AUTOMATIC TICKET ASSIGNMENT – INTERIM REPORT

February 2020

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

CONTENTS

INTRODUCTION

ABSTRACT

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.
7. Decrease in morale of Support Resource.
8. Increase in Expenses.
9. Call quality could impact pertinent data capture

OBJECTIVE

Build Multi-Class classifier that can classify the tickets by analyzing text.

Guided by powerful AI techniques that can classify incidents to right functional groups can help organizations to reduce the resolving time of the issue and can focus on more productive tasks.

MILESTONES

1. Milestone 1: 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 2: 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

APPROACH/DESIGN

PACKAGE INSTALLATIONS & CONFIGURATIONS

Import all necessary packages. Install the libraries which are not included in Anaconda distribution by default using pypi channel or conda forge **!pip install ftfy wordcloud goslate spacy**
conda install -c conda-forge ftfy wordcloud goslate spacy

```
# Utilities
from time import time
from PIL import Image
from zipfile import ZipFile
import os, sys, itertools, re
import warnings, pickle, string
from ftfy import fix_encoding, fix_text, badness
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator

# Translation APIs
from goslate import Goslate # Provided by Google

# Numerical calculation
import numpy as np

# Data Handling
import pandas as pd

# Data Visualization
import matplotlib.pyplot as plt
import seaborn as sns
import cufflinks as cf
import plotly as py
import plotly.graph_objs as go
from plotly.offline import download_plotlyjs,init_notebook_mode,plot,iplot

# Sequential Modeling
import keras.backend as K
from keras.datasets import imdb
from keras.models import Sequential
from keras.layers import Dropout, Flatten, Dense, Embedding, LSTM, TimeDistributed, Conv1D, MaxPooling1D
from keras.constraints import max_norm, unit_norm
from keras.preprocessing.text import Tokenizer, text_to_word_sequence
from keras.preprocessing.sequence import pad_sequences

# Tools & Evaluation metrics
from sklearn.metrics import confusion_matrix, classification_report, auc
from sklearn.metrics import roc_curve, accuracy_score, precision_recall_curve
from sklearn.feature_extraction.text import CountVectorizer
from imblearn.over_sampling import RandomOverSampler

# NLP toolkits
import spacy
import nltk
from nltk import tokenize
```

```
# Configure for any default setting of any library
warnings.filterwarnings('ignore')
get_ipython().magic(u'matplotlib inline')
plt.style.use('ggplot')
init_notebook_mode(connected=True)
cf.go_offline()
%matplotlib inline
```

- Extract Glove 6Billion word embeddings. We're going to use the 200d file which has 200 embedding dimensions for each word in the corpus.
- Load the bodies and headlines CSVs for both training and test data and join them together individually to form datasets.

LIBRARIES

Following are the external libraries used for the assignment :

Library	Purpose	Remarks
Numpy	Numerical Calculation	
Pandas	Data Handling	
Plotly	Data Visualization	
Seaborn	Data Visualization	
Keras	Sequential Modelling	
Sklearn	Tools & Evaluation metrics	
NLTK	NLP Toolkit	
Mojibake	Text Pre-Processing	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

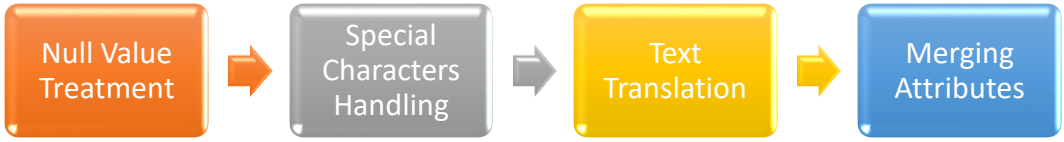
		that’s broken in various ways. The goal of ftfy is to take in bad Unicode and output good Unicode. Installation: using pypi: !pip install ftfy using conda: conda install -c conda-forge ftfy
Language Translation (Goslate: Free Google Translate API)	Text Pre-Processing	Goslate is an open source python 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 selected 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 Service 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

DESIGN

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)
- Model Selection

Data Clean-up Sub flow :



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

PRE-PROCESSING, DATA VISUALIZATION AND EDA

LOADING & EXPLORING THE DATASET

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.

```
# Block which runs on both Google Colab and Local PC without any modification
if 'google.colab' in sys.modules:
    project_path = "/content/drive/My Drive/Colab Notebooks/DLCP/Capstone-NLP/"
    # Google Colab lib
    from google.colab import drive
    # Mount the drive
    drive.mount('/content/drive/', force_remount=True)
    sys.path.append(project_path)
    %cd $project_path

# Let's look at the sys path
print('Current working directory', os.getcwd())
```

```
Current working directory D:\Hands-On\PGP-AIML\CaseStudies\Capstone Projects\1-NLP-Automatic.Ticket.Assignment
```

Extract Glove Embeddings

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

```
# Check if it is already extracted else Open the zipped file as readonly
if not os.path.isfile('glove.6B/glove.6B.200d.txt'):
    glove_embeddings = 'glove.6B.zip'
    with ZipFile(glove_embeddings, 'r') as archive:
        archive.extractall('glove.6B')

# List the files under extracted folder
os.listdir('glove.6B')
```

```
['glove.6B.100d.txt',
 'glove.6B.200d.txt',
 'glove.6B.300d.txt',
 'glove.6B.50d.txt']
```

Load the dataset

```
# Load the dataset into a Pandas dataframe called ticket and check the head of the dataset
ticket = pd.read_excel('Input Data Synthetic (created but not used in our project).xlsx', )
```

Check the first and last rows

```
ticket.head()
```

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

```
# Check the tail of the dataset
ticket.tail()
```

	Short description	Description	Caller	Assignment group
8495	emails not coming in from zz mail	\r\n\r\nreceived from: avglmrts.vhqmtiua@gmail...	avglmrts vhqmtiua	GRP_29
8496	telephony_software issue	telephony_software issue	rbozivdq gmlhrtvp	GRP_0
8497	vip2: windows password reset for tifpdchb pedx...	vip2: windows password reset for tifpdchb pedx...	oybwdsgx oxyhwrfz	GRP_0
8498	machine nÃ£o estÃ¡ funcionando	i am unable to access the machine utilities to...	ufawcgob aowhxjky	GRP_62
8499	an mehreren pc`s lassen sich verschiedene prgr...	an mehreren pc`s lassen sich verschiedene prgr...	kqvbrspl jyzoklfx	GRP_49

Observations :-

To take a closer look at the data, pandas library provides ".head()" function which returns first five observations and ".tail()" function which returns last five observations of the data set.

STRUCTURE OF DATASET

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.

Get the shape and size of the dataset

```
print('No of rows:\033[1m', ticket.shape[0], '\033[0m')
print('No of cols:\033[1m', ticket.shape[1], '\033[0m')
```

```
No of rows: 8500
No of cols: 4
```

Get more information about the data

1. Name of Columns

2. Find the data types of each columns

3. Look for any null/missing values

```
ticket.info()
```

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

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

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

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 occuring tickets which reflects in the Short description column.
- The top occuring 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.

DEALING WITH MISSING VALUES & TREATMENT

NULL treatment

Check the rows with null values

```
ticket[pd.isnull(ticket).any(axis=1)]
```

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

NULL/Missing value treatment replaces the NaN cells with just empty string:-

```
In [13]: # NULL replacement
         ticket.fillna(str(), inplace=True)
         ticket[pd.isnull(ticket).any(axis=1)]
```

Out[13]:

	Short description	Description	Caller	Assignment group

Verify if the replacement is as expected

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

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

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.

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**

Function applied to detect Mojibakes in the dataset

```
def is_mojibake_impacted(text):
    if not badness.sequence_weirdness(text):
        # nothing weird, should be okay
        return True
    try:
        text.encode('sloppy-windows-1252')
    except UnicodeEncodeError:
        # Not CP-1252 encodable, probably fine
        return True
    else:
        # Encodable as CP-1252, Mojibake alert level high
        return False

# Check the dataset for mojibake impact
ticket[~ticket.iloc[:,:-1].applymap(is_mojibake_impacted).all(1)]
```

	Short description	Description	Caller	Assignment group
99	password expiry tomorrow	\n\nreceived from: ecprjbod.litmjwsy@gmail.com...	ecprjbod litmjwsy	GRP_0
116	server issues	\r\n\r\nreceived from: bgqpotek.cuxakvml@gmail...	bgqpotek cuxakvml	GRP_0
124	mobile device activation	from: tvcdfqgp nrbcqwgj \nsent: friday, octobe...	tvcdfqgp nrbcqwgj	GRP_0
164	æ'¤å›ž: ticket_no1564867 -- comments added	\n\nreceived from: abcdri@company.com\n\nwindy...	tycludks cjofwigv	GRP_0
170	[urgent!!] delivery note creation request!!	\n\nreceived from: fbvpcytz.nokypgvx@gmail.com...	fbvpcytz nokypgvx	GRP_18
...	...	...	...	...
8470	please review your recent ticketing_tool ticke...	from: mikhghytr wafglhdrhjop \nsent: thursday,...	azxhejvq fyemlavd	GRP_16
8471	ç"µè„'å¼€æœ°å¼€ä¸□å‡°æ□¥	to å°□è´°ï¼Œæ—©ä¸Šç"µè„'å¼€æœ°å¼€ä¸□å‡°æ□¥	xqyjztnm onfusvlz	GRP_30
8480	customer group enhanced field	\r\n\r\nreceived from: nlearzwi.ukdzstwi@gmail...	nlearzwi ukdzstwi	GRP_9
8498	machine nÃ£o estÃ¡ funcionando	i am unable to access the machine utilities to...	ufawcgob aowhxjky	GRP_62
8499	an mehreren pc`s lassen sich verschiedene prgr...	an mehreren pc`s lassen sich verschiedene prgr...	kqvbrspl jyzoklfx	GRP_49

828 rows × 4 columns

Observation:-

828 rows contain Garbled text.

Check the list of Mojobakes in ftfy library, and test a sample fix:-

```
# Take an example of row# 8471 Short Desc and fix it
print('Grabled text: \033[1m%s\033[0m\nFixed text: \033[1m%s\033[0m' % (ticket['Short description'][8471],
                                                                       fix_text(ticket['Short description'][8471])))

# List all mojibakes defined in ftfy library
print('\nMojibake Symbol RegEx:\n', badness.MOJIBAKE_SYMBOL_RE.pattern)
```

```
Grabled text: ç”µè„‘å¼€æœºå¼€ä¸□å‡ºæ□¥
Fixed text: 电脑开机开不出来

Mojibake Symbol RegEx:
 [ÂÃĂ][□-□€ƒ„†‡ˆ‰‹Œ“•˜œŸ¡¢£¤¥¦§¨ª«¬¯°±²³µ¶·¸¹º¼½¾¿ˇ˘˝]|[ÂÃĂ][›»‘’“”©™]\w|[¬√][ÄÅÇÉÑÖÜáàâäãåçéèêëíìîïñúùûüı†¢£§¶ß®©™≠ÆØ¥ªæø≤≥]|\w√
[±∂]\w|[ðđ][Ÿ□]|â€|вђ[љћ¦°№™ќ“”]
```

Since the fix is successful, applying the fix to rest of the data.

```
ticket['Short description'] = ticket['Short description'].apply(fix_text)
ticket['Description'] = ticket['Description'].apply(fix_text)

# Visualize that row# 8471
ticket.iloc[8471,:]
```

```
Short description              电脑开机开不出来
Description           to 小贺,早上电脑开机开不出来
Caller               xqyjztnm onfusvlz
Assignment group                GRP_30
Name: 8471, dtype: object
```

Row 8471, which initially contained Mojibake is verified.

Serialize the mojibake treated dataset:-

```
ticket.to_csv('mojibake_treated.csv', index=False, encoding='utf_8_sig')
with open('mojibake_treated.pkl', 'wb') as handle:
    pickle.dump(ticket, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

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 successfuly able to get the grabled characters back into their original form using **ftfy.fix_text()``**, however it is observed that the row# 8471 is not English but Mandarine.
- 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.

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**

Define and construct the service urls:-

```
# Define and construct the service urls
svc_domains = ['.com','.com.au','.com.ar','.co.kr','.co.in','.co.jp','.at','.de','.ru','.ch','.fr','.es','.ae']
svc_urls = ['http://translate.google' + domain for domain in svc_domains]
```

Test Translation on row# 8471 Short description:-

```
# Take an example of row# 8471 Short Desc and fix it
# gs = Goslate(service_urls=svc_urls)
# trans_8471 = gs.translate(ticket['Short description'][8471], target_language='en', source_language='auto')
# print('Original text: \033[1m%s\033[0m\nFixed text: \033[1m%s\033[0m' % (ticket['Short description'][8471], trans_8471))
print('Original text: \033[1m%s\033[0m\nFixed text: \033[1m%s\033[0m' % ('电脑开机开不出来', 'Boot the computer does not really c
ome out'))
```

```
Original text: 电脑开机开不出来
Fixed text: Boot the computer does not really come out
```

Add Column Language to capture the language present in 'Short description' & 'Description'

```
# List of column data to consider for translation
trans_cols = ['Short description','Description']

# Add a new column to store the detected language
ticket.insert(loc=2, column='Language', value=np.nan, allow_duplicates=True)

# for idx in range(ticket.shape[0]):
#     # Instantiate Goslate class in each iteration
#     gs = Goslate(service_urls=svc_urls)
#     lang = gs.detect(' '.join(ticket.loc[idx, trans_cols].tolist()))
#     row_iter = gs.translate(ticket.loc[idx, trans_cols].tolist(),
#                             target_language='en',
#                             source_language='auto')
#     ticket.loc[idx, trans_cols] = list(row_iter)

# ticket.Language = lang
# DELETE this
with open('translated_ticket.pkl','rb') as f:
    ticket = pickle.load(f)
ticket.head()
```

	Short description	Description	Language	Caller	Assignment group
0	login issue	-verified user details.(employee# & manager na...	English	spxjnwir pjlcoqds	GRP_0
1	outlook	received from: hmjdrvpb.komuaywn@gmail.com\n\n...	English	hmjdrvpb komuaywn	GRP_0
2	cant log in to vpn	received from: eylqgodm.ybqkwiam@gmail.com\n\n...	English	eylqgodm ybqkwiam	GRP_0
3	unable to access hr_tool page	unable to access hr_tool page	English	xbkucsvz gcpydteq	GRP_0
4	Error skype	Error skype	English	owlgqjme qhcozdfx	GRP_0

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.

Serialize the translated dataset:-

```
# # Serialize the translated dataset
# ticket.to_csv('translated_ticket.csv', index=False, encoding='utf_8_sig')
# with open('translated_ticket.pkl','wb') as f:
#     pickle.dump(ticket, f, pickle.HIGHEST_PROTOCOL)
```

Load the translated pickle file incase the IP gets blocked:-

```
# Load the translated pickle file incase the IP gets blocked
with open('translated_ticket.pkl','rb') as f:
    ticket = pickle.load(f)
```

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.

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:

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:-

```
# Define regex patterns
EMAIL_PATTERN = r"([\w.+-]+@[a-z\d-]+\.[a-z\d.-]+)"
PUNCT_PATTERN = r"[,|@|\|?|\\|$&*|%|\r|\n|.:|\s+|/|//|\\|/|\||-|<|>|;|(|)|=|+|#|-|\"|[-\]]|{|}]"
# Negative Lookbehind for EmailId replacement- Don't match any number which follows the text "RetainedEmailId"
NUMER_PATTERN = r"(?<!RetainedEmailId)(\d+(?:\.\d+)?)"

