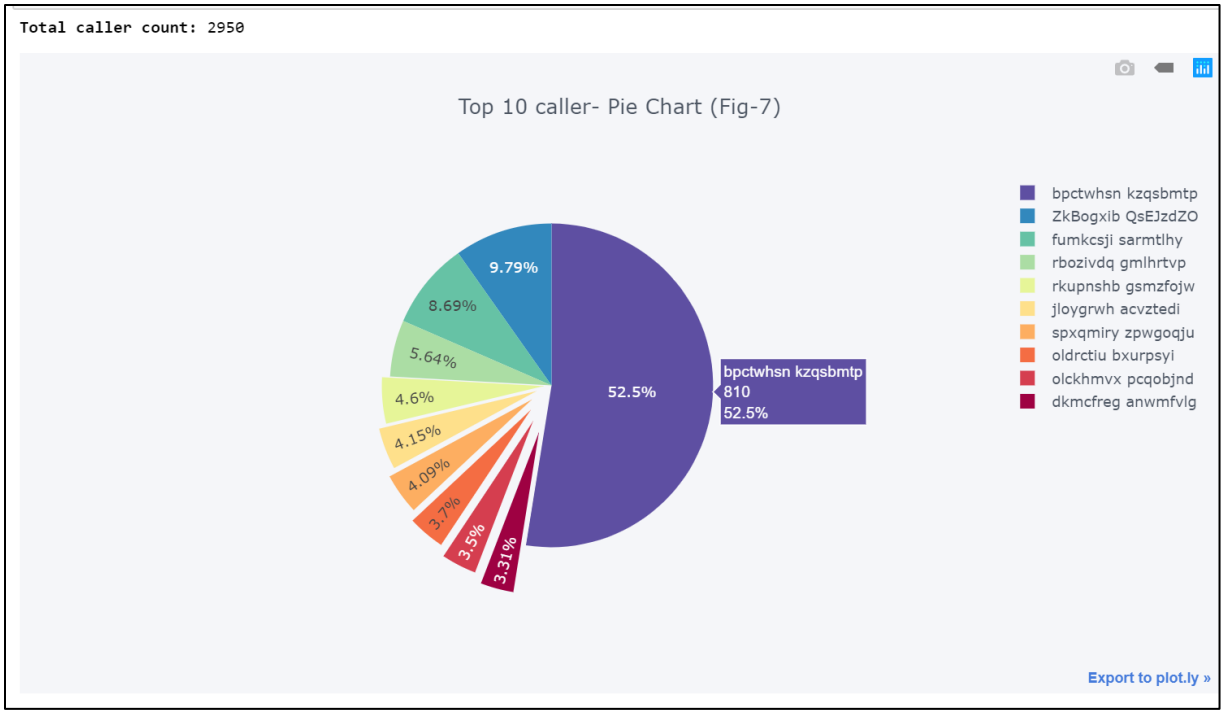


6.2. VISUALIZATION OF CALLER PATTERNS

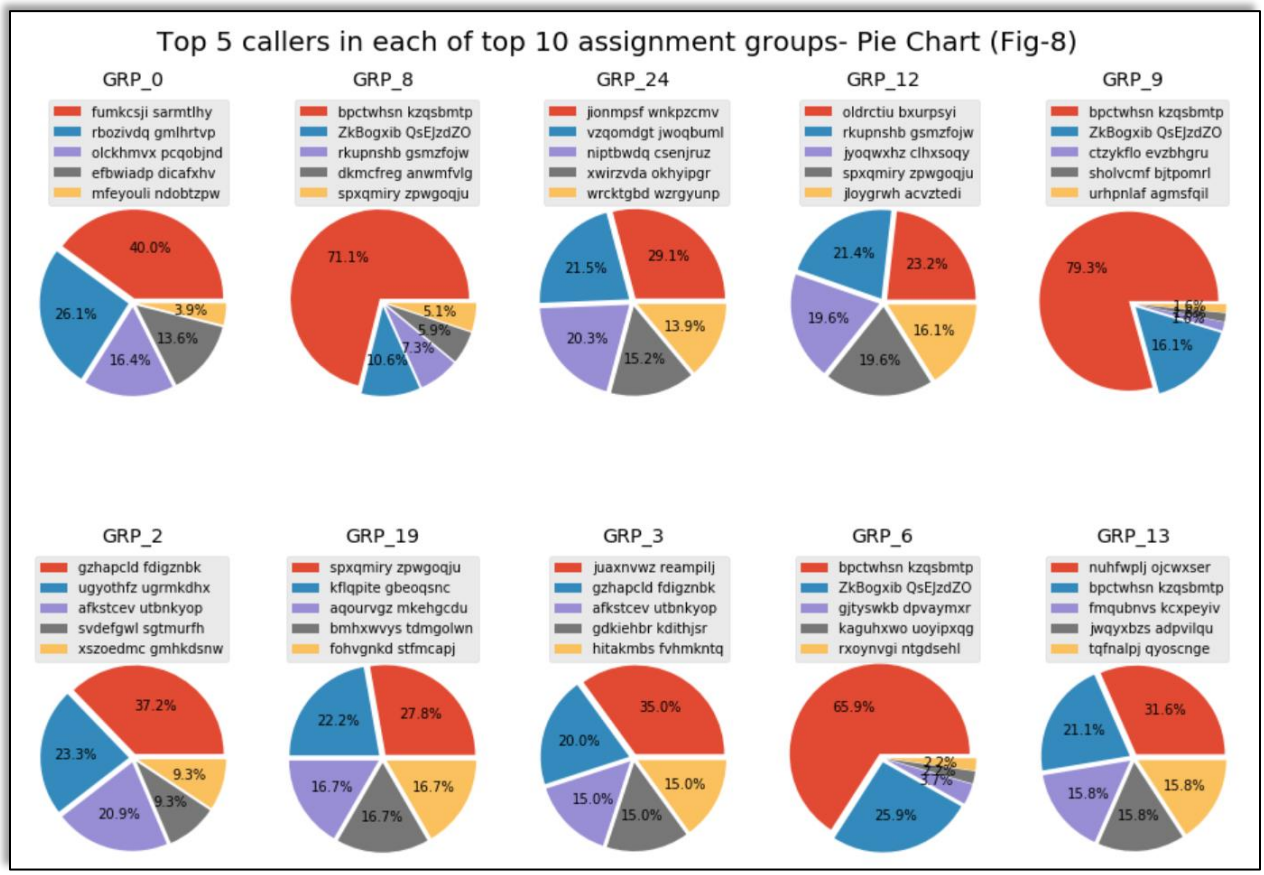
The distribution of Callers:-

Plots how the callers are associated with tickets and what are the assignment groups they frequently raise tickets for.

6.2.1. Find out top 10 callers in terms of frequency in the entire dataset:-



6.2.2. Visualizing Top 5 callers in each of the Top 10 Assignment Groups:-



6.2.3. Check if any caller raise Incident for multiple Assignment groups

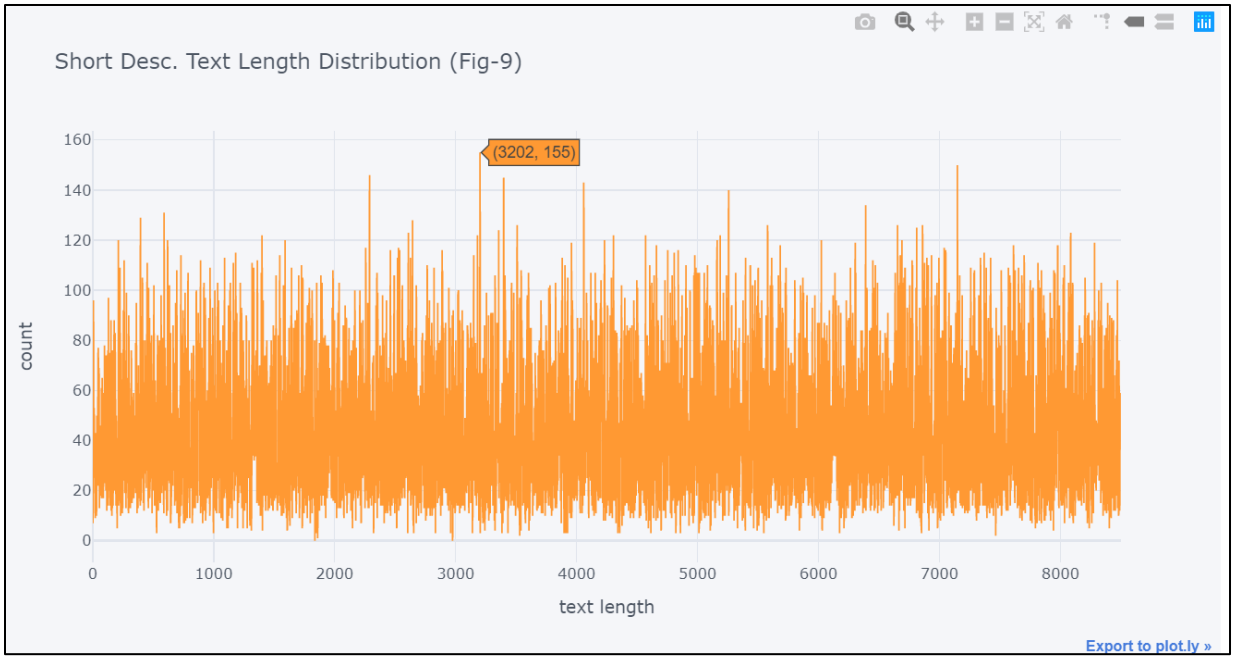
Observations:-

- There are 2950 callers have been captured in the entire dataset.
- 15 callers are identified to be involved in raising tickets for multiple assignment groups, there by contributing total 281 tickets in the dataset.
- The analysis of top 5 callers volume in each of the top 10 assignment groups shows a wide variety of distributions in each pie chart. This is an indicative of the fact that each assignment group deals with issues impacting the business in NOT biased proportions.

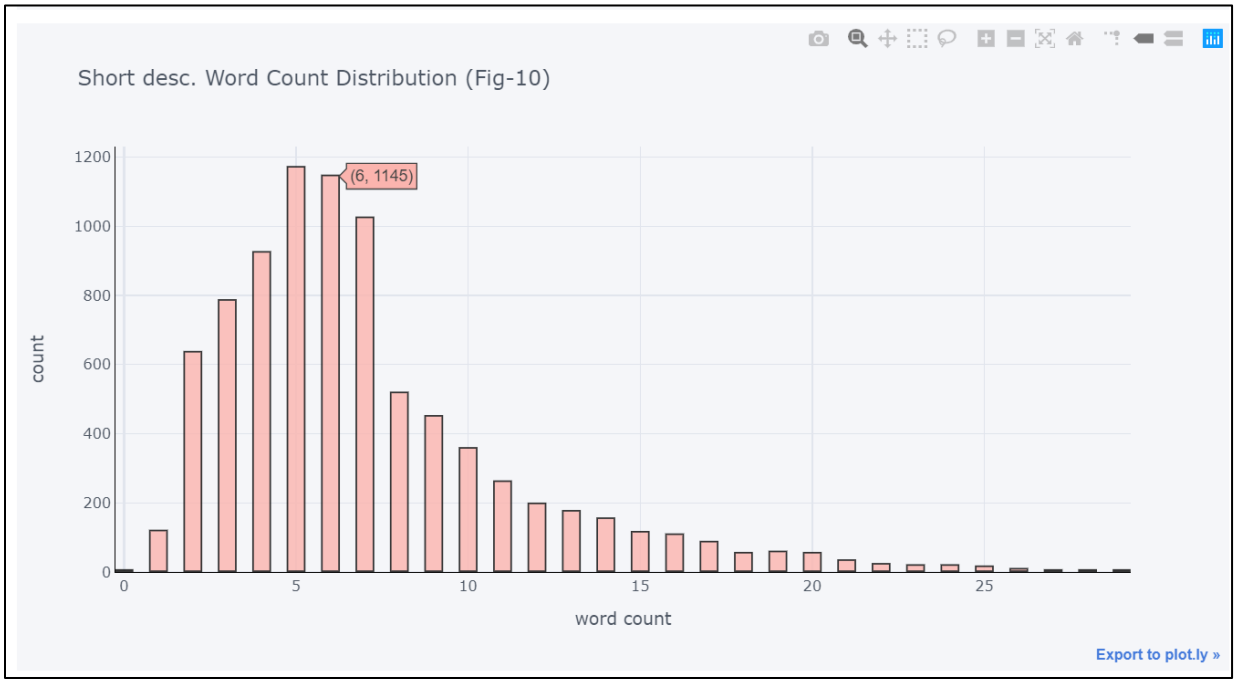
6.3. VISUALIZATION OF DIFFERENT TEXT PATTERNS

6.3.1. Distribution of Short description length

6.3.1.1. Visualize variation of length of Short description attribute



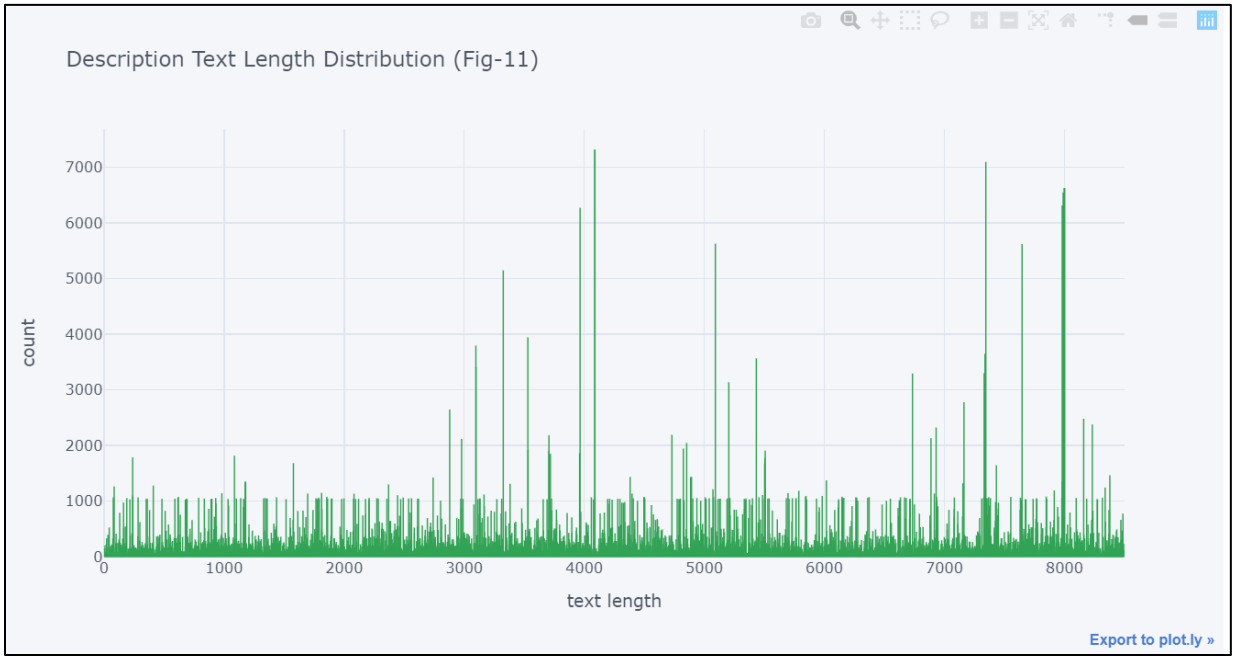
6.3.1.2. Visualize variation of word count of Short description attribute