# Define a function to treat the texts
def cleanseText(text):
    # Make the text unicase (lower)
    text = str(text).lower()
    # Remove email adresses
    # text = re.sub(EMAIL_PATTERN, '', text, flags=re.IGNORECASE)
    # Save Email addresses and replace them with custom keyword
    email_dict = extract_email(text)
    for key in email_dict.keys():
        text = text.replace(email_dict[key], key)
    # Remove all numbers
    text = re.sub(NUMER_PATTERN, '', text)
    # Replace all punctuations with blank space
    # text = re.sub(PUNCT_PATTERN, " ", text, flags=re.MULTILINE)
    text = text.translate(str.maketrans("","", string.punctuation))
    text = re.sub(r'\s+', ' ', text)
    # Replace multiple spaces from prev step to single
    text = re.sub(r' {2,}', " ", text, flags=re.MULTILINE)
    text = text.replace('`',"'")
    # Replace the email ids back into their original position
    for key in email_dict.keys():
        text = text.replace(key, email_dict[key])
    return text.strip()

def extract_email(text):
    # Replaces the email addresses with custom key word and
    # save them into a dictionary for future use
    unique_emailid = set(re.findall(EMAIL_PATTERN, text))
    email_replacement = dict()
    for idx, email in enumerate(unique_emailid):
        email_replacement[f'RetainedEmailId{idx}'] = email
    return email_replacement
```

Test preprocessing on Description for row# 32:-

```
# Take an example of row# 32 Description and fix it
print('\033[1mOriginal text:\033[0m')
print(ticket['Description'][32])
print('_'*100)
print('\033[1mCleaned text:\033[0m')
print(cleanseText(ticket['Description'][32]))
```

```
Original text:
received from: kxsceyzo.naokumlb@gmail.com

gentles,

i have two devices that are trying to share an ip address. they are trying to share 96.26.27.9619. one is a printer with the ho
stname of prtjc0074, and the other is a new display for erp. the display is using dhcp to get its address assigned and the prin
ter is hard coded.

my guess is that the address 96.26.27.9619 did not get set to a static address in dhcp. i need this corrected so the display wi
ll pick up another address.

____________________________________________________________________________________________________
Cleaned text:
received from kxsceyzo.naokumlb@gmail.com gentles i have two devices that are trying to share an ip address they are trying to
share one is a printer with the hostname of prtjc and the other is a new display for erp the display is using dhcp to get its a
ddress assigned and the printer is hard coded my guess is that the address did not get set to a static address in dhcp i need t
his corrected so the display will pick up another address
```

Apply preprocessing to entire dataset:-

```
# Apply the cleaning function to entire dataset
ticket['Description'] = ticket['Description'].apply(cleanseText)
ticket['Short description'] = ticket['Short description'].apply(cleanseText)

# Verify the data
ticket.tail()
```

	Short description	Description	Language	Caller	Assignment group
8495	emails not coming in from zz mail	received from avglmrts.vhqmtiua@gmail.com good...	English	avglmrts vhqmtiua	GRP_29
8496	telephonysoftware issue	telephonysoftware issue	English	rbozivdq gmlhrtvp	GRP_0
8497	vip windows password reset for tifpdchb pedxruyf	vip windows password reset for tifpdchb pedxruyf	English	oybwdsgx oxyhwrfz	GRP_0
8498	machine is not working	i am unable to access the machine utilities to...	Portuguese	ufawcgob aowhxjky	GRP_62
8499	various prgramdntyme can not be opened on mult...	various prgramdntyme can not be opened on mult...	German	kqvbrspl jyzoklfx	GRP_49

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.

BUILDING WORD VOCABULARY

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.

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

Initialize spacy 'en' medium model, keeping only tagger component needed for lemmatization:-

```
nlp = spacy.load('en_core_web_md', disable=['parser', 'ner'])
```

Define a function to lemmatize the descriptions:-

```
def lemmatizer(sentence):
    # Parse the sentence using the loaded 'en' model object `nlp`
    doc = nlp(sentence)
    return " ".join([token.lemma_ for token in doc if token.lemma_ !='-PRON-'])
```

Testing lemmatization on Description in row# 43:-

```
print('\033[1mOriginal text:\033[0m')
print(ticket['Description'][43])
print('_'*100)
print('\033[1mLemmatized text:\033[0m')
print(lemmatizer(ticket['Description'][43]))
```

```
Original text:
received from yisohglr.uvteflgb@gmail.com hi the printer printer is not working and needs a part replaced can you reroute the j
obs in queue to printer printer wihuyjdo qpogfwkb has indicated that prqos needs a new part and it may not deliver for a few da
ys so the inwarehousetools will need to print on printer for now this needs to be taken care of today since the inwarehousetool
s are printed and are picked up by an outside vendor at pm in usa on a daily basis please contact dkmcfreg anwmfvlgenkataramdnt
yana if you have questions about the jobs in queue for today

____________________________________________________________________________________________________
Lemmatized text:
receive from yisohglr.uvteflgb@gmail.com hi the printer printer be not work and need a part replace can reroute the job in queu
e to printer printer wihuyjdo qpogfwkb have indicate that prqos need a new part and may not deliver for a few day so the inware
housetool will need to print on printer for now this need to be take care of today since the inwarehousetool be print and be pi
ck up by an outside vendor at pm in usa on a daily basis please contact dkmcfreg anwmfvlgenkataramdntyana if have question abou
t the job in queue for today
```

Apply Lemmatization to entire dataset and verify:-

```
ticket['Description'] = ticket['Description'].apply(lemmatizer)
ticket['Short description'] = ticket['Short description'].apply(lemmatizer)

# Verify the data
ticket.tail()
```

	Short description	Description	Language	Caller	Assignment group
8495	email not come in from zz mail	receive from avglmrts.vhqmtiua@gmail.com good ...	English	avglmrts vhqmtiua	GRP_29
8496	telephonysoftware issue	telephonysoftware issue	English	rbozivdq gmlhrtvp	GRP_0
8497	vip windows password reset for tifpdchb pedxruyf	vip windows password reset for tifpdchb pedxruyf	English	oybwdsgx oxyhwrfz	GRP_0
8498	machine be not work	i be unable to access the machine utility to f...	Portuguese	ufawcgob aowhxjky	GRP_62
8499	various prgramdntyme can not be open on multip...	various prgramdntyme can not be open on multip...	German	kqvbrspl jyzoklfx	GRP_49

Serialize the preprocessed dataset:-

```
ticket.to_csv('preprocessed_ticket.csv', index=False, encoding='utf_8_sig')
with open('preprocessed_ticket.pkl','wb') as f:
    pickle.dump(ticket, f, pickle.HIGHEST_PROTOCOL)
```

Create new features of length and word count for both of the description columns:-

```
ticket.insert(1, 'sd_len', ticket['Short description'].astype(str).apply(len))
ticket.insert(2, 'sd_word_count', ticket['Short description'].apply(lambda x: len(str(x).split())))
ticket.insert(4, 'desc_len', ticket['Description'].astype(str).apply(len))
ticket.insert(5, 'desc_word_count', ticket['Description'].apply(lambda x: len(str(x).split())))
ticket.head()
```

	Short description	sd_len	sd_word_count	Description	desc_len	desc_word_count	Language	Caller	Assignment group
0	login issue	11	2	verified user detailsemployee manager name che...	177	31	English	spxjnwir pjlcoqds	GRP_0
1	outlook	7	1	receive from hmjdrvpb.komuaywn@gmail.com hello...	163	23	English	hmjdrvpb komuaywn	GRP_0
2	can not log in to vpn	21	6	receive from eylqgodm.ybqkwiam@gmail.com hi i ...	72	12	English	eylqgodm ybqkwiam	GRP_0
3	unable to access hrtool page	28	5	unable to access hrtool page	28	5	English	xbkucsvz gcpydteq	GRP_0
4	error skype	11	2	error skype	11	2	English	owlgqjme qhcozdfx	GRP_0

CREATING TOKENS

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

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
```

Univariate visualization

Single-variable or univariate visualization is the simplest type of visualization which consists of observations on only a single characteristic or attribute. Univariate visualization includes histogram, bar plots and line charts.

Visualizations continued as part of Finding Inconsistencies in data…

FINDING INCONSISTENCIES IN THE DATA

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

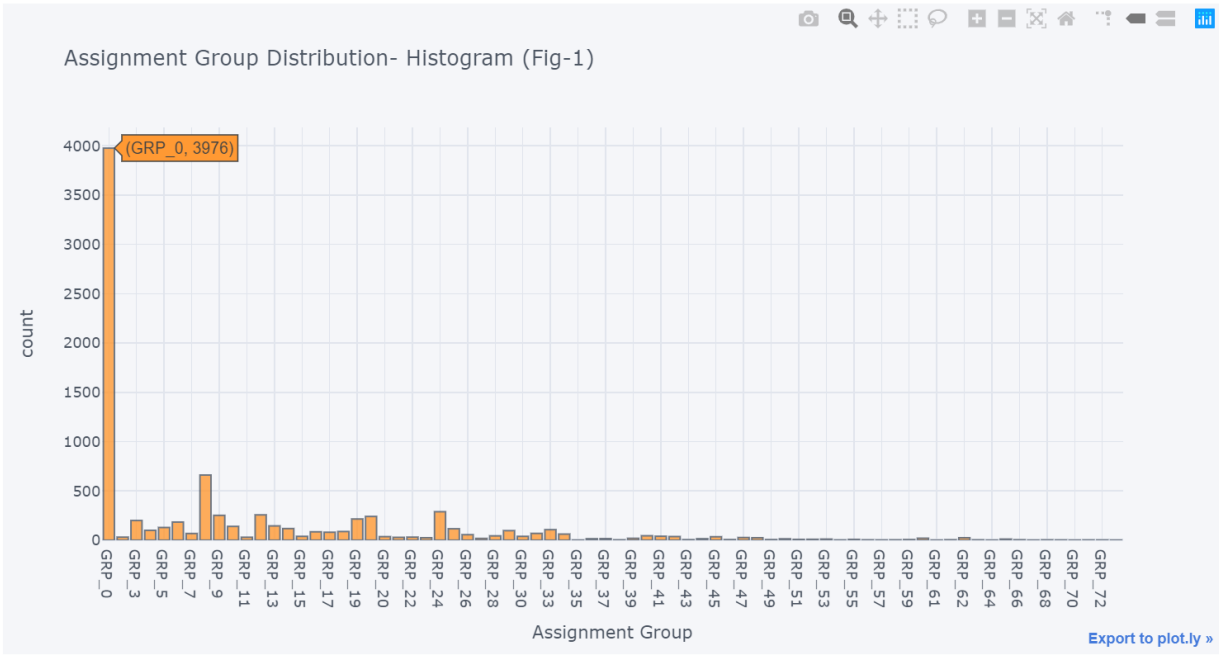
Following Plots display how the Assignments Groups are scattered across the dataset.

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

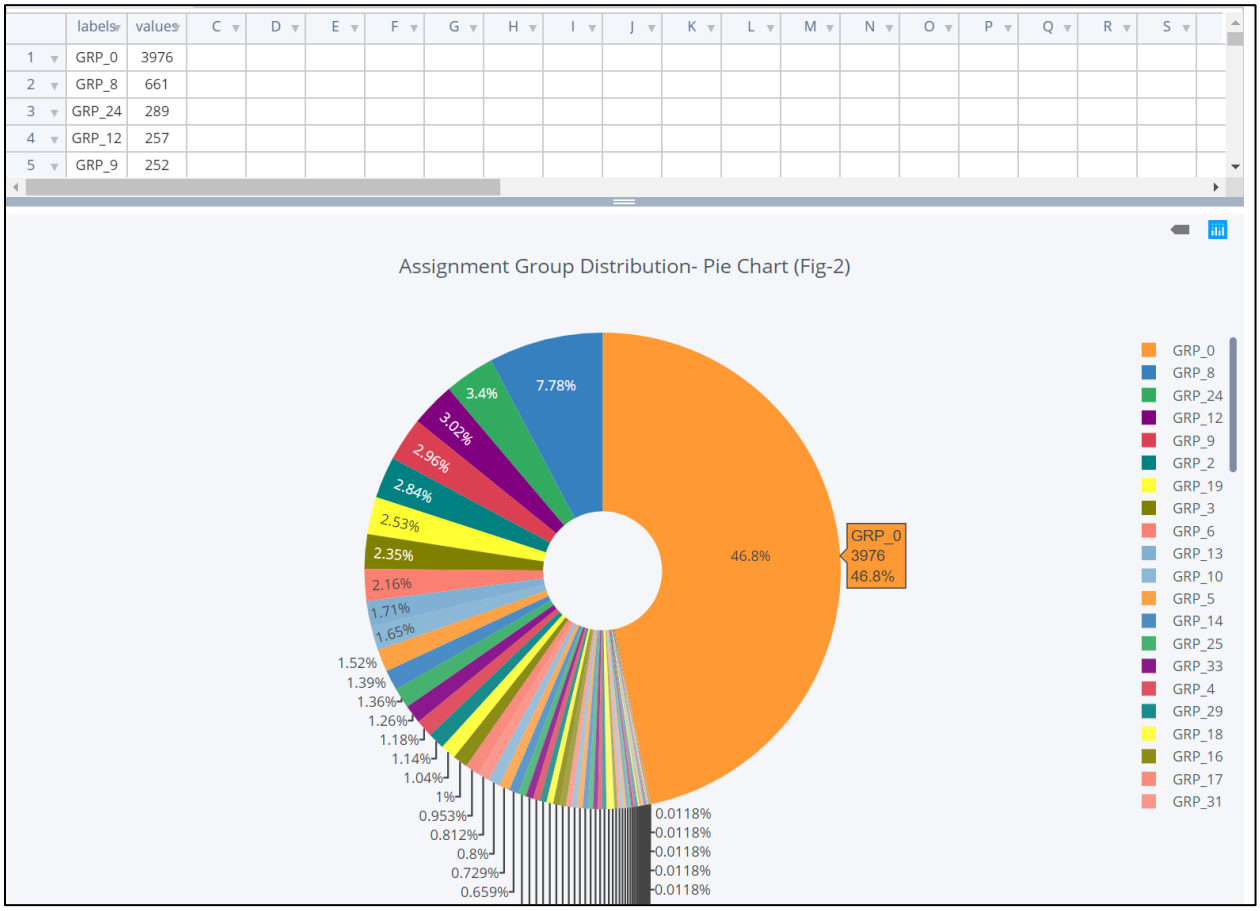
```
# Assignment group distribution
print('\033[1mTotal assignment groups:\033[0m', ticket['Assignment group'].nunique())

# Histogram
ticket['Assignment group'].iplot(
    kind='hist',
    xTitle='Assignment Group',
    yTitle='count',
    title='Assignment Group Distribution- Histogram (Fig-1)')

# Pie chart
assgn_grp = pd.DataFrame(ticket.groupby('Assignment group').size(),columns = ['Count']).reset_index()
assgn_grp.iplot(
    kind='pie',
    labels='Assignment group',
    values='Count',
    title='Assignment Group Distribution- Pie Chart (Fig-2)',
    hoverinfo="label+percent+name", hole=0.25)

# Bar plot
ticket['Assignment group'].iplot(
    kind='bar',
    yTitle='Assignment Group',
    xTitle='Record #',
    colorscale='-plotly',
    title='Assignment Group Distribution- Bar Chart (Fig-3)')
```