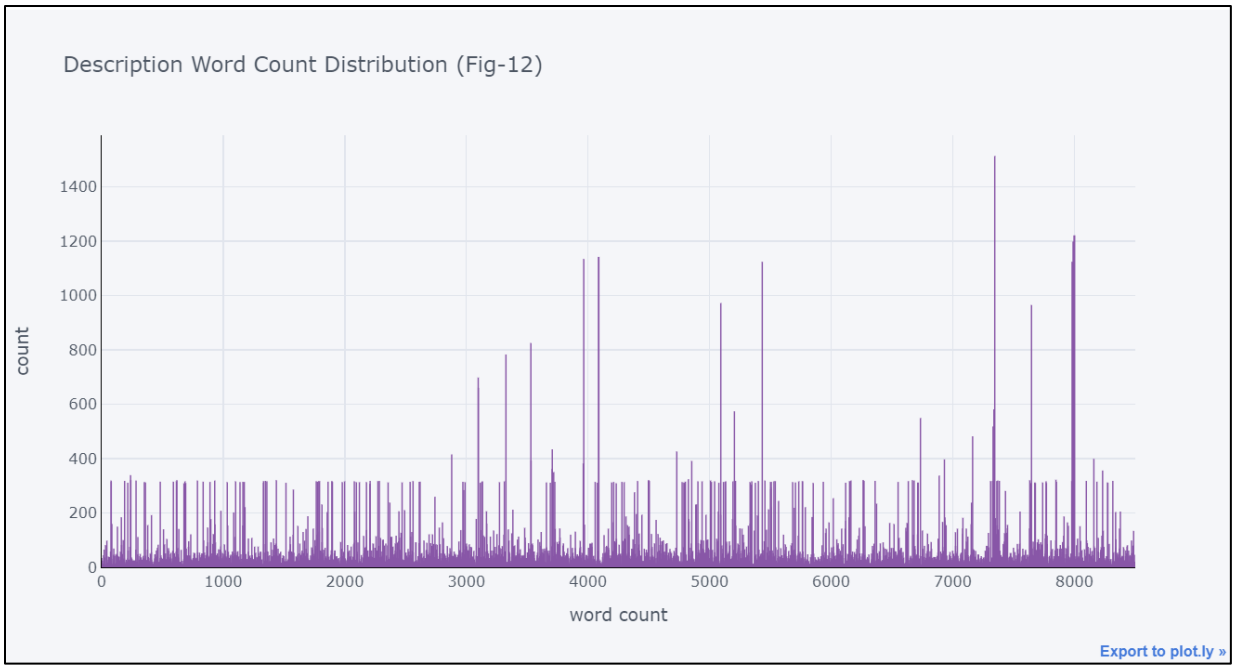
6.3.2. Distribution of Description length

Plots the variation of length and word count of Description attribute:-

6.3.2.1. Visualize variation of length of Description attribute



6.3.2.2. Visualize variation of word count of Description attribute



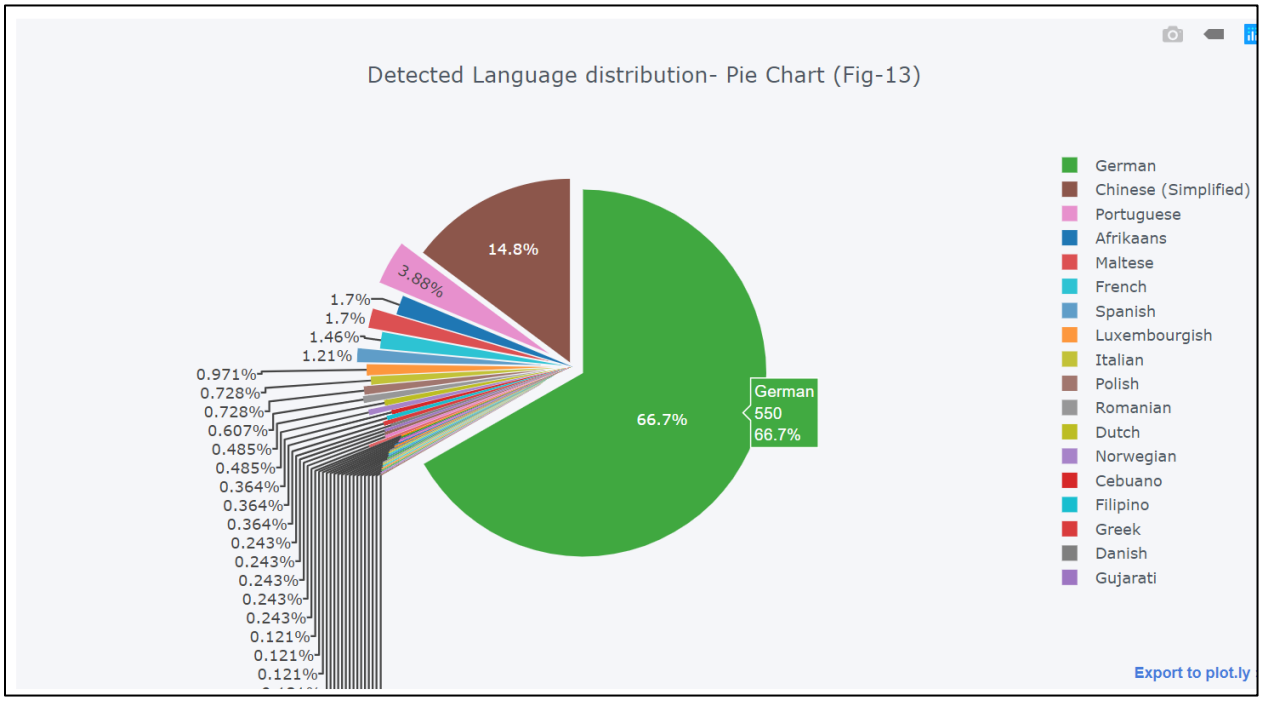
Observation:-

Users preferred to leave long descriptions.

6.3.3. The distribution of Language

Plots the variation of tickets received in different languages:-

6.3.3.1. Visualize Incident distribution across non-English languages



6.3.4. N-Grams

N-gram is a contiguous sequence of N items from a given sample of text or speech, in the fields of computational linguistics and probability. The items can be phonemes, syllables, letters, words or base pairs according to the application. N-grams are used to describe the number of words used as observation points, e.g., unigram means singly-worded, bigram means 2-worded phrase, and trigram means 3-worded phrase.

We'll be using scikit-learn's CountVectorizer function to derive n-grams and compare them before and after removing stop words. Stop words are a set of commonly used words in any language. We'll be using english corpus stopwords and extend it to include some business specific common words considered to be stop words in our case.

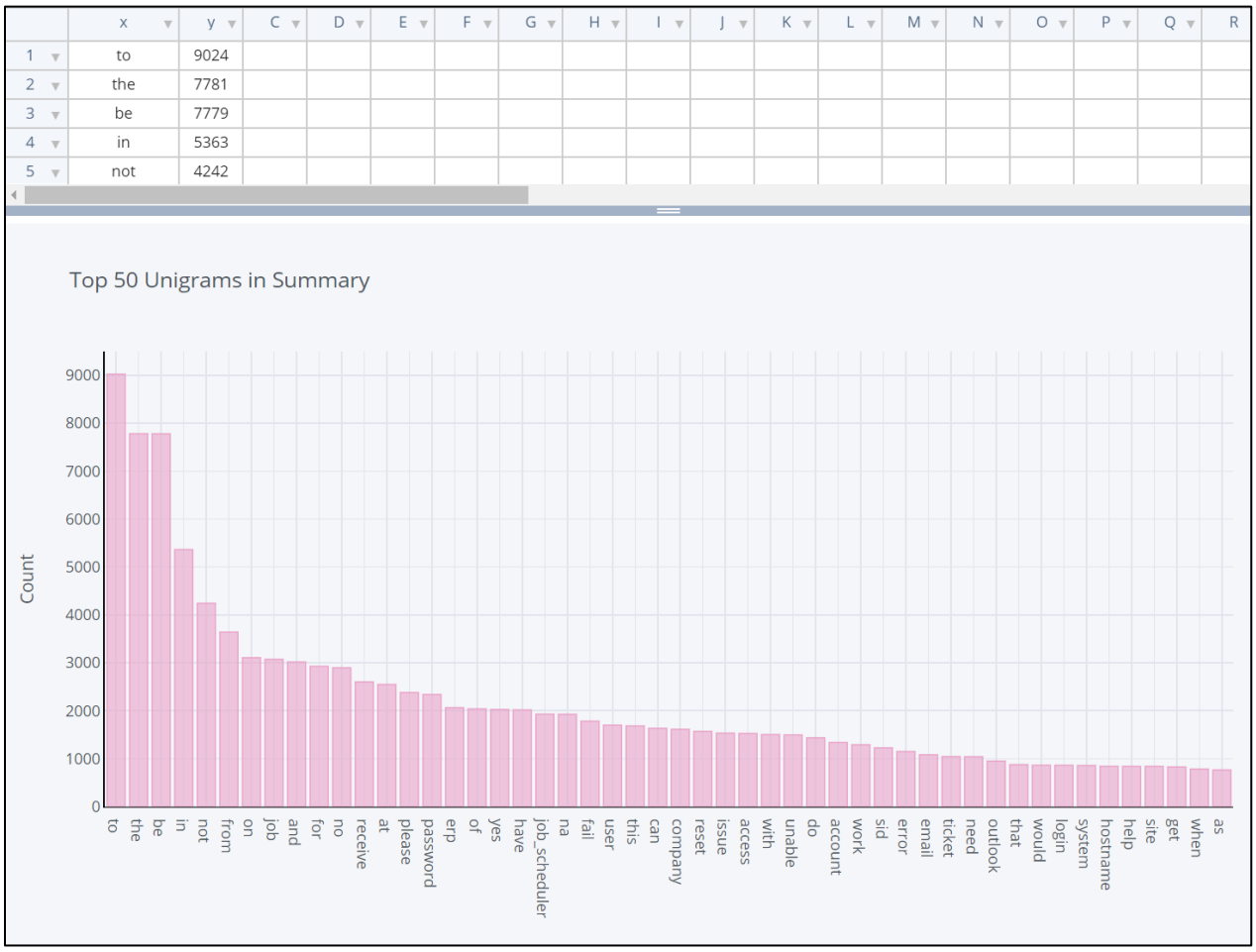
But before that let's merge the Short description and Description column texts and write a generic method to derive the n-grams.

I. Merge the Short description and Description column texts.

6.3.4.1. Top 50 Unigrams before removing Stop Words:-

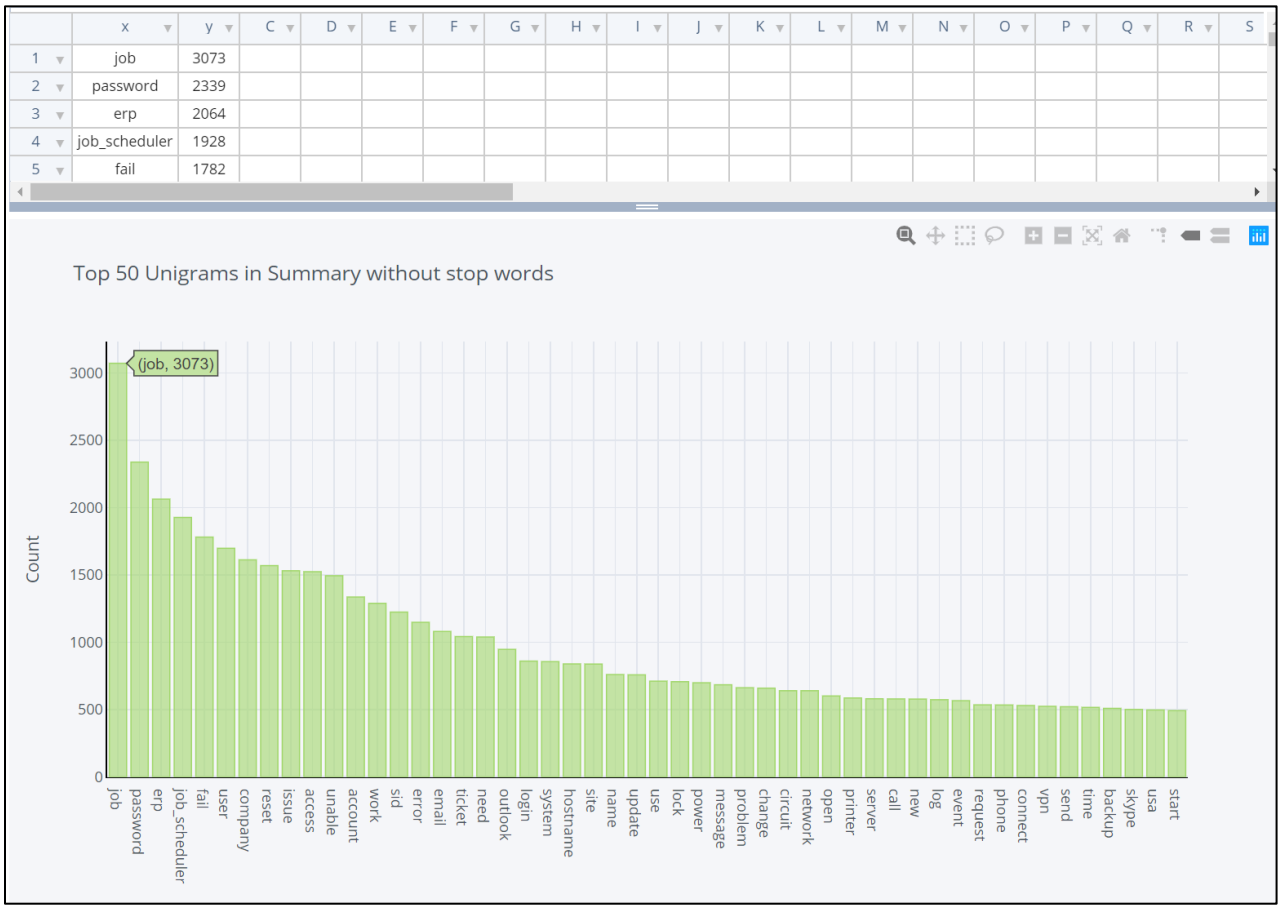
Summary = Short description + Description

	x	y	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	to	9024																
2	the	7781																
3	be	7779																
4	in	5363																
5	not	4242																