Assignment Group Distribution- Histogram:-

Total assignment groups: 74

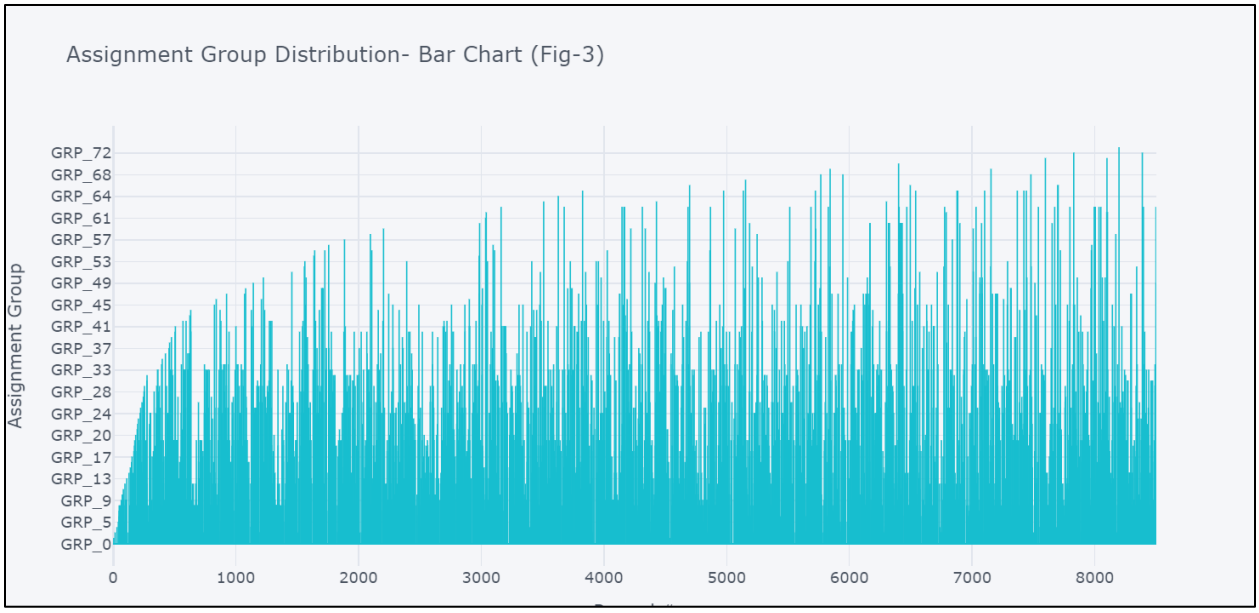


Assignment Group Distribution- Pie Chart:-



	labels	values
1	GRP_0	3976
2	GRP_8	661
3	GRP_24	289
4	GRP_12	257
5	GRP_9	252

Assignment Group Distribution- Bar Chart:-



Central Limit Theorem

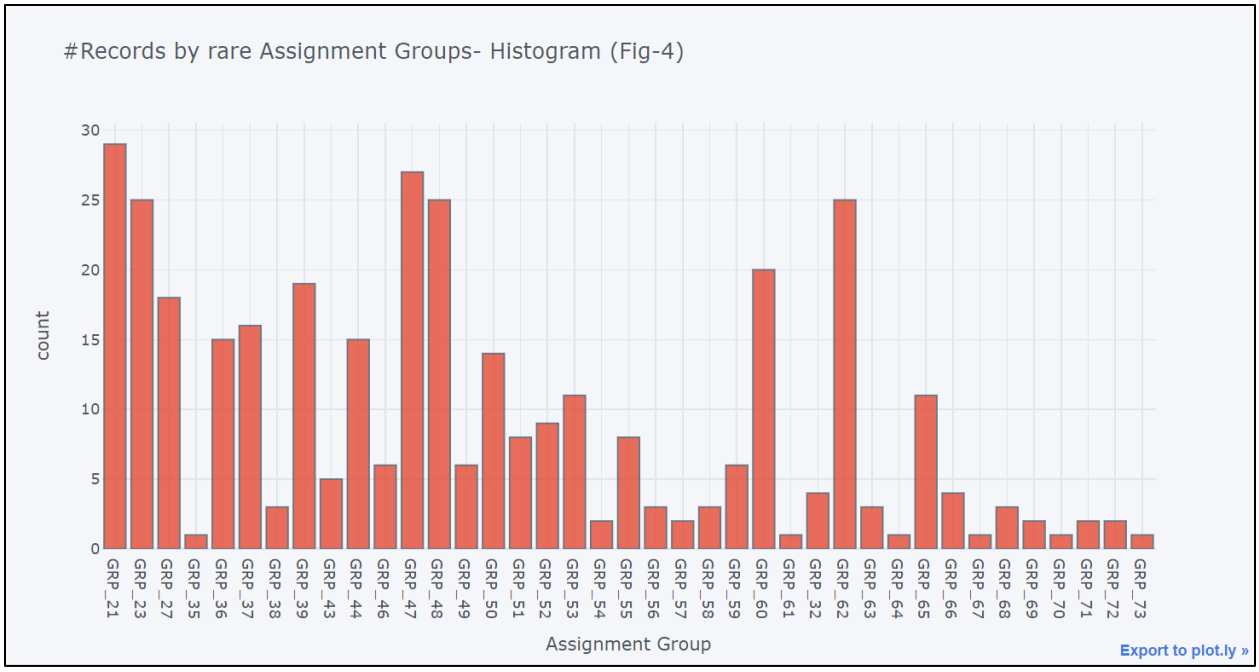
The Central Limit Theorem states that the sampling distribution of the sample means approaches a normal distribution as the sample size gets larger — no matter what the shape of the population distribution. This fact holds especially true and practitioners have been presumably considered it for sample 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:-

```
rare_ticket = ticket.groupby(['Assignment group']).filter(lambda x: len(x) < 30)
print('\033[1m#Groups with less than 30 tickets assigned:\033[0m', rare_ticket['Assignment group'].nunique())

rare_ticket['Assignment group'].iplot(
    kind='hist',
    xTitle='Assignment Group',
    yTitle='count',
    colorscale='-orrd',
    title='#Records by rare Assignment Groups- Histogram (Fig-4)')
```

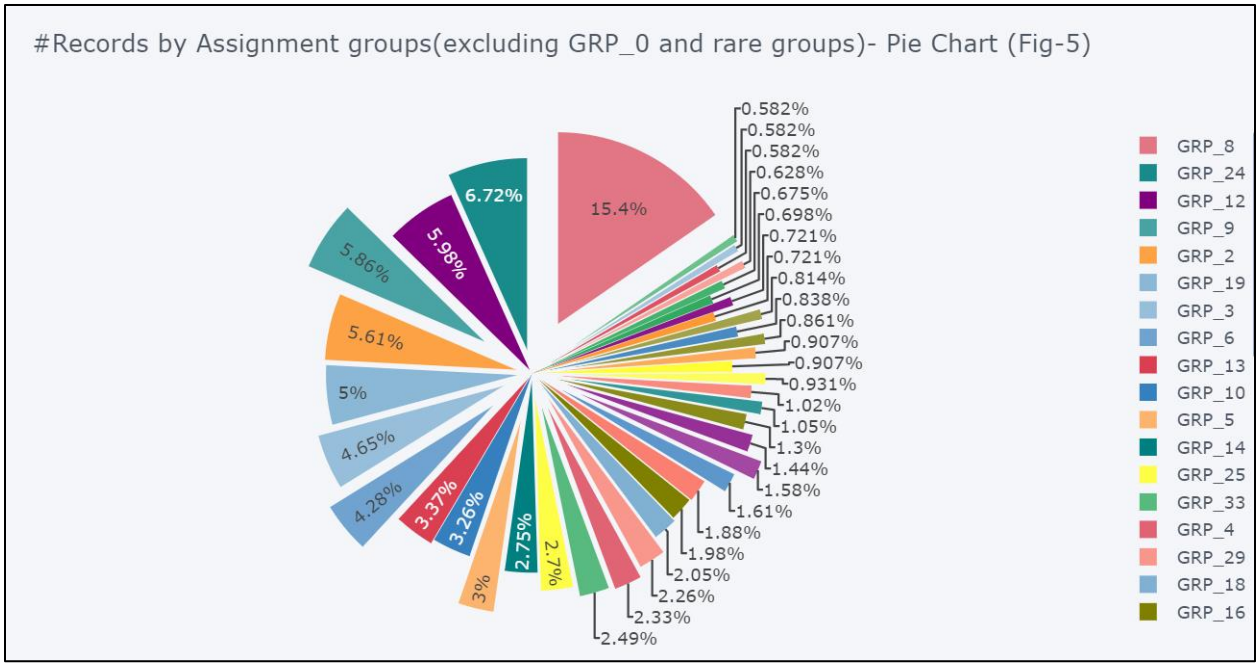
Incident Distribution across 40 Assignment Groups:-



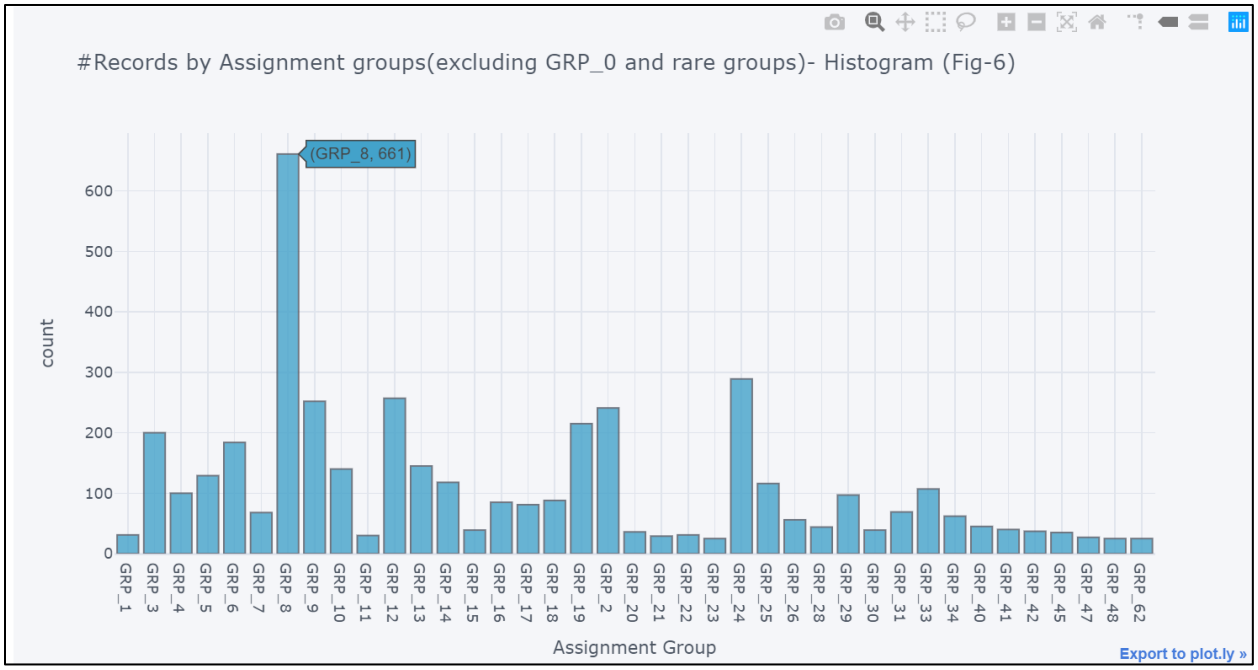
Distribution of Assignment groups excluding GRP_0 & rare groups (groups with 30 or less tickets assigned):-

```
excluded_grp = ['GRP_0']
excluded_grp.extend(rare_ticket['Assignment group'].unique())
filtered_tkt = ticket[~ticket['Assignment group'].isin(excluded_grp)]

# Pie chart
filtered_assgn_grp = pd.DataFrame(filtered_tkt.groupby('Assignment group').size(),columns = ['Count']).reset_index()
filtered_assgn_grp.iplot(
    kind='pie',
    labels='Assignment group',
    values='Count',
    title='#Records by Assignment groups(excluding GRP_0 and rare groups)- Pie Chart (Fig-5)',
    pull=np.linspace(0,0.3,filtered_assgn_grp['Assignment group'].nunique()))

# Histogram
filtered_tkt['Assignment group'].iplot(
    kind='histogram',
    xTitle='Assignment Group',
    yTitle='count',
    colorscale='-gnbu',
    title='#Records by Assignment groups(excluding GRP_0 and rare groups)- Histogram (Fig-6)')
```

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



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.

VISUALIZATION DIFFERENT PATTERNS

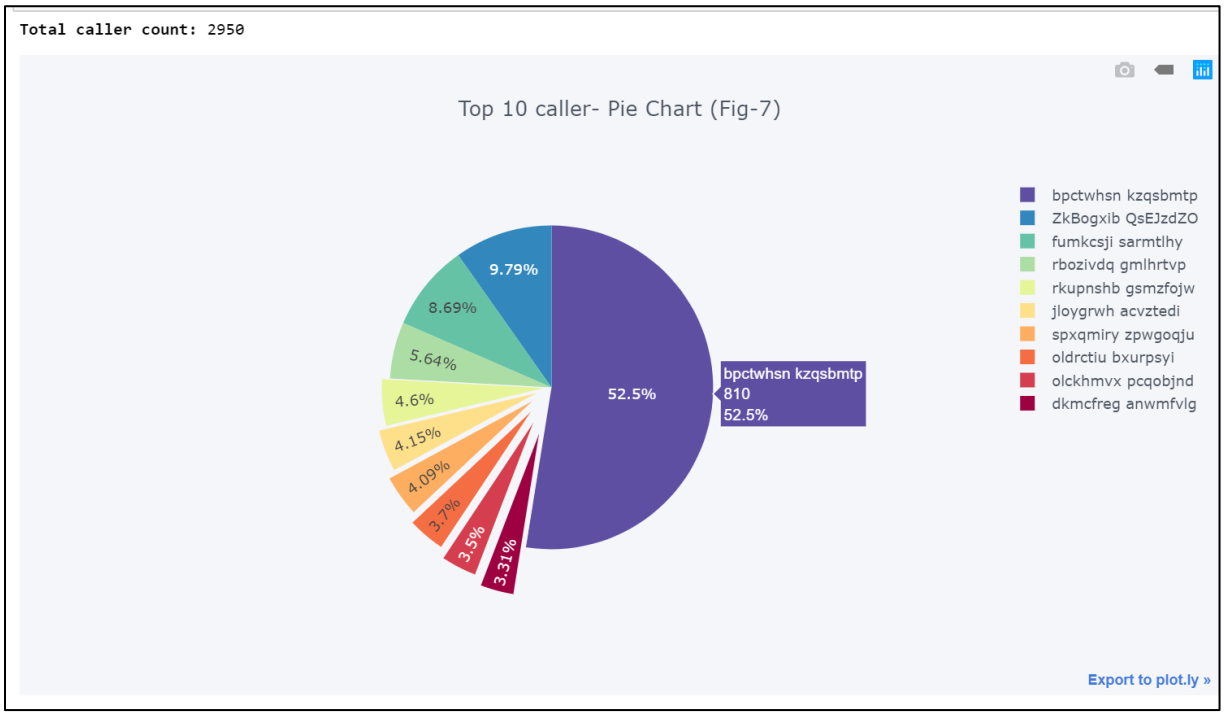
The distribution of Callers:-

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

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

```
print('\033[1mTotal caller count:\033[0m', ticket['Caller'].nunique())
df = pd.DataFrame(ticket.groupby(['Caller']).size().nlargest(10), columns=['Count']).reset_index()
df.iplot(kind='pie',
         labels='Caller',
         values='Count',
         title='Top 10 caller- Pie Chart (Fig-7)',
         colorscale='-spectral',
         pull=[0,0,0,0,0.05,0.1,0.15,0.2,0.25,0.3])
```

```
Total caller count: 2950
```



Top 5 callers in each assignment group:-

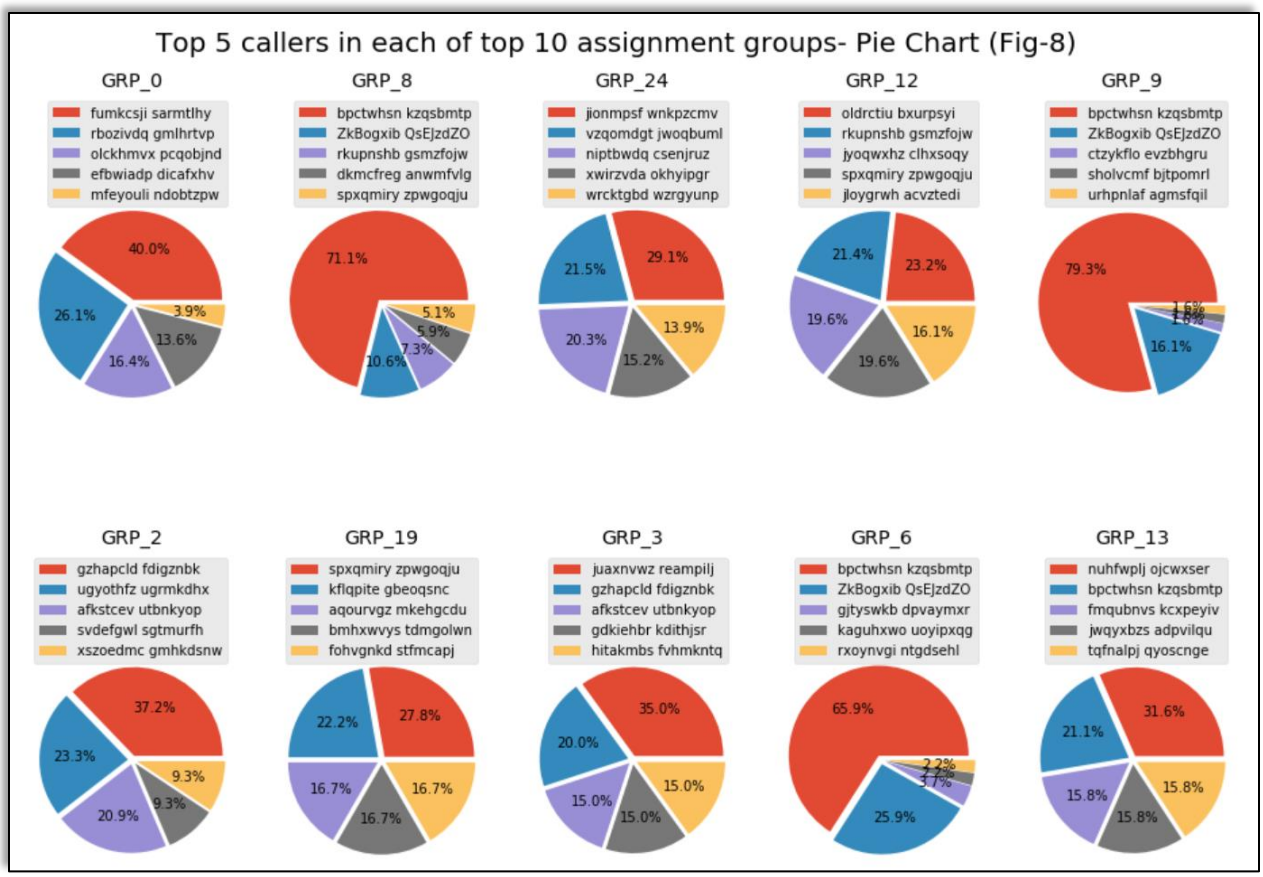
```
top_n = 5
s = ticket['Caller'].groupby(ticket['Assignment group']).value_counts()
caller_grp = pd.DataFrame(s.groupby(level=0).nlargest(top_n).reset_index(level=0, drop=True))
caller_grp.head(15)
```