6.3.4.2. Top 50 Unigrams after removing Stop Words:-

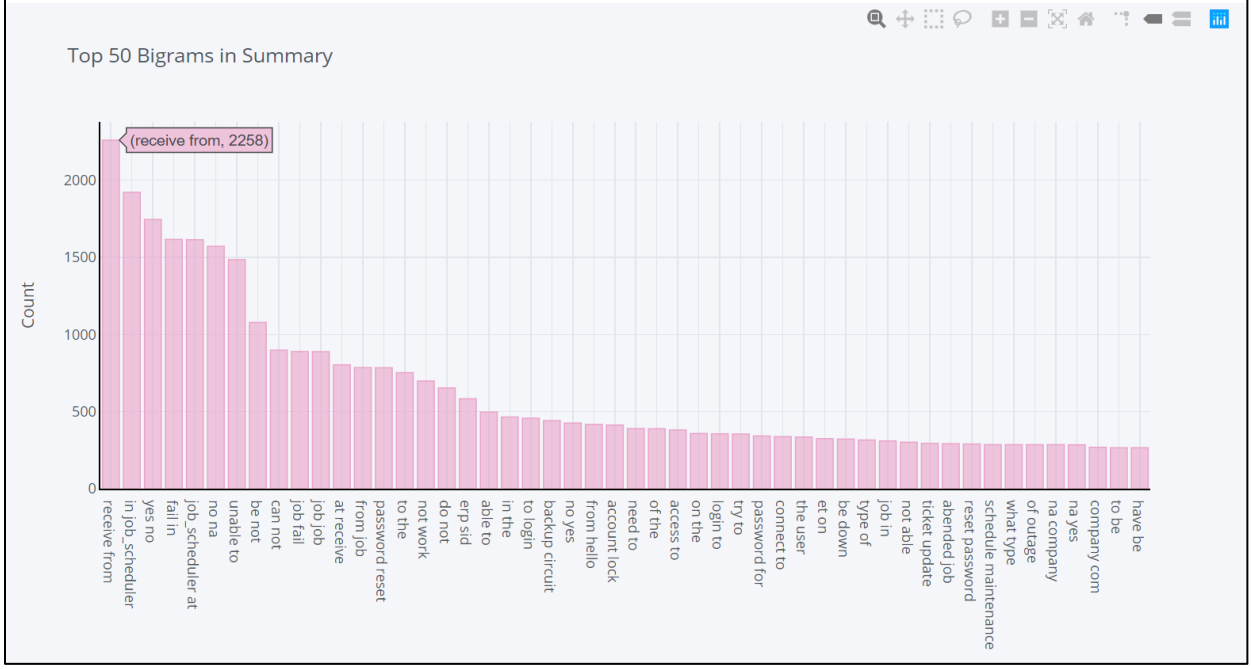
Summary = Short description + Description



6.3.4.3. Top 50 Bigrams before removing Stop Words:-

Summary = Short description + Description

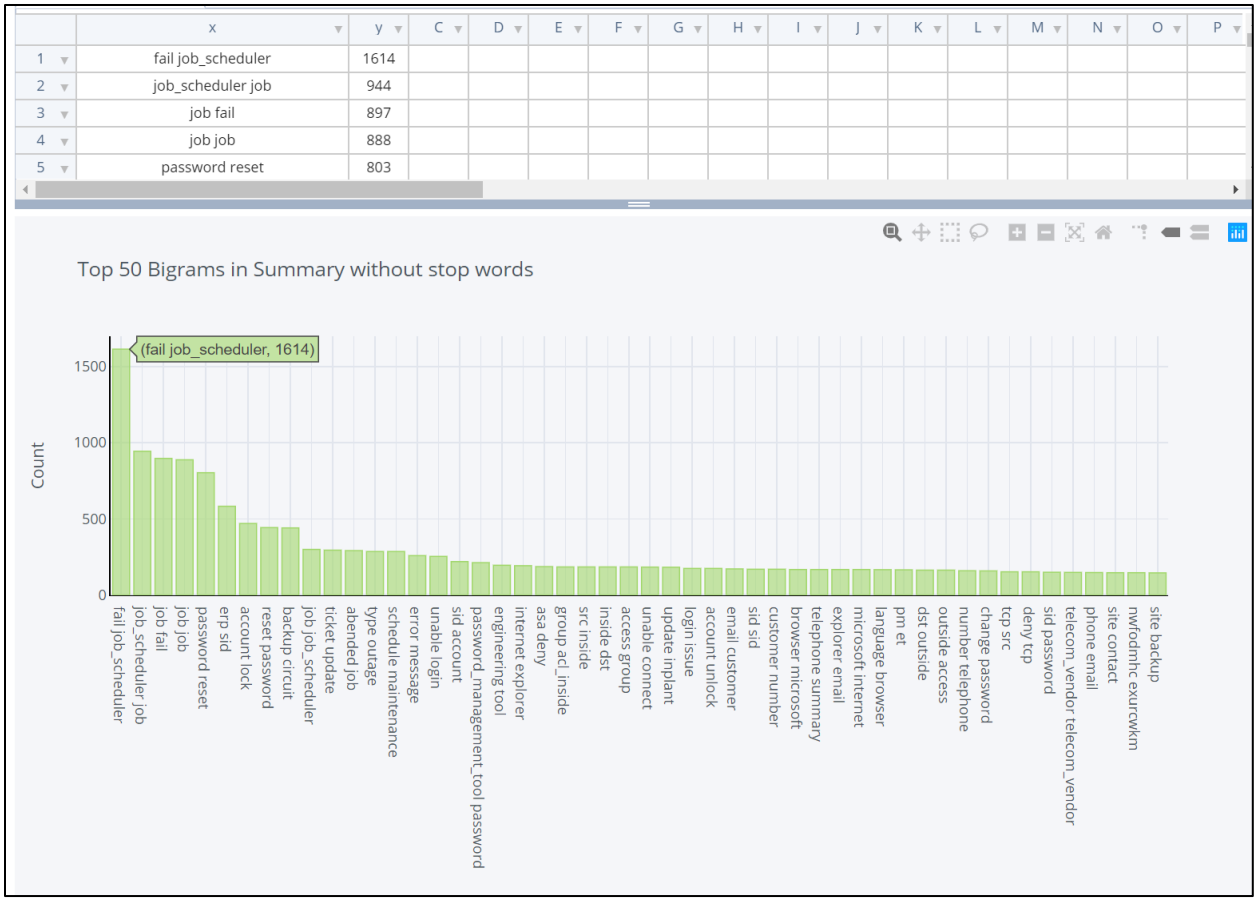
	x	y	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	receive from	2258																
2	in job_scheduler	1920																
3	yes no	1744																
4	fail in	1615																
5	job_scheduler at	1613																



6.3.4.4. Top 50 Bigrams after removing Stop Words:-

Summary = Short description + Description

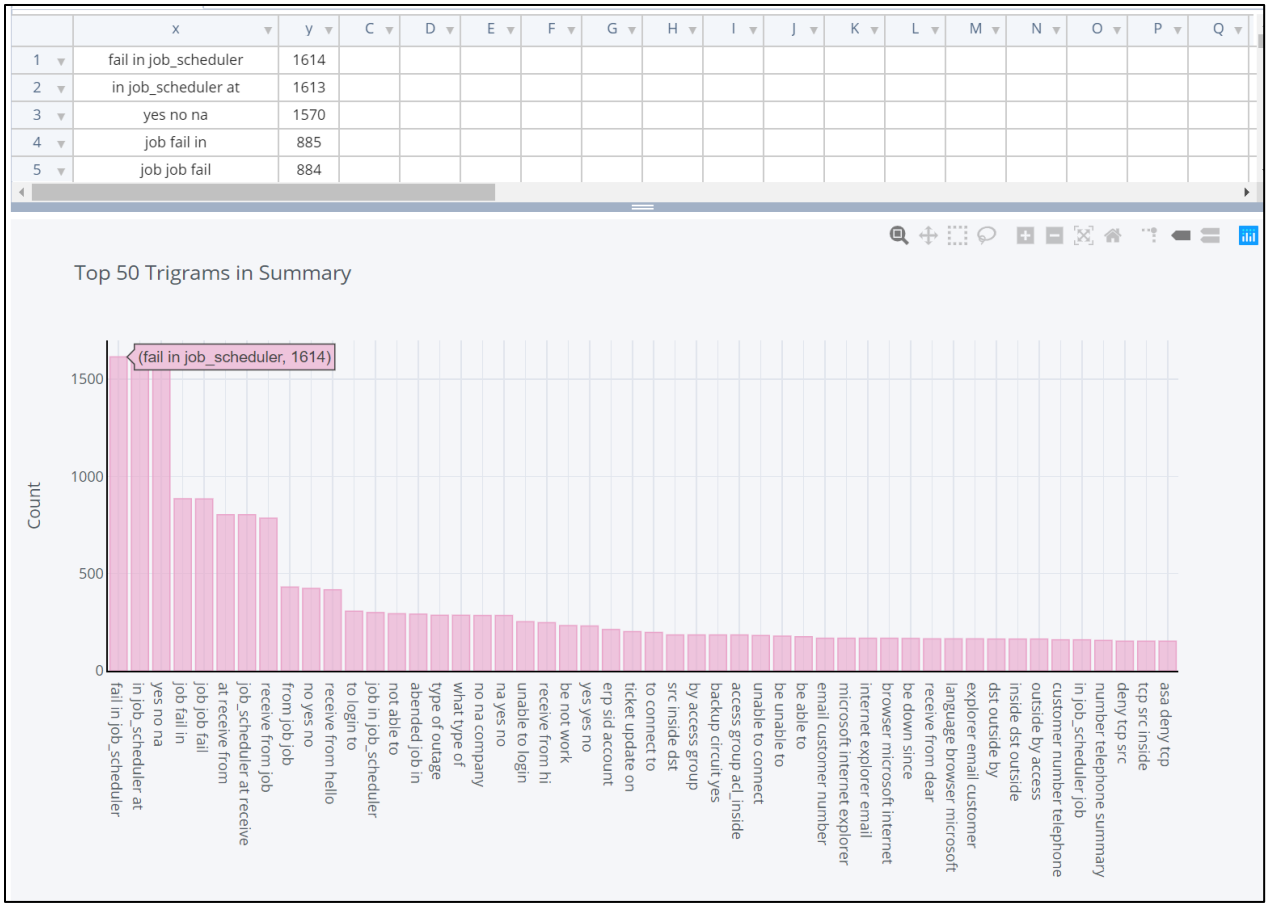
	x	y	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	fail job_scheduler	1614														
2	job_scheduler job	944														
3	job fail	897														
4	job job	888														
5	password reset	803														



6.3.4.5. Top 50 Trigrams before removing Stop Words:-

Summary = Short description + Description

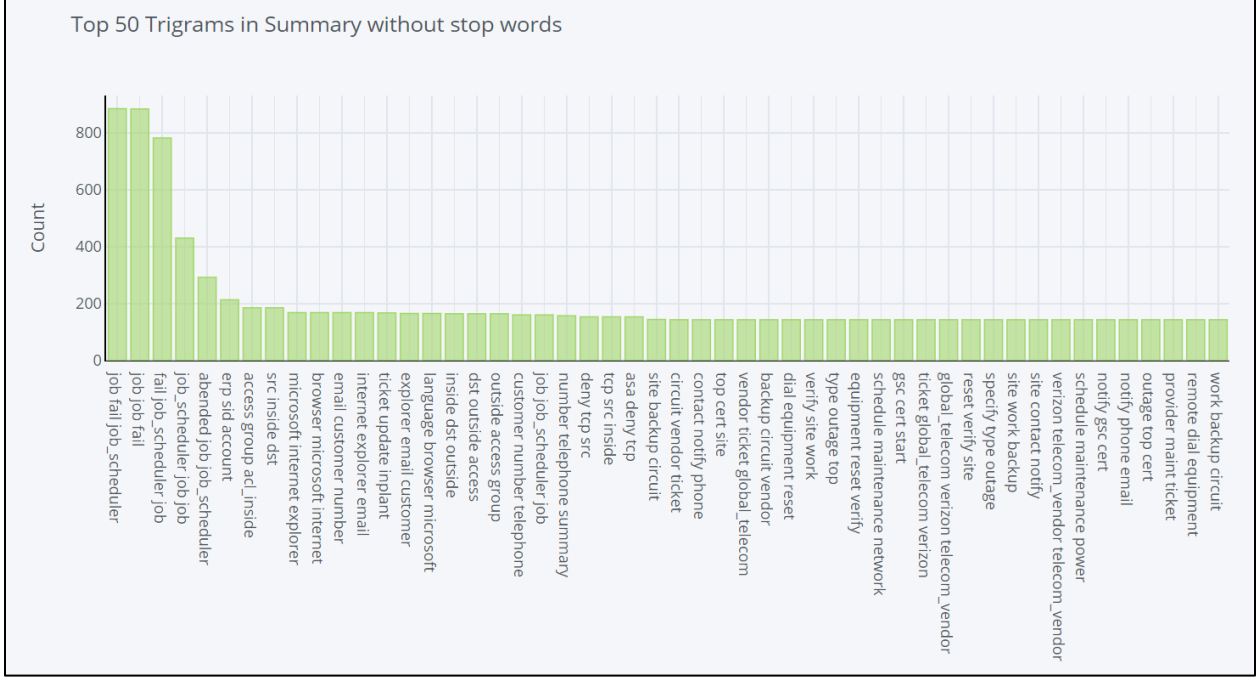
	x	y	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	fail in job_scheduler	1614															
2	in job_scheduler at	1613															
3	yes no na	1570															
4	job fail in	885															
5	job job fail	884															



6.3.4.6. Top 50 Trigrams after removing Stop Words:-

Summary = Short description + Description

	x	y	C	D	E	F	G	H	I	J	K	L	M	N	O
1	job fail job_scheduler	885													
2	job job fail	884													
3	fail job_scheduler job	782													
4	job_scheduler job job	430													
5	abended job job_scheduler	292													