		Caller
Assignment group	**Caller**	
GRP_0	**fumkcsji sarmtlhy**	132
	rbozivdq gmlhrtvp	86
	olckhmvx pcqobjnd	54
	efbwiadp dicafxhv	45
	mfeyouli ndobtzpw	13
GRP_1	**bpctwhsn kzqsbmtp**	6
	jloygrwh acvztedi	4
	jyoqwxhz clhxsoqy	3
	spxqmiry zpwgoqju	3
	kbnfxpsy gehxzayq	2
GRP_10	**bpctwhsn kzqsbmtp**	60
	ihfkwzjd erbxoyqk	6
	dizquolf hlykecxa	5
	gnasmtvx cwxtsvkm	3
	hlrmufzx qcdzierm	3

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

```
top_n = 10
top_grps = assgn_grp.nlargest(top_n, 'Count')['Assignment group'].tolist()

fig_cols = 5
fig_rows = int(np.ceil(top_n/fig_cols))
fig, axes = plt.subplots(fig_rows, fig_cols, figsize=(13,9.5))
fig.suptitle('Top 5 callers in each of top 10 assignment groups- Pie Chart (Fig-8)', y=1, va= 'bottom', size='20')
for row in range(fig_rows):
    for col in range(fig_cols):
        grp_n = fig_cols * row + col
        if grp_n < top_n:
            xs = caller_grp.xs(top_grps[grp_n])
            _ = axes[row,col].pie(xs, autopct='%1.1f%%', explode=[0.05]*5)
            axes[row,col].legend(labels=xs.index,loc="best")
            axes[row,col].axis('equal')
            axes[row,col].set_title(top_grps[grp_n])

plt.tight_layout()
```

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



Check if any caller raise Incident for multiple Assignment groups:-

```
mul_caller = caller_grp[caller_grp.Caller.duplicated()]
uni_mul_caller = [idx[1] for idx in mul_caller.index[mul_caller.Caller.unique()]]
print(f'\033[1mFollowing {len(uni_mul_caller)} callers happen to raise tickets for multiple groups:\033[0m\n')
print(uni_mul_caller)

mul_caller
```

Following 15 callers happen to raise tickets for multiple groups:

```
['hlrmufzx qcdzierm', 'fbgetczn jlsvxura', 'gnasmtvx cwxtsvkm', 'ihfkwzjd erbxoyqk', 'tqfnalpj qyoscnge', 'fmqubnvs kcxpeyiv',
'tghrloks jbgcvlmf', 'jwqyxbzs adpvilqu', 'nuhfwplj ojcwxser', 'oldrctiu bxurpsyi', 'vlymsnej whlqxcst', 'dkmcfreg anwmfvlg',
'bpctwhsn kzqsbmtp', 'spxqmiry zpwgoqju', 'obanjrhg rnafleys']
```

		Caller
Assignment group	**Caller**	
GRP_1	**spxqmiry zpwgoqju**	3
GRP_10	**ihfkwzjd erbxoyqk**	6
	gnasmtvx cwxtsvkm	3
	hlrmufzx qcdzierm	3
GRP_11	**tghrloks jbgcvlmf**	2
...	...	...
GRP_73	**kcnosyae zlpmfxgs**	1
GRP_8	**ZkBogxib QsEJzdZO**	54
GRP_9	**ctzykflo evzbhgru**	3
	sholvcmf bjtpomrl	3
	urhpnlaf agmsfqil	3

281 rows × 1 columns

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.

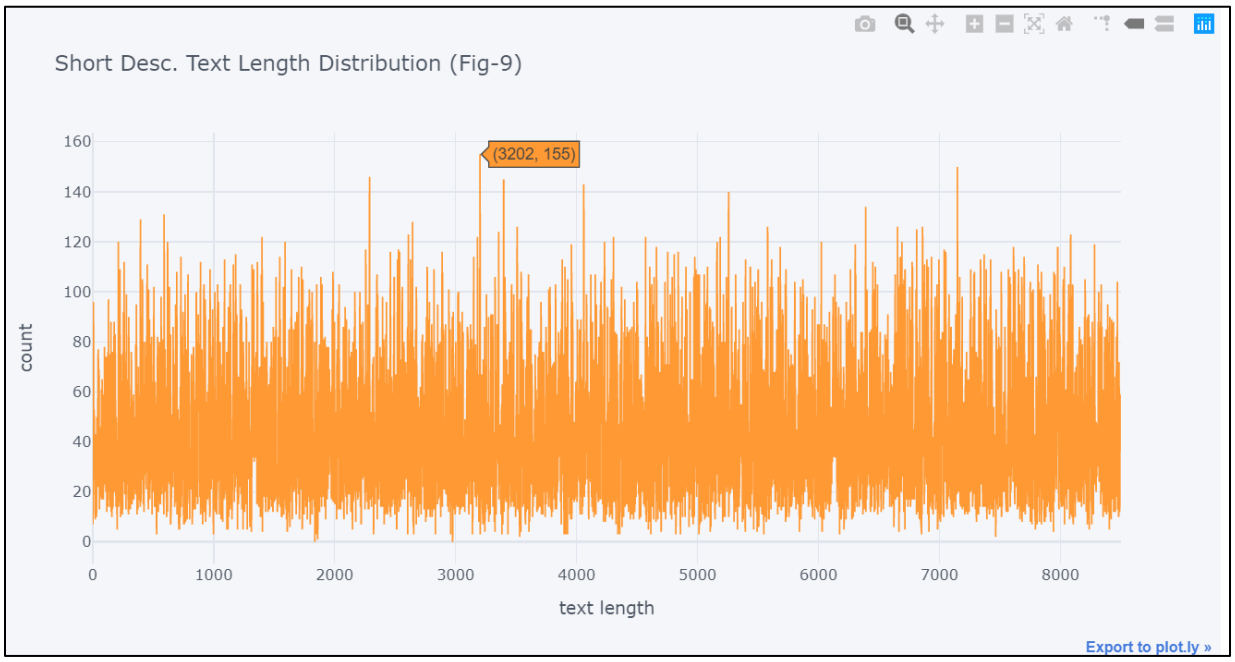
VISUALIZATION DIFFERENT TEXT PATTERNS

Distribution of Short description length

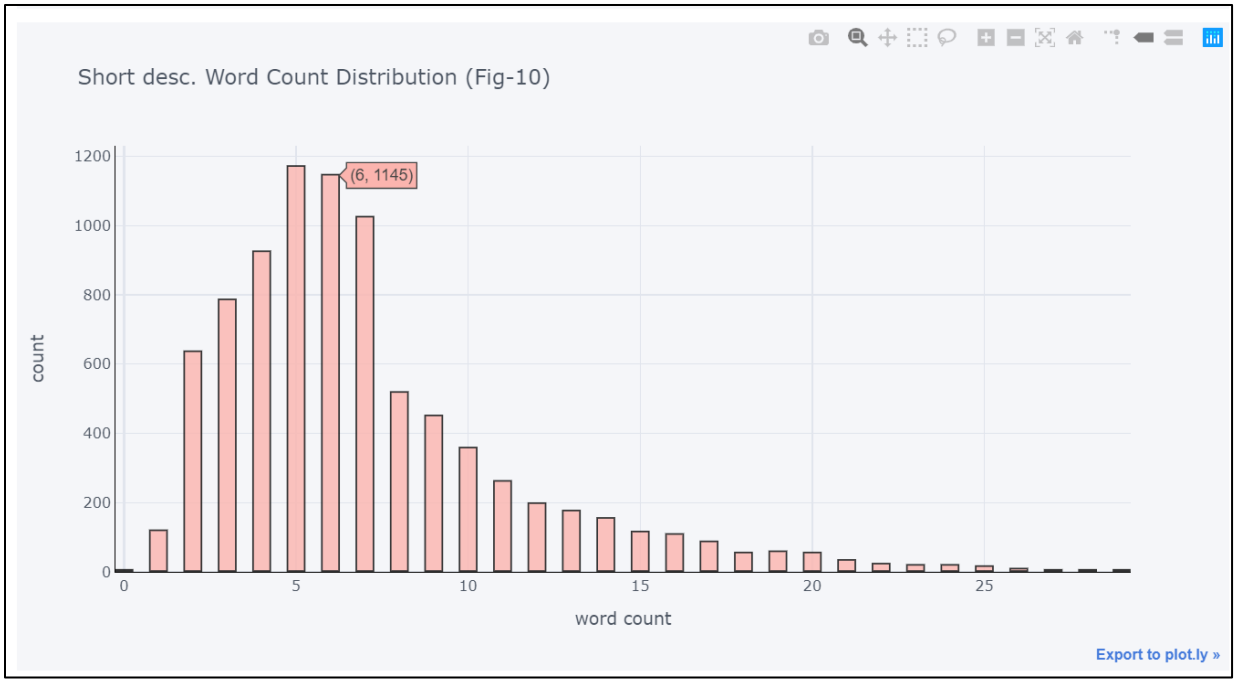
Plots the variation of length and word count of Short description attribute:-

```
# Short Desc text Length
ticket['sd_len'].iplot(
    kind='scatter',
    xTitle='text length',
    yTitle='count',
    title='Short Desc. Text Length Distribution (Fig-9)')

# Short desc word count
ticket['sd_word_count'].iplot(
    kind='hist',
    bins=100,
    xTitle='word count',
    linecolor='black',
    yTitle='count',
    colorscale='pastel1',
    title='Short desc. Word Count Distribution (Fig-10)')
```

Visualize variation of length of Short description attribute



Visualize variation of word count of Short description attribute

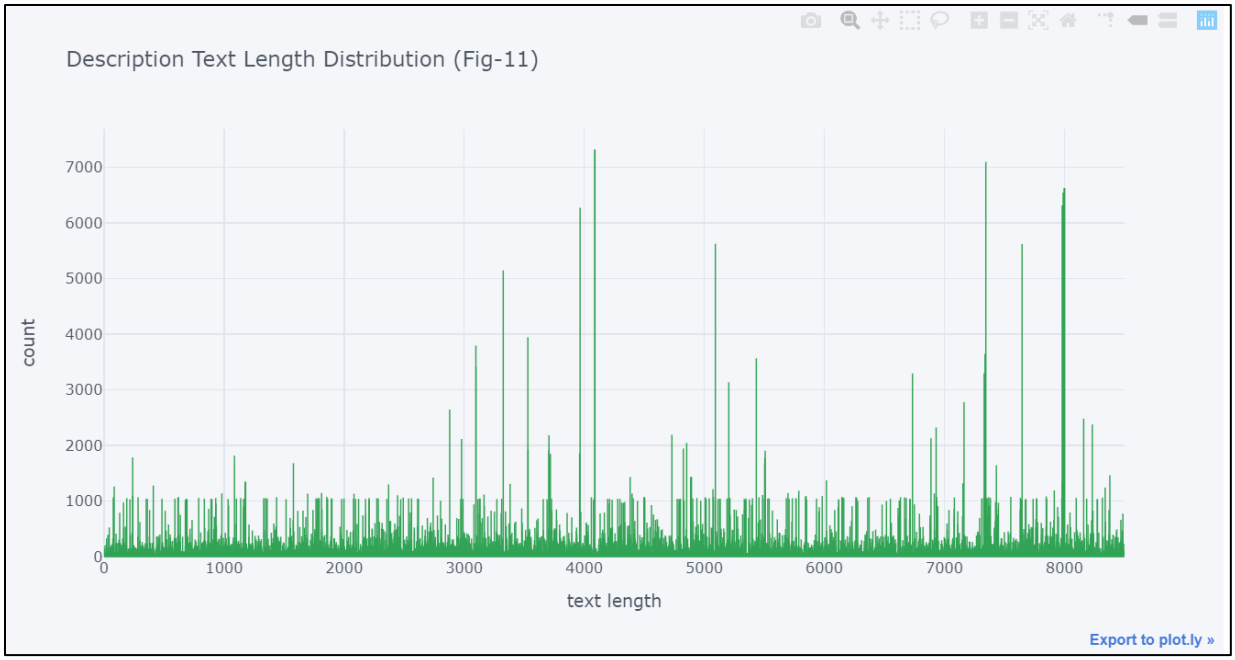


Distribution of Description length

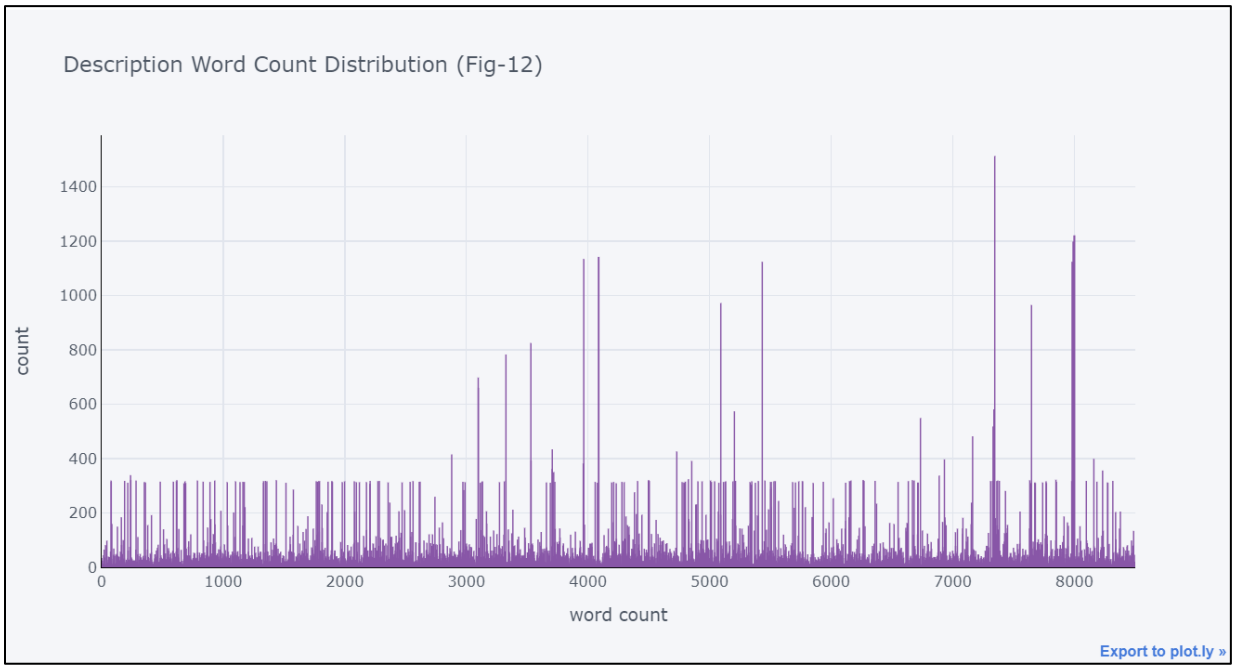
Plots the variation of length and word count of Description attribute:-

```
# Description text length
ticket['desc_len'].iplot(
    kind='bar',
    xTitle='text length',
    yTitle='count',
    colorscale='-ylgn',
    title='Description Text Length Distribution (Fig-11)')

# Description word count
ticket['desc_word_count'].iplot(
    kind='bar',
    xTitle='word count',
    linecolor='black',
    yTitle='count',
    colorscale='-bupu',
    title='Description Word Count Distribution (Fig-12)')
```

Visualize variation of length of Description attribute



Visualize variation of word count of Description attribute



Observation:-

Users preferred to leave long descriptions.