Observations:-

- It's indicative from the n-gram analysis is that the entire dataset speaks more about issues around
  - **password reset (1246 times)**
  - **fail job_scheduler (1614 times)**
  - **outlook (948 times)**
  - **login (861 times)**
  - **job fail (897 times)**

6.3.5. Word Cloud

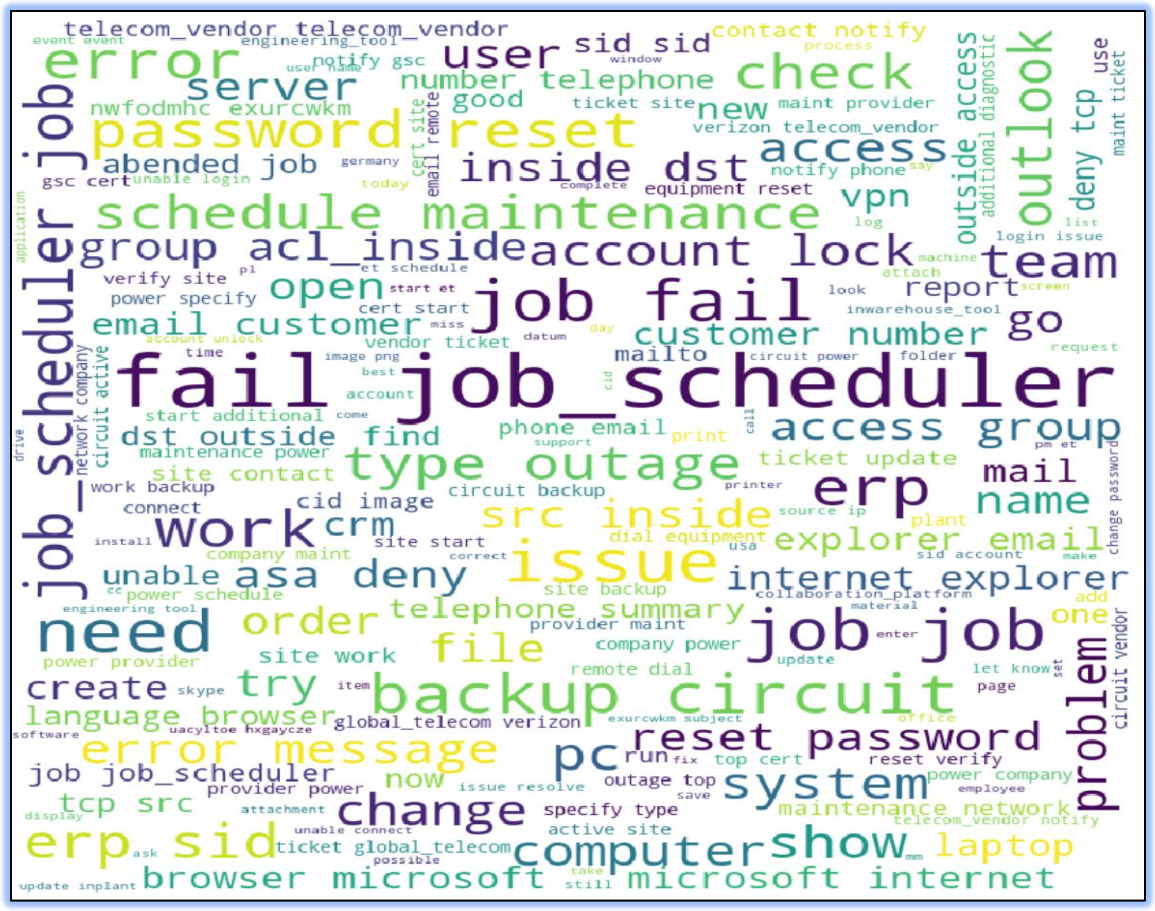
A word cloud is a collection, or cluster, of words depicted in different sizes. The bigger and bolder the word appears, the more often it's mentioned within a given text and the more important it is.

Also known as tag clouds or text clouds, these are ideal ways to pull out the most pertinent parts of textual data, often also help business users compare and contrast two different pieces of text to find the wording similarities between the two.

I. Generic method to generate Word Clouds for both Short and Long Description columns.

6.3.5.1. Word Cloud for ticket Short Description: -

6.3.5.2. Word Cloud for Ticket Description: -



6.3.5.3. Word Cloud for combined Ticket Description and Short Description: -

Observations: -

1. Failure of job_scheduler and job failure have significant contribution.

Root cause analysis (RCA) need to be performed to further analyze and fix problem jobs.

No. of Incident Ticket reduction expected by performing RCA:- 1928

2. Password Rest process need to be automated.

No. of Incident Ticket reduction expected by automating password reset:- 1246

6.3.5.4. Word Cloud on combined Ticket Description and Short Description for Group 0: -

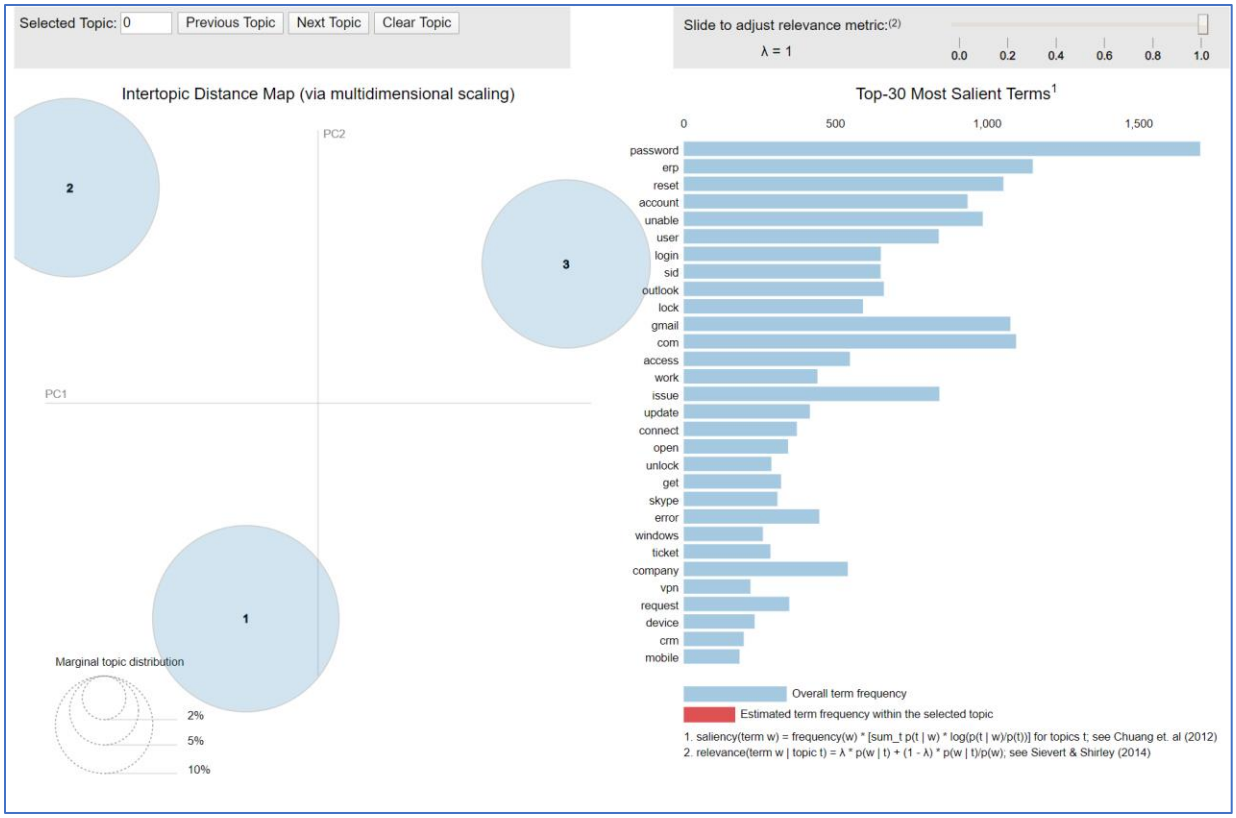
Observations: -

- Analysis on GRP_0 which is the most frequent group to assign a ticket to reveals that this group deals with mostly the maintenance problems such as *password reset*, *account lock*, *login issue*, *ticket update* etc.
- Maximum of the tickets from GRP_0 can be reduced by self correcting itself by putting automation scripts/mechanisms to help resolve these common maintenance issues. This will help in lowering the inflow of service tickets thereby saving the person/hour efforts spend and increasing the business revenue.