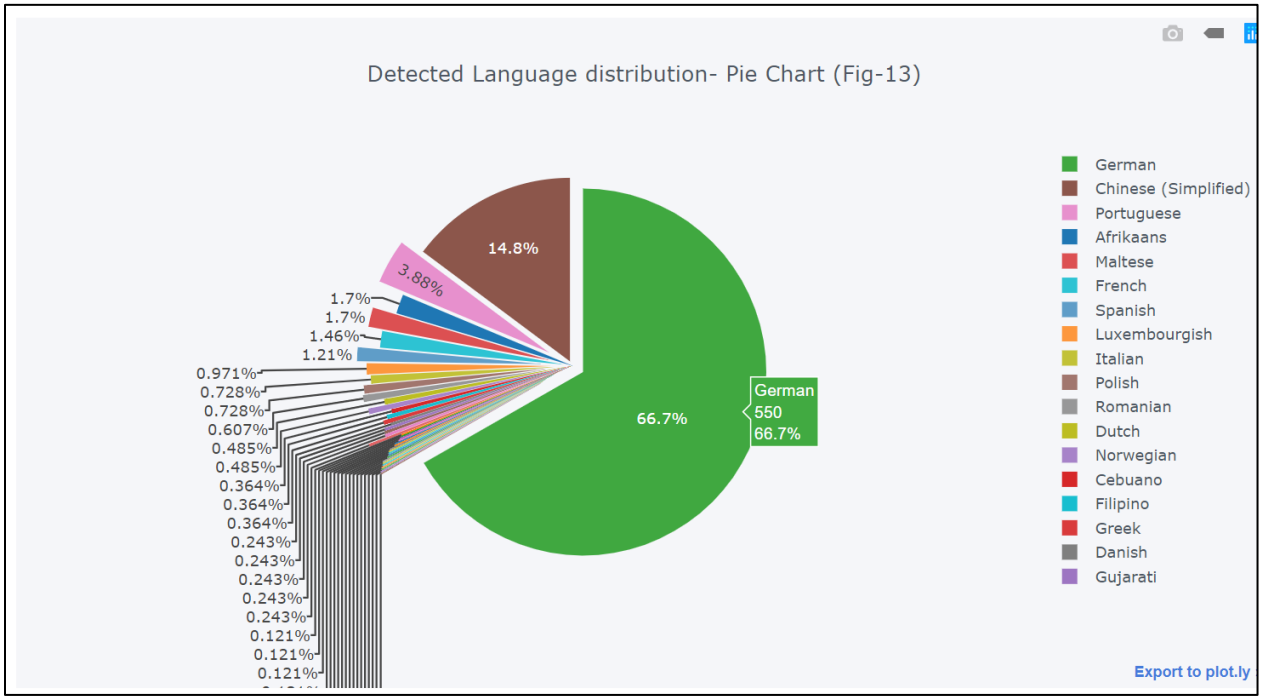
The distribution of Language

Plots the variation of tickets received in different languages:-

```
# Find out the volume of tickets received in different languages other than English
print('\033[1mTotal languages detected:\033[0m', ticket.Language.nunique())
df = pd.DataFrame(ticket.groupby(['Language']).size(), columns=['Count']).reset_index()

# Pie Chart
df[df.Language != 'English'].iplot(kind='pie',
         labels='Language',
         values='Count',
         title='Detected Language distribution- Pie Chart (Fig-13)',
         colorscale='plotly',
         pull=np.linspace(0,0.2,df.Language.nunique() - 1))
```

Total languages detected: 40

Visualize Incident distribution across non-English languages



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.

Merge the Short description and Description column texts:-

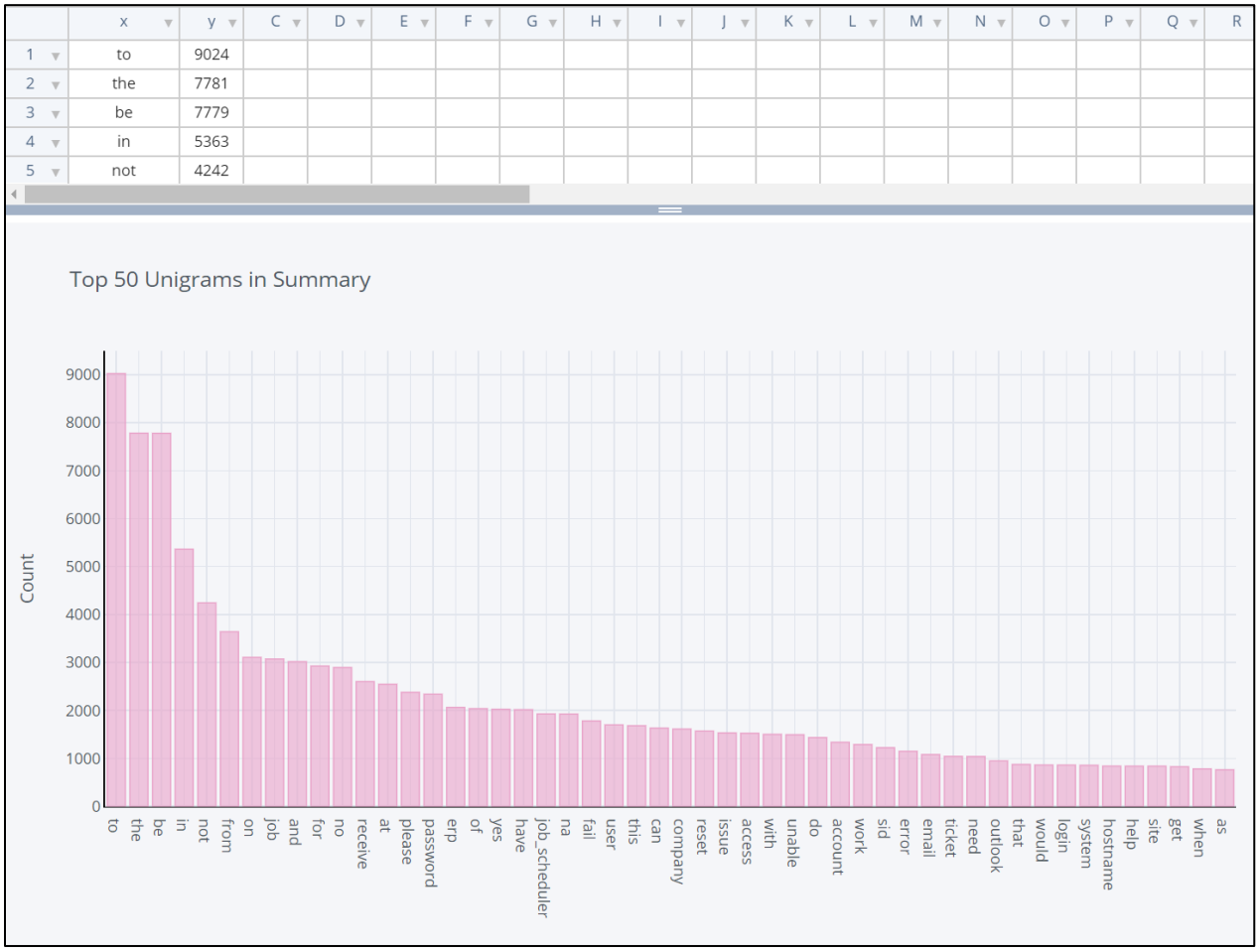
```
ticket['Summary'] = ticket['Short description'].str.strip() + ' ' + ticket['Description'].str.strip()
# Extend the English Stop Wordss
STOP_WORDS = STOPWORDS.union({'yes','na','hi',
                              'receive','hello',
                              'regards','thanks',
                              'from','greeting',
                              'forward','reply',
                              'will','please',
                              'see','help','able'})

# Generic function to derive top N n-grams from the corpus
def get_top_n_ngrams(corpus, top_n=None, ngram_range=(1,1), stopwords=None):
    vec = CountVectorizer(ngram_range=ngram_range,
                          stop_words=stopwords).fit(corpus)
    bag_of_words = vec.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in vec.vocabulary_.items()]
    words_freq = sorted(words_freq, key = lambda x: x[1], reverse=True)
    return words_freq[:top_n]
```

Top 50 Unigrams before removing Stop Words:-

Summary = Short description + Description

```
top_n = 50
ngram_range = (1,1)
uni_grams = get_top_n_ngrams(ticket.Summary, top_n, ngram_range)

df = pd.DataFrame(uni_grams, columns = ['Summary' , 'count'])
df.groupby('Summary').sum()['count'].sort_values(ascending=False).iplot(
    kind='bar',
    yTitle='Count',
    linecolor='black',
    colorscale='piyg',
    title=f'Top {top_n} Unigrams in Summary')
```

	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																

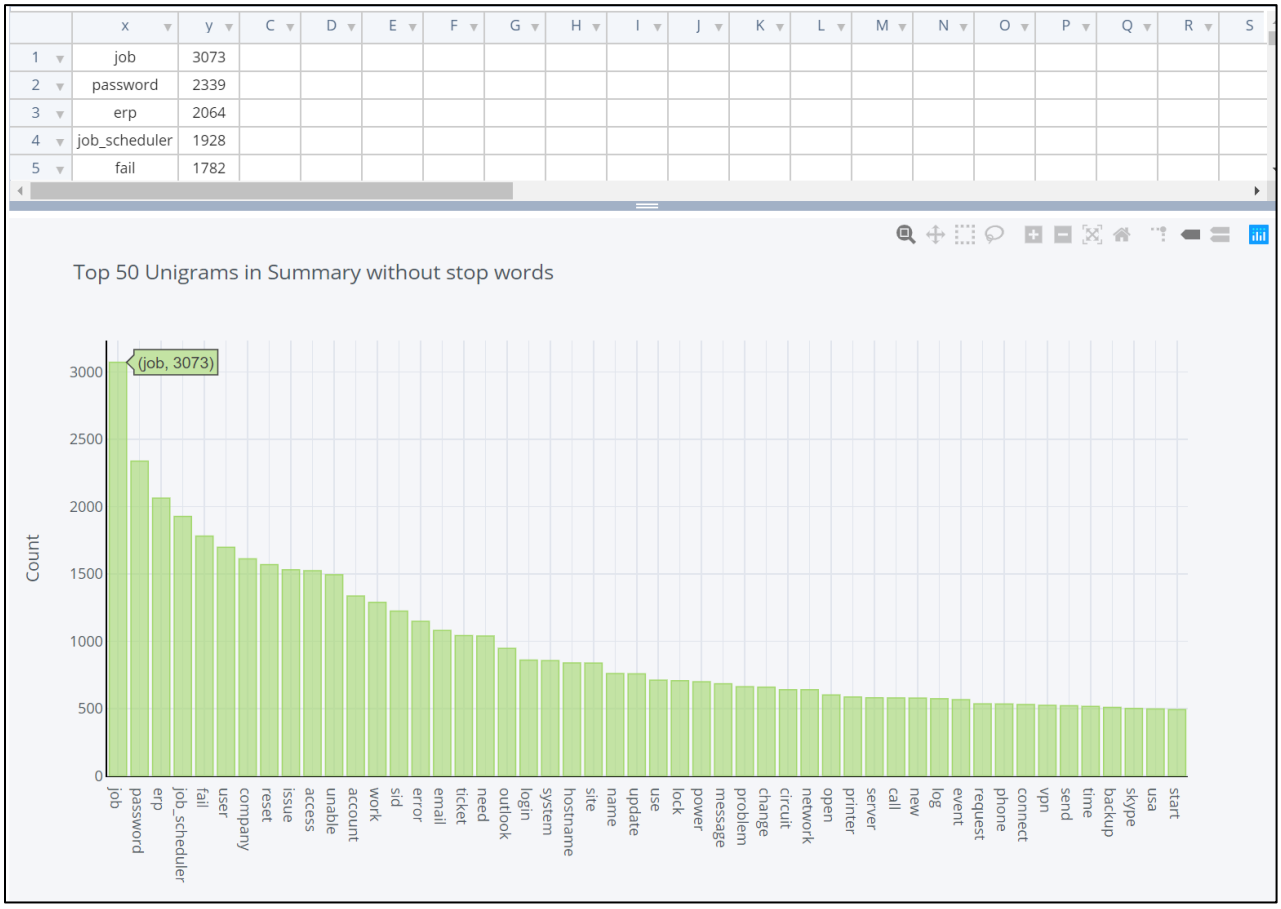


Top 50 Unigrams after removing Stop Words:-

Summary = Short description + Description

```
uni_grams_sw = get_top_n_ngrams(ticket.Summary, top_n, ngram_range, stopwords=STOP_WORDS)

df = pd.DataFrame(uni_grams_sw, columns = ['Summary' , 'count'])
df.groupby('Summary').sum()['count'].sort_values(ascending=False).iplot(
    kind='bar',
    yTitle='Count',
    linecolor='black',
    colorscale='-piyg',
    title=f'Top {top_n} Unigrams in Summary without stop words')
```

	x	y	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	job	3073																	
2	password	2339																	
3	erp	2064																	
4	job_scheduler	1928																	
5	fail	1782																	

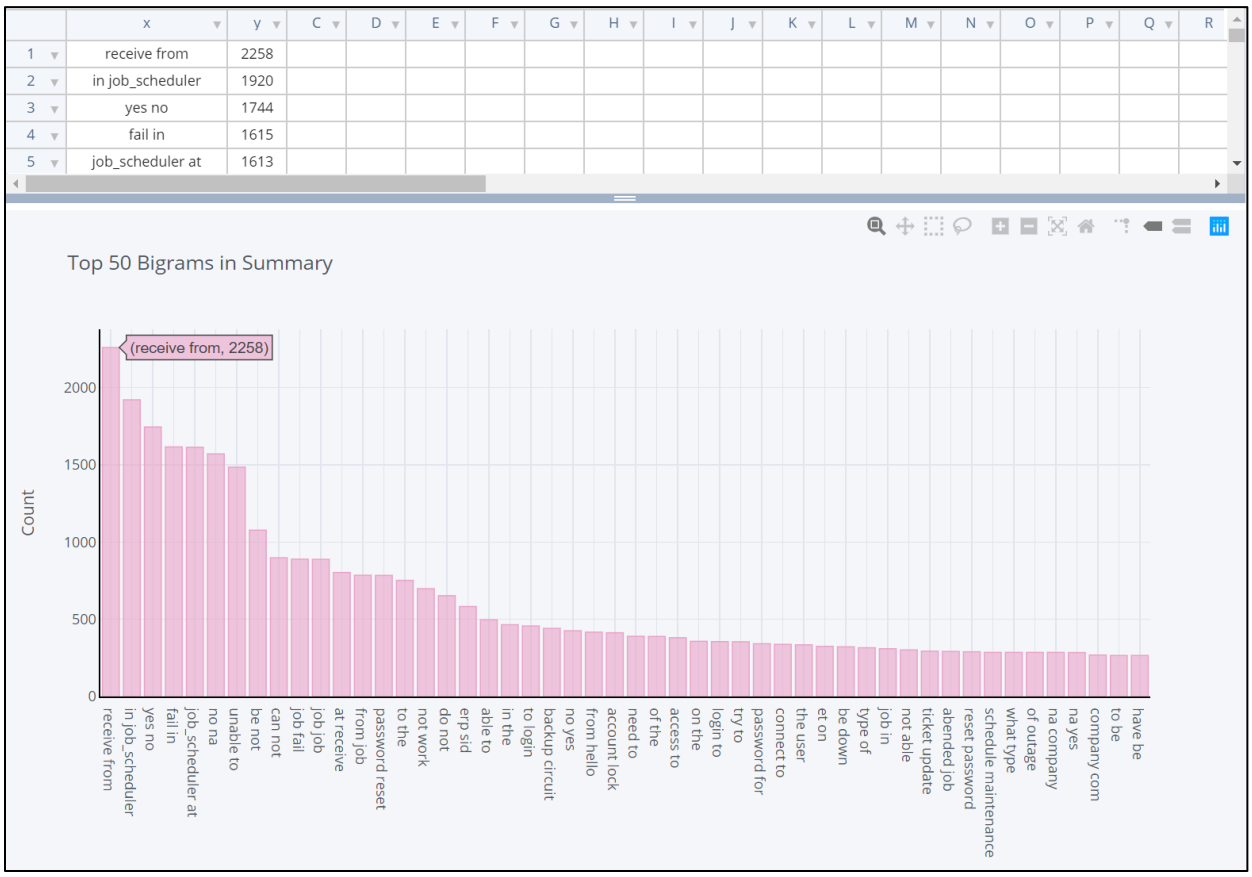


Top 50 Bigrams before removing Stop Words:-

Summary = Short description + Description

```
top_n = 50
ngram_range = (2,2)
bi_grams = get_top_n_ngrams(ticket.Summary, top_n, ngram_range)

df = pd.DataFrame(bi_grams, columns = ['Summary' , 'count'])
df.groupby('Summary').sum()['count'].sort_values(ascending=False).iplot(
    kind='bar',
    yTitle='Count',
    linecolor='black',
    colorscale='piyg',
    title=f'Top {top_n} Bigrams in Summary')
```

	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																

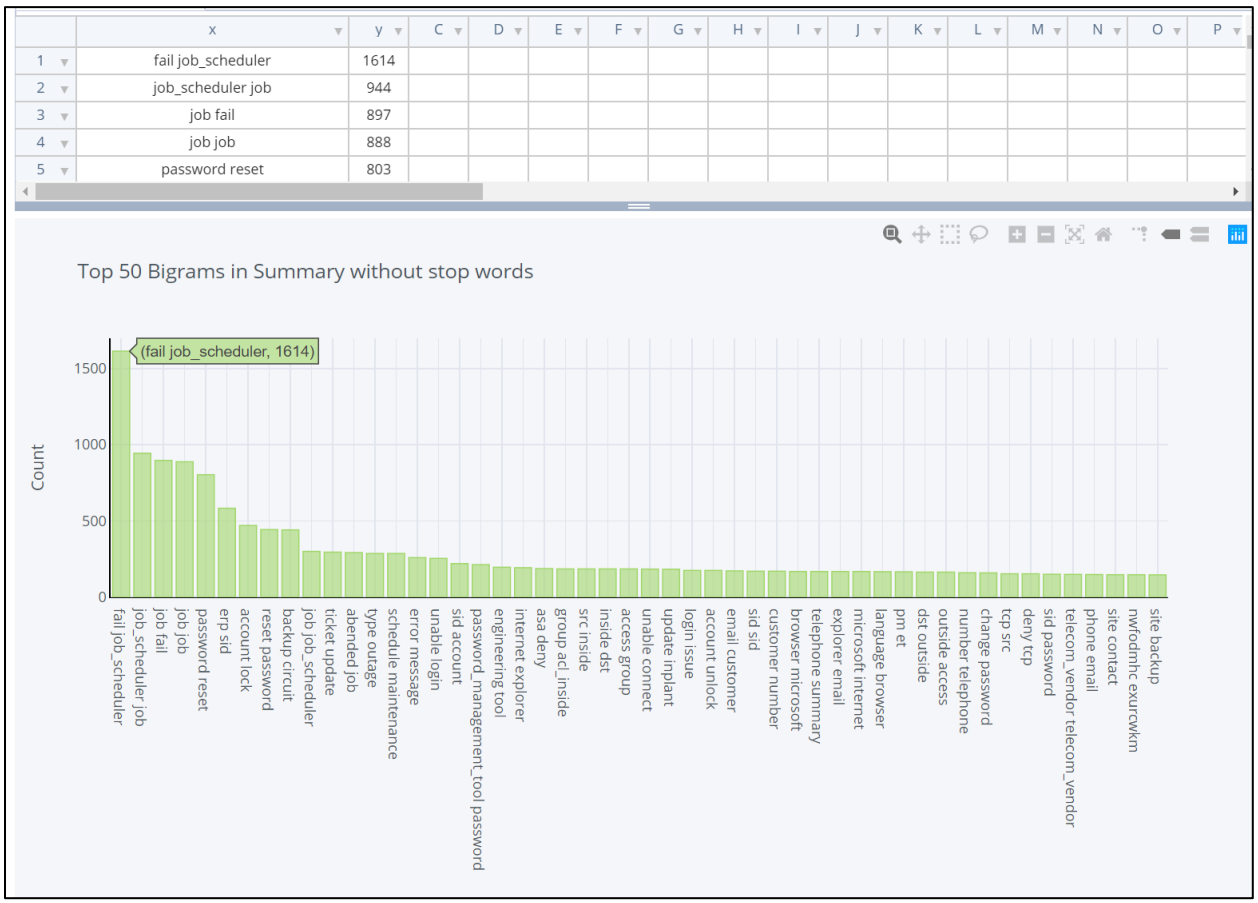


Top 50 Bigrams after removing Stop Words:-

Summary = Short description + Description

```
bi_grams_sw = get_top_n_ngrams(ticket.Summary, top_n, ngram_range, stopwords=STOP_WORDS)

df = pd.DataFrame(bi_grams_sw, columns = ['Summary' , 'count'])
df.groupby('Summary').sum()['count'].sort_values(ascending=False).iplot(
    kind='bar',
    yTitle='Count',
    linecolor='black',
    colorscale='-piyg',
    title=f'Top {top_n} Bigrams in Summary without stop words')
```

	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														

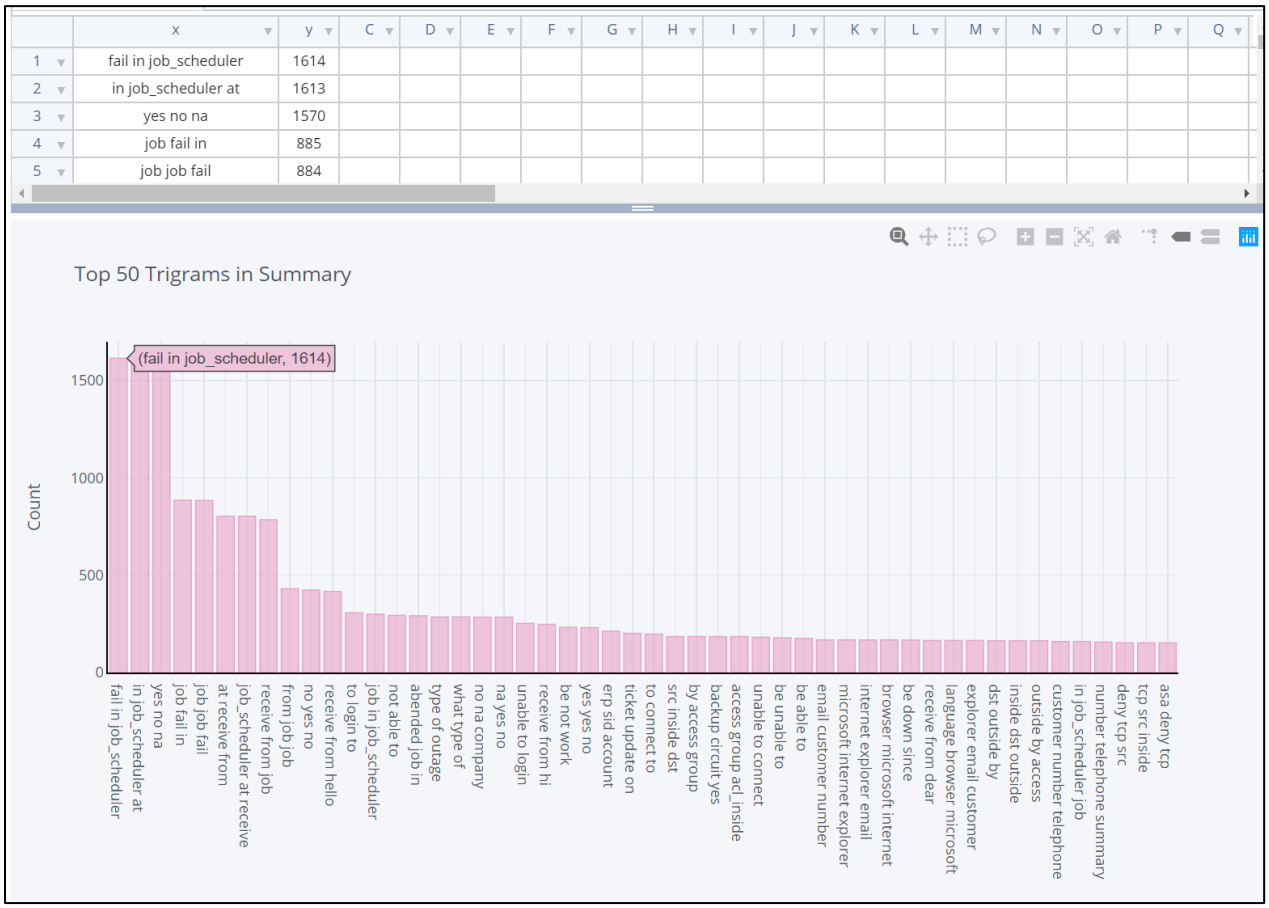


Top 50 Trigrams before removing Stop Words:-

Summary = Short description + Description

```
top_n = 50
ngram_range = (3,3)
tri_grams = get_top_n_ngrams(ticket.Summary, top_n, ngram_range)

df = pd.DataFrame(tri_grams, columns = ['Summary' , 'count'])
df.groupby('Summary').sum()['count'].sort_values(ascending=False).iplot(
    kind='bar',
    yTitle='Count',
    linecolor='black',
    colorscale='piyg',
    title=f'Top {top_n} Trigrams in Summary')
```

	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															

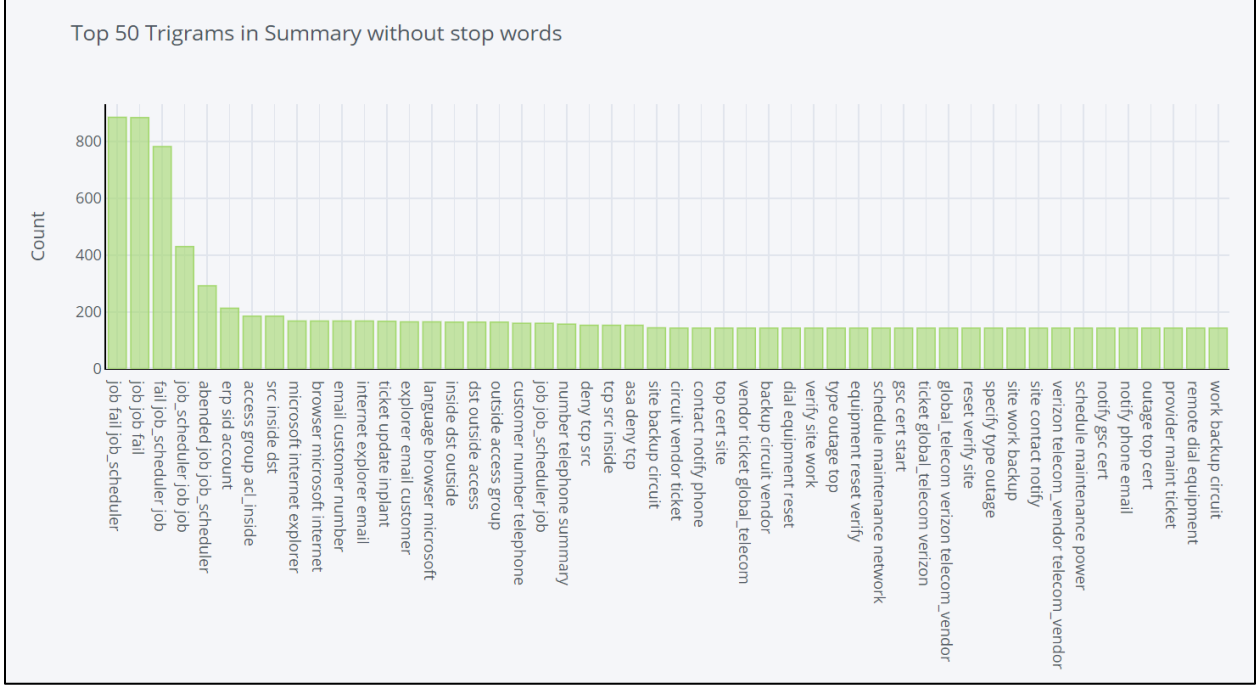


Top 50 Trigrams after removing Stop Words:-

Summary = Short description + Description

```
tri_grams_sw = get_top_n_ngrams(ticket.Summary, top_n, ngram_range, stopwords=STOP_WORDS)

df = pd.DataFrame(tri_grams_sw, columns = ['Summary' , 'count'])
df.groupby('Summary').sum()['count'].sort_values(ascending=False).iplot(
    kind='bar',
    yTitle='Count',
    linecolor='black',
    colorscale='-piyg',
    title=f'Top {top_n} Trigrams in Summary without stop words')
```

	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)**

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.

Generic method to generate Word Clouds for both Short and Long Description columns:-

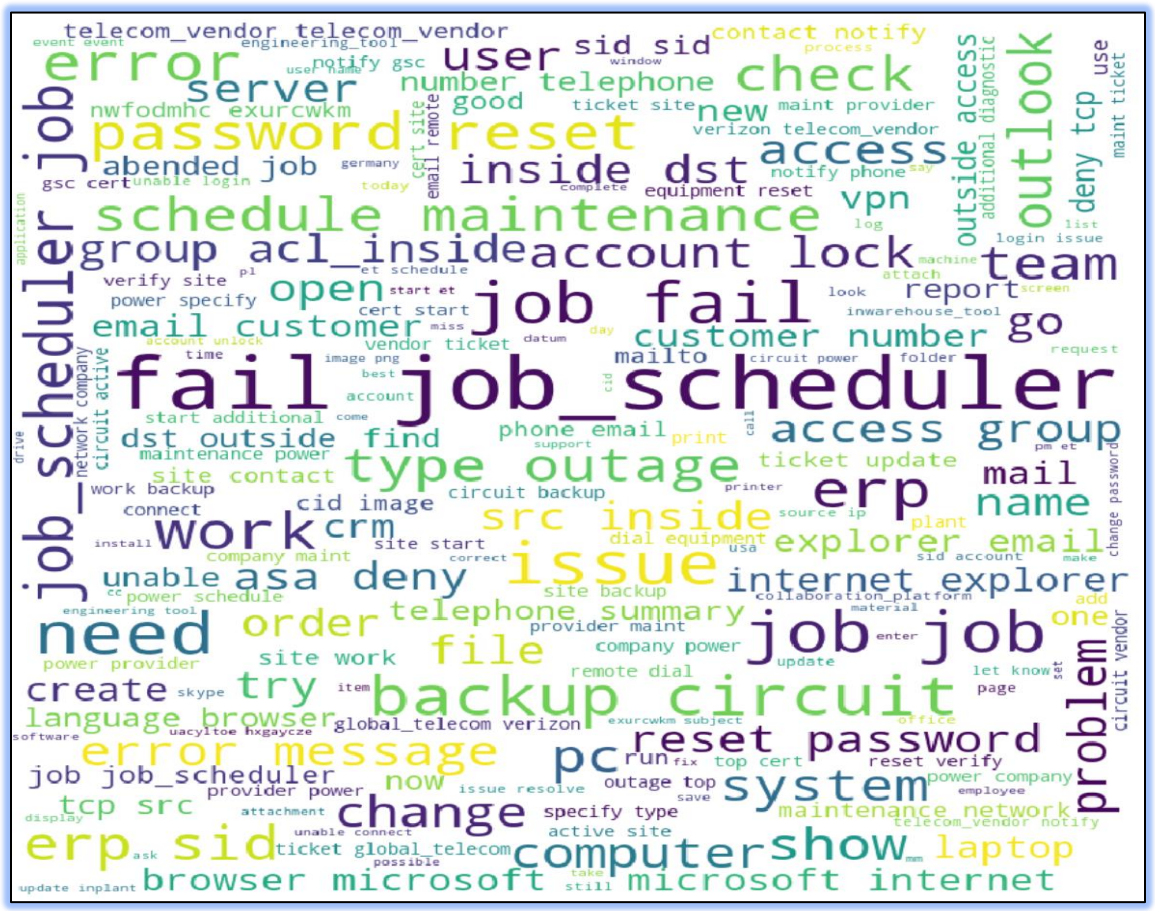
```
def generate_word_clod(corpus):
    # mask = np.array(Image.open('cloud2.png'))
    # Instantiate the wordcloud object
    wordcloud = WordCloud(width = 800, height = 800,
                    background_color ='white',
                    stopwords=STOP_WORDS,
                    # mask=mask,
                    min_font_size = 10).generate(corpus)

    # plot the WordCloud image
    plt.figure(figsize = (12, 12), facecolor = None)
    plt.imshow(wordcloud)
    plt.axis("off")
    plt.tight_layout(pad = 0)

    plt.show()
```

Word Cloud for ticket Short Description: -

```
generate_word_clod(' '.join(ticket['Short description'].str.strip()))
```

Word Cloud for Ticket Description: -

```
generate_word_clod(' '.join(ticket.Description.str.strip()))
```



Word Cloud for combined Ticket Description and Short Description: -

```
generate_word_clod(' '.join(ticket.Summary.str.strip()))
```

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

Word Cloud on combined Ticket Description and Short Description for <u>Group 0</u>: -

```
generate_word_clod(' '.join(ticket[ticket['Assignment group'] == 'GRP_0'].Summary.str.strip()))
```

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.