6.4. VISUALIZATION OF TOPICS-KEYWORDS

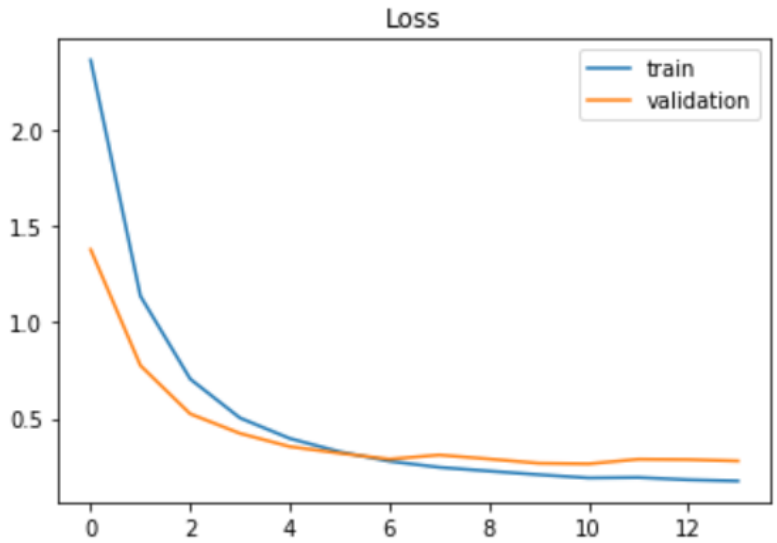
Examine the produced topics and the associated keywords using pyLDAvis.

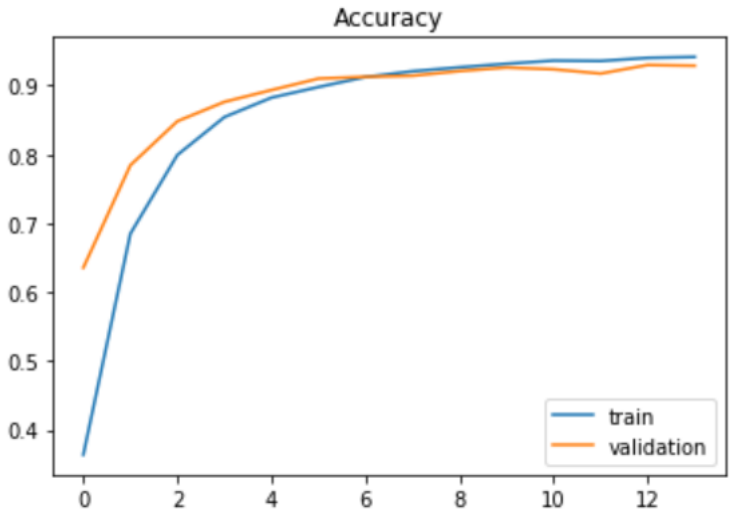
6.4.1. Visualize the topics and associated keywords



6.5. VISUALIZATION OF MODEL ACCURACY & LOSS

Plotting Train v/s Validation performance in Terms of Accuracy and Loss for the final RNN with LSTM model.





7. IMPLICATIONS

7.1. Automation of Ticket Assignment has following benefits: -

1. Increase in Customer Satisfaction.
2. Decrease in the response and resolution time.
3. Eliminate human error in Ticket Assignment. (Which was ~25% Incidents)
4. Avoid missing SLAs due to error in Ticket Assignment.
5. Eliminate any Financial penalty associated with missed SLAs.
6. Excellent Customer Service.
7. Reallocate (~1 FTE) requirement for Productive Work.
8. Increase in morale of L1 / L2 Team.
9. Eradicate 15 mins Effort spent for SOP review (~25-30% of Incidents OR 531.25-637.5 Person Hours).
10. Decrease in associated Expense.
11. L1 / L2 Team can focus on resolving ~54% of the incidents
12. Functional / L3 teams can focus on resolving ~56% of incidents

~1 FTE from L1 / L2 Team saved through automating Ticket Assignment can focus on Continuous Improvement activities.

~25% of Incidents which is 2125 additional Incidents will now get resolved within SLA.

7.2. Confidence Interval

There is a 95% likelihood that the confidence interval [92.21, 93.31] covers the true classification of the model on unseen data.

7.3. Additional Business Insights and Savings Suggestions from Ticket Analysis: -

1. Root cause analysis (RCA) need to be performed on job_scheduler, to understand the cause of failure.

No. of Incident Ticket reduction expected by performing RCA:- 1928.

22.68% of Total Incident volume of 8500.

Hence, we can reduce the Resource / FTE allocation also by approximately 22.68%.

2. Password Rest process need to be automated.

No. of Incident Ticket reduction expected by automating password reset process:- 1246

14.66% of Total Incident volume of 8500.

Hence, we can reduce the Resource / FTE allocation also by approximately 14.66%.

Hence a cumulative reduction of 3174 Incidents means 37.34% reduction in Total Incident volume of 8500.

Hence, cumulative Resource / FTE allocation reduction by approximately 37.34%.

Business can operate at ~62.66% of original Estimates.

8. LIMITATIONS

- Since the word embedding were generated of given text description, it is possible over the time that the model may not be able to recognize new words. In that case, a weekly model regeneration should be kept until the word recognition is stable with 90% text coverage.
- While checking the distribution of 8500 Incidents across 74 Assignment groups, it was noticed that 37 Assignment Groups had 25 or less tickets, we had to clubbed the 37 Assignment Groups into a Category Miscellaneous. Manual Assignment will be required for the tickets belonging to these 40 Assignment Groups.

9. CLOSING REFLECTIONS

Support Vector Machine was giving the highest Accuracy of 68% during Milestone-1.

Observation from Milestone-1

Out of all the models we've tried in Milestone-1, Support Vector Machine (SVM) under statistical ML algorithms and Neural Networks are performing better than all others. The models were highly overfitted and one of the obvious reason was the dataset was highly imbalanced. Ratio of GRP_0 to all others is 47:53 and there are 40 groups having less than or equal to 30 tickets assigned each.

We have addressed this problem to fine tune the model accuracy by:-

- Dealing with imbalanced dataset.
    - Creating distinctive clusters under GRP_0 and down sampling top clusters
    - Clubbing together all those groups into one which has 25 or less tickets assigned
- Replacing TF-IDF vectorizer technique with word embeddings.

We could successfully increase the Accuracy to 95%.