PREDICTION MODELS

Import Model, Packages, Evaluation Metrics, Visualization tools, Utilities

```
import csv
import keras
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import pandas as pd
import seaborn as sns
import tensorflow as tf

from google.colab import drive
from IPython.core.display import display, HTML
from keras.callbacks import ModelCheckpoint
from keras.layers import Input, Dense, Dropout, Embedding, LSTM, Flatten
from keras.models import Model
from keras.models import Sequential
from keras.preprocessing.sequence import pad_sequences
from keras.preprocessing.text import Tokenizer
from keras.utils import to_categorical
from nltk.corpus import stopwords
from numpy import asarray
from numpy import zeros
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.naive_bayes import MultinomialNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import Pipeline
from sklearn import metrics
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.svm import LinearSVC
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier

#Utilities
from time import time

# Evaluation metrics
from sklearn.metrics import confusion_matrix, classification_report, roc_curve, auc, accuracy_score, precision_recall_curve
```

Generic Class and Methods to evaluate Model Performance:-

```
# A class that logs the time
class Timer():
    '''
    A generic class to log the time
    '''
    def __init__(self):
        self.start_ts = None
    def start(self):
        self.start_ts = time()
    def stop(self):
        return 'Time taken: %2fs' % (time()-self.start_ts)

timer = Timer()


# A method that plots the Precision-Recall curve
def plot_prec_recall_vs_thresh(precisions, recalls, thresholds):
    plt.figure(figsize=(10,5))
    plt.plot(thresholds, precisions[:-1], 'b--', label='precision')
    plt.plot(thresholds, recalls[:-1], 'g--', label = 'recall')
    plt.xlabel('Threshold')
    plt.legend()

# Let's create a generic method to train and test the model
def run_classification(estimator, X_train, X_test, y_train, y_test,pipelineRequired = True,isDeepModel=False):
    timer.start()

    # train the model
    clf = estimator

    if pipelineRequired :
      clf = Pipeline([('vect', CountVectorizer()),
                      ('tfidf', TfidfTransformer()),
                      ('clf', estimator),
                      ])

    if isDeepModel :
      clf.fit(X_train, y_train, validation_data=(X_test, y_test),epochs=10, batch_size=128,verbose=1)
      # predict from the claffier
      y_pred = clf.predict(X_test)
      y_pred = np.argmax(y_pred, axis=1)
      y_train_pred = clf.predict(X_train)
      y_train_pred = np.argmax(y_train_pred, axis=1)
    else :
      clf.fit(X_train, y_train)
      # predict from the claffier
      y_pred = clf.predict(X_test)
      y_train_pred = clf.predict(X_train)


    print('Estimator:', clf)
    print('='*80)
    print('Training accuracy: %.2f%%' % (accuracy_score(y_train,y_train_pred) * 100))
    print('Testing accuracy: %.2f%%' % (accuracy_score(y_test, y_pred) * 100))
    print('='*80)
    print('Confusion matrix:\n %s' % (confusion_matrix(y_test, y_pred)))
    print('='*80)
    print('Classification report:\n %s' % (classification_report(y_test, y_pred)))
    print(timer.stop(), 'to run the model')
```

Split Data into Training and Test Set (Training : 80% of Data, Testing : 20% of Data)

```
X_train, X_test, y_train, y_test = train_test_split(data_content, y, test_size=0.15,random_state=42)
```

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.

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

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

```
run_classification(MultinomialNB(), X_train, X_test, y_train, y_test)
```

```
Estimator: Pipeline(memory=None,
         steps=[('vect',
                 CountVectorizer(analyzer='word', binary=False,
                                 decode_error='strict',
                                 dtype=<class 'numpy.int64'>, encoding='utf-8',
                                 input='content', lowercase=True, max_df=1.0,
                                 max_features=None, min_df=1,
                                 ngram_range=(1, 1), preprocessor=None,
                                 stop_words=None, strip_accents=None,
                                 token_pattern='(?u)\\b\\w\\w+\\b',
                                 tokenizer=None, vocabulary=None)),
                ('tfidf',
                 TfidfTransformer(norm='l2', smooth_idf=True,
                                  sublinear_tf=False, use_idf=True)),
                ('clf',
                 MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True))],
         verbose=False)
================================================================================
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

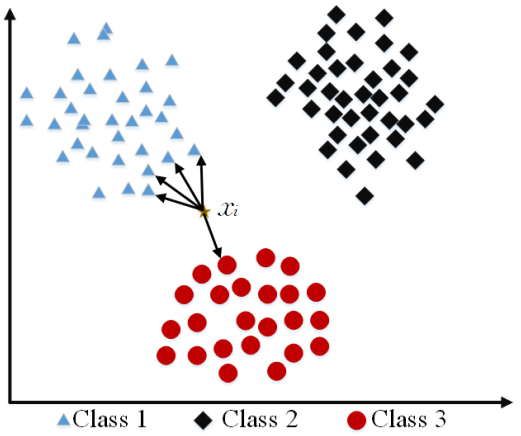
In pattern recognition, the k-nearest neighbors algorithm (k-NN) is a non-parametric method used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space. The output depends on whether k-NN is used for classification or regression:

In k-NN classification, the output is a class membership. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor. In k-NN regression, the output is the property value for the object. This value is the average of the values of k nearest neighbors. 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 neighbors, so that the nearer neighbors contribute more to the average than the more distant ones. For example, a common weighting scheme consists in giving each neighbor a weight of 1/d, where d is the distance to the neighbor.

The neighbors 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.




```
run_classification(KNeighborsClassifier(), X_train, X_test, y_train, y_test)
```

```
Estimator: Pipeline(memory=None,
         steps=[('vect',
                 CountVectorizer(analyzer='word', binary=False,
                                 decode_error='strict',
                                 dtype=<class 'numpy.int64'>, encoding='utf-8',
                                 input='content', lowercase=True, max_df=1.0,
                                 max_features=None, min_df=1,
                                 ngram_range=(1, 1), preprocessor=None,
                                 stop_words=None, strip_accents=None,
                                 token_pattern='(?u)\\b\\w\\w+\\b',
                                 tokenizer=None, vocabulary=None)),
                ('tfidf',
                 TfidfTransformer(norm='l2', smooth_idf=True,
                                  sublinear_tf=False, use_idf=True)),
                ('clf',
                 KNeighborsClassifier(algorithm='auto', leaf_size=30,
                                      metric='minkowski', metric_params=None,
                                      n_jobs=None, n_neighbors=5, p=2,
                                      weights='uniform'))],
         verbose=False)
================================================================================
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.

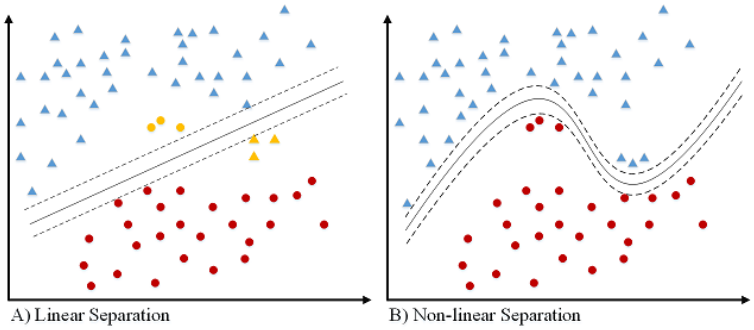
When data are unlabelled, supervised learning is not possible, and an unsupervised learning approach is required, which attempts to find natural clustering of the data to groups, and then map new data to these formed groups. The support-vector clustering algorithm, created by Hava Siegelmann and Vladimir 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).




```
run_classification(LinearSVC(), X_train, X_test, y_train, y_test)

Estimator: Pipeline(memory=None,
         steps=[('vect',
                 CountVectorizer(analyzer='word', binary=False,
                                 decode_error='strict',
                                 dtype=<class 'numpy.int64'>, encoding='utf-8',
                                 input='content', lowercase=True, max_df=1.0,
                                 max_features=None, min_df=1,
                                 ngram_range=(1, 1), preprocessor=None,
                                 stop_words=None, strip_accents=None,
                                 token_pattern='(?u)\\b\\w\\w+\\b',
                                 tokenizer=None, vocabulary=None)),
                ('tfidf',
                 TfidfTransformer(norm='l2', smooth_idf=True,
                                  sublinear_tf=False, use_idf=True)),
                ('clf',
                 LinearSVC(C=1.0, class_weight=None, dual=True,
                           fit_intercept=True, intercept_scaling=1,
                           loss='squared_hinge', max_iter=1000,
                           multi_class='ovr', penalty='l2', random_state=None,
                           tol=0.0001, verbose=0))],
         verbose=False)
================================================================================
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
```

```
# SVM with RBF kernel
run_classification(SVC(kernel='rbf'), X_train, X_test, y_train, y_test)
```

```
Estimator: Pipeline(memory=None,
         steps=[('vect',
                 CountVectorizer(analyzer='word', binary=False,
                                 decode_error='strict',
                                 dtype=<class 'numpy.int64'>, encoding='utf-8',
                                 input='content', lowercase=True, max_df=1.0,
                                 max_features=None, min_df=1,
                                 ngram_range=(1, 1), preprocessor=None,
                                 stop_words=None, strip_accents=None,
                                 token_pattern='(?u)\\b\\w\\w+\\b',
                                 tokenizer=None, vocabulary=None)),
                ('tfidf',
                 TfidfTransformer(norm='l2', smooth_idf=True,
                                  sublinear_tf=False, use_idf=True)),
                ('clf',
                 SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None,
                     coef0=0.0, decision_function_shape='ovr', degree=3,
                     gamma='scale', kernel='rbf', max_iter=-1,
                     probability=False, random_state=None, shrinking=True,
                     tol=0.001, verbose=False))],
         verbose=False)
================================================================================
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.

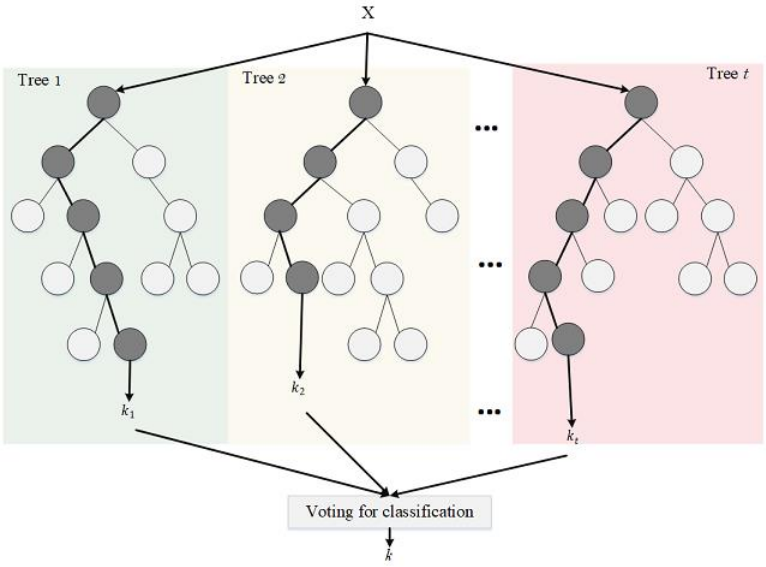
```
run_classification(DecisionTreeClassifier(), X_train, X_test, y_train, y_test)
```

```
Estimator: Pipeline(memory=None,
         steps=[('vect',
                 CountVectorizer(analyzer='word', binary=False,
                                 decode_error='strict',
                                 dtype=<class 'numpy.int64'>, encoding='utf-8',
                                 input='content', lowercase=True, max_df=1.0,
                                 max_features=None, min_df=1,
                                 ngram_range=(1, 1), preprocessor=None,
                                 stop_words=None, strip_accents=None,
                                 token_pattern='(?u)\\b\\w\\w+\\b',
                                 tokenizer=None, vocabulary=Non...
                                  sublinear_tf=False, use_idf=True)),
                ('clf',
                 DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None,
                                        criterion='gini', max_depth=None,
                                        max_features=None, max_leaf_nodes=None,
                                        min_impurity_decrease=0.0,
                                        min_impurity_split=None,
                                        min_samples_leaf=1, min_samples_split=2,
                                        min_weight_fraction_leaf=0.0,
                                        presort='deprecated', random_state=None,
                                        splitter='best'))],
         verbose=False)
================================================================================
Training accuracy: 95.59%
Testing accuracy: 56.65%
================================================================================
Confusion matrix:
 [[633   0   1 ...   1   5   3]
 [  1   1   0 ...   0   1   0]
 [  8   0   9 ...   0   0   1]
 ...
 [  1   0   0 ...   0   0   0]
 [  1   1   2 ...   0  92   8]
 [ 10   0   0 ...   0  28   9]]
================================================================================
```

```
------------------------------------------------------------
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



```
run_classification(RandomForestClassifier(n_estimators=100), X_train, X_test, y_train, y_test)
```

```
Estimator: Pipeline(memory=None,
         steps=[('vect',
                 CountVectorizer(analyzer='word', binary=False,
                                 decode_error='strict',
                                 dtype=<class 'numpy.int64'>, encoding='utf-8',
                                 input='content', lowercase=True, max_df=1.0,
                                 max_features=None, min_df=1,
                                 ngram_range=(1, 1), preprocessor=None,
                                 stop_words=None, strip_accents=None,
                                 token_pattern='(?u)\\b\\w\\w+\\b',
                                 tokenizer=None, vocabulary=Non...
                 RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                        class_weight=None, criterion='gini',
                                        max_depth=None, max_features='auto',
                                        max_leaf_nodes=None, max_samples=None,
                                        min_impurity_decrease=0.0,
                                        min_impurity_split=None,
                                        min_samples_leaf=1, min_samples_split=2,
                                        min_weight_fraction_leaf=0.0,
                                        n_estimators=100, n_jobs=None,
                                        oob_score=False, random_state=None,
                                        verbose=0, warm_start=False))],
         verbose=False)
================================================================================
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

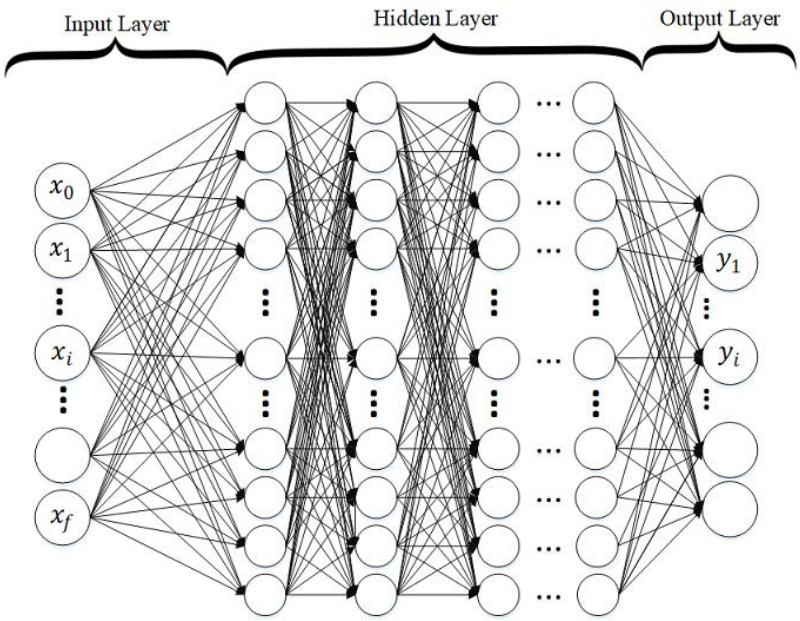
A Neural Network, unlike statistical ML algorithms, is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. Neural networks can adapt to changing input; so the network generates the best possible result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

Define a function for checkpoints

```
#Path where you want to save the weights, model and checkpoints
model_path = "Weights/"
%mkdir Weights

# Define model callbacks
def call_backs(name):
    early_stopping = EarlyStopping(monitor='val_loss', min_delta=0.005, patience=100)
    model_checkpoint =  ModelCheckpoint(model_path + name + '_epoch{epoch:02d}_loss{val_loss:.4f}.h5',
                                        monitor='val_loss',
                                        verbose=1,
                                        save_best_only=True,
                                        save_weights_only=False,
                                        mode='min',
                                        period=1)
    return [model_checkpoint, early_stopping]
```

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


Input Layer
Hidden Layer
Output Layer
x_0
x_1
x_i
x_f
y_1
y_i

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

```
Tfidf_vect = TfidfVectorizer(max_features=5000)
Tfidf_vect.fit(ticket.Summary)
X_train_tfidf = Tfidf_vect.transform(X_train)
X_test_tfidf = Tfidf_vect.transform(X_test)

# Instantiate the network
model_DNN = Build_Model_DNN_Text(X_train_tfidf.shape[1], 75)
```

```
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
```

```
model_DNN.fit(X_train_tfidf, y_train,
                              validation_data=(X_test_tfidf, y_test),
                              callbacks=call_backs("NN"),
                              epochs=10,
                              batch_size=128,
                              verbose=2)
predicted = model_DNN.predict(X_test_tfidf)
```

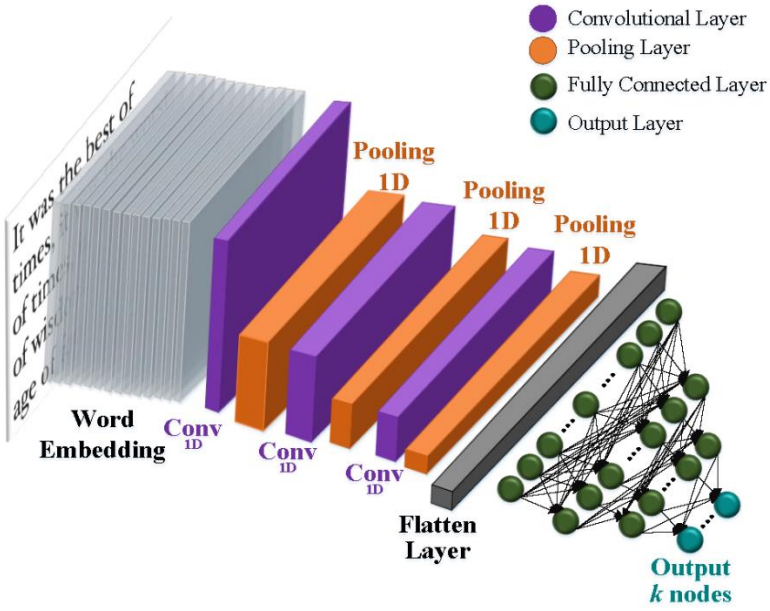
```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:223: The name tf.variables_i
nitializer is deprecated. Please use tf.compat.v1.variables_initializer instead.

 - 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.



```
gloveFileName = 'glove.6B/glove.6B.200d.txt'
MAX_SEQUENCE_LENGTH = 500
EMBEDDING_DIM=200
MAX_NB_WORDS=75000

# Function to generate Embedding
def loadData_Tokenizer(X_train, X_test,filename):
    np.random.seed(7)
    text = np.concatenate((X_train, X_test), axis=0)
    text = np.array(text)
    tokenizer = Tokenizer(num_words=MAX_NB_WORDS)
    tokenizer.fit_on_texts(text)
    sequences = tokenizer.texts_to_sequences(text)
    word_index = tokenizer.word_index
    text = pad_sequences(sequences, maxlen=MAX_SEQUENCE_LENGTH)
    print('Found %s unique tokens.' % len(word_index))
    indices = np.arange(text.shape[0])
    # np.random.shuffle(indices)
    text = text[indices]
    print(text.shape)
    X_train = text[0:len(X_train), ]
    X_test = text[len(X_train):, ]
    embeddings_index = {}
    f = open(filename, encoding="utf8")
    for line in f:
        values = line.split()
        word = values[0]
        try:
            coefs = np.asarray(values[1:], dtype='float32')
        except:
            pass
        embeddings_index[word] = coefs
    f.close()
    print('Total %s word vectors.' % len(embeddings_index))
    return (X_train, X_test, word_index,embeddings_index)


embedding_matrix = []

def buildEmbed_matrices(word_index,embedding_dim):
    embedding_matrix = np.random.random((len(word_index) + 1, embedding_dim))
    for word, i in word_index.items():
        embedding_vector = embeddings_index.get(word)
        if embedding_vector is not None:
            # words not found in embedding index will be all-zeros.
            if len(embedding_matrix[i]) !=len(embedding_vector):
                print("could not broadcast input array from shape",str(len(embedding_matrix[i])), "into shape",str(len(embedding
_vector)),
                      " Please make sure your"" EMBEDDING_DIM is equal to embedding_vector file ,GloVe,")
                exit(1)
            embedding_matrix[i] = embedding_vector
    return embedding_matrix
```

```
# Generate Glove embedded datasets
X_train_Glove, X_test_Glove, word_index, embeddings_index = loadData_Tokenizer(X_train,X_test,gloveFileName)
embedding_matrix = buildEmbed_matrices(word_index,EMBEDDING_DIM)
```

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

```
def Build_Model_CNN_Text(word_index, embeddings_matrix, nclasses,dropout=0.5):
    """
        def buildModel_CNN(word_index, embeddings_index, nclasses, MAX_SEQUENCE_LENGTH=500, EMBEDDING_DIM=50, dropout=0.5):
        word_index in word index ,
        embeddings_index is embeddings index, look at data_helper.py
        nClasses is number of classes,
        MAX_SEQUENCE_LENGTH is maximum lenght of text sequences,
        EMBEDDING_DIM is an int value for dimention of word embedding look at data_helper.py
    """
    model = Sequential()
    embedding_layer = Embedding(len(word_index) + 1,
                                EMBEDDING_DIM,
                                weights=[embeddings_matrix],
                                input_length=MAX_SEQUENCE_LENGTH,
                                trainable=True)
    # applying a more complex convolutional approach
    convs = []
    filter_sizes = []
    layer = 5
    print("Filter  ",layer)
    for fl in range(0,layer):
        filter_sizes.append((fl+2))
    node = 128
    sequence_input = Input(shape=(MAX_SEQUENCE_LENGTH,), dtype='int32')
    embedded_sequences = embedding_layer(sequence_input)
    for fsz in filter_sizes:
        l_conv = Conv1D(node, kernel_size=fsz, activation='relu')(embedded_sequences)
        l_pool = MaxPooling1D(5)(l_conv)
        #l_pool = Dropout(0.25)(l_pool)
        convs.append(l_pool)
    l_merge = Concatenate(axis=1)(convs)
    l_cov1 = Conv1D(node, 5, activation='relu')(l_merge)
    l_cov1 = Dropout(dropout)(l_cov1)
    l_batch1 = BatchNormalization()(l_cov1)
    l_pool1 = MaxPooling1D(5)(l_batch1)
    l_cov2 = Conv1D(node, 5, activation='relu')(l_pool1)
    l_cov2 = Dropout(dropout)(l_cov2)
    l_batch2 = BatchNormalization()(l_cov2)
    l_pool2 = MaxPooling1D(30)(l_batch2)
    l_flat = Flatten()(l_pool2)
    l_dense = Dense(1024, activation='relu')(l_flat)
    l_dense = Dropout(dropout)(l_dense)
    l_dense = Dense(512, activation='relu')(l_dense)
    l_dense = Dropout(dropout)(l_dense)
    preds = Dense(nclasses, activation='softmax')(l_dense)
    model = Model(sequence_input, preds)
    model.compile(loss='sparse_categorical_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy'])

    print(model.summary())
    return model
```

```
# Train the network and run classification
model_CNN = Build_Model_CNN_Text(word_index,embedding_matrix, 75)
run_classification(model_CNN, X_train_Glove, X_test_Glove, y_train, y_test,pipelineRequired = False,isDeepModel=True, arch_name
='CNN')
```

```
Filter   5
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:4267: The name tf.nn.max_poo
l is deprecated. Please use tf.nn.max_pool2d instead.

Model: "model_1"
```

```
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).

```
def Build_Model_RNN_Text(word_index, embeddings_matrix, nclasses,dropout=0.5):
    """
    def buildModel_RNN(word_index, embeddings_matrix, nclasses,  MAX_SEQUENCE_LENGTH=500, EMBEDDING_DIM=100, dropout=0.5):
    word_index in word index ,
    embeddings_matrix is embeddings_matrix, look at data_helper.py
    nClasses is number of classes,
    MAX_SEQUENCE_LENGTH is maximum lenght of text sequences
    """
    model = Sequential()
    hidden_layer = 3
    gru_node = 32

    model.add(Embedding(len(word_index) + 1,
                                EMBEDDING_DIM,
                                weights=[embeddings_matrix],
                                input_length=MAX_SEQUENCE_LENGTH,
                                trainable=True))
    print(gru_node)
    for i in range(0,hidden_layer):
        model.add(GRU(gru_node,return_sequences=True, recurrent_dropout=0.2))
        model.add(Dropout(dropout))
        model.add(BatchNormalization())
    model.add(GRU(gru_node, recurrent_dropout=0.2))
    model.add(Dropout(dropout))
    model.add(BatchNormalization())
    model.add(Dense(256, activation='relu'))
    model.add(BatchNormalization())
    model.add(Dense(nclasses, activation='softmax'))
    model.compile(loss='sparse_categorical_crossentropy',
                      optimizer='sgd',
                      metrics=['accuracy'])

    print(model.summary())
    return model
```

```
# Train the network and run classification
model_RNN = Build_Model_RNN_Text(word_index,embedding_matrix, 75)
run_classification(model_RNN, X_train_Glove, X_test_Glove, y_train, y_test,pipelineRequired = False,isDeepModel=True, arch_name
='RNN')
```

```
32
Model: "sequential_3"
```

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.

```
def Build_Model_RCNN_Text(word_index, embeddings_matrix, nclasses):
    kernel_size = 2
    filters = 256
    pool_size = 2
    gru_node = 256

    model = Sequential()
    model.add(Embedding(len(word_index) + 1,
                                EMBEDDING_DIM,
                                weights=[embeddings_matrix],
                                input_length=MAX_SEQUENCE_LENGTH,
                                trainable=True))
    model.add(Dropout(0.25))
    model.add(BatchNormalization())
    model.add(Conv1D(filters, kernel_size, activation='relu'))
    model.add(MaxPooling1D(pool_size=pool_size))
    model.add(Conv1D(filters, kernel_size, activation='relu'))
    model.add(MaxPooling1D(pool_size=pool_size))
    model.add(Conv1D(filters, kernel_size, activation='relu'))
    model.add(MaxPooling1D(pool_size=pool_size))
    model.add(Conv1D(filters, kernel_size, activation='relu'))
    model.add(MaxPooling1D(pool_size=pool_size))
    model.add(LSTM(gru_node, return_sequences=True, recurrent_dropout=0.2))
    model.add(Dropout(0.25))
    model.add(BatchNormalization())
    model.add(LSTM(gru_node, return_sequences=True, recurrent_dropout=0.2))
    model.add(Dropout(0.25))
    model.add(BatchNormalization())
    model.add(LSTM(gru_node, return_sequences=True, recurrent_dropout=0.2))
    model.add(Dropout(0.25))
    model.add(BatchNormalization())
    model.add(LSTM(gru_node, recurrent_dropout=0.2))
    model.add(Dropout(0.25))
    model.add(BatchNormalization())
    model.add(Dense(1024,activation='relu'))
    model.add(Dense(nclasses))
    model.add(Activation('softmax'))
    model.compile(loss='sparse_categorical_crossentropy',
                  optimizer='sgd',
                  metrics=['accuracy'])

    print(model.summary())
    return model
```

```
# Train the network and run classification
model_RCNN = Build_Model_CNN_Text(word_index,embedding_matrix, 75)
run_classification(model_RCNN, X_train_Glove, X_test_Glove, y_train, y_test,pipelineRequired = False,isDeepModel=True, arch_name
='RCNN')
```

```
Filter   5
Model: "model_2"
```

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

]

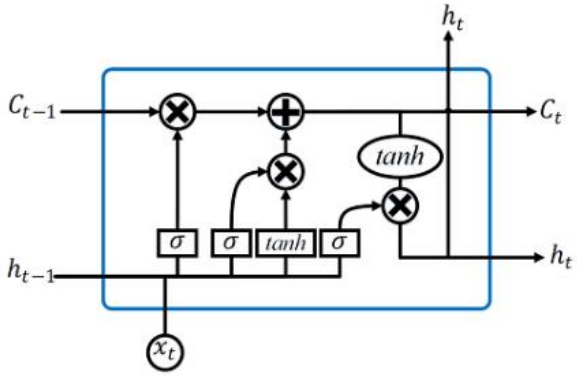
```
_________________________________________________________________________________________________
None
Train on 6800 samples, validate on 1700 samples
Epoch 1/10
6800/6800 [==============================] - 6s 881us/step - loss: 3.0501 - acc: 0.4438 - val_loss: 3.5348 - val_acc: 0.4476

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.



```
EMBEDDING_DIM = 100
gloveFileName = 'glove.6B/glove.6B.100d.txt'

from keras.models import Sequential
from keras.layers import Dense, LSTM, TimeDistributed, Activation
from keras.layers import Flatten, Permute, merge, Input
from keras.layers import Embedding
from keras.models import Model
from keras.layers import Input, Dense, multiply, concatenate, Dropout
from keras.layers import GRU, Bidirectional


def Build_Model_LTSM_Text(word_index, embeddings_matrix, nclasses):
    kernel_size = 2
    filters = 256
    pool_size = 2
    gru_node = 256

    model = Sequential()
    model.add(Embedding(len(word_index) + 1,
                                EMBEDDING_DIM,
                                weights=[embeddings_matrix],
                                input_length=MAX_SEQUENCE_LENGTH,
                                trainable=True))
    model.add(Dropout(0.25))
    model.add(Conv1D(filters, kernel_size, activation='relu'))
    model.add(MaxPooling1D(pool_size=pool_size))
    model.add(Conv1D(filters, kernel_size, activation='relu'))
    model.add(MaxPooling1D(pool_size=pool_size))
    model.add(Conv1D(filters, kernel_size, activation='relu'))
    model.add(MaxPooling1D(pool_size=pool_size))
    model.add(Conv1D(filters, kernel_size, activation='relu'))
    model.add(MaxPooling1D(pool_size=pool_size))
    model.add(Bidirectional(LSTM(gru_node, return_sequences=True, recurrent_dropout=0.2)))
    model.add(Bidirectional(LSTM(gru_node, return_sequences=True, recurrent_dropout=0.2)))
    model.add(Bidirectional(LSTM(gru_node, return_sequences=True, recurrent_dropout=0.2)))
    model.add(Bidirectional(LSTM(gru_node, recurrent_dropout=0.2)))
    model.add(Dense(1024,activation='relu'))
    model.add(Dense(nclasses))
    model.add(Activation('softmax'))
    model.compile(loss='sparse_categorical_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy'])

    print(model.summary())
    return model
```

```
X_train_Glove,X_test_Glove, word_index,embeddings_index = loadData_Tokenizer(X_train,X_test,gloveFileName)
embedding_matrix = buildEmbed_matrices(word_index,EMBEDDING_DIM)

model_LTSM = Build_Model_LTSM_Text(word_index,embedding_matrix, 75)
run_classification(model_LTSM, X_train_Glove, X_test_Glove, y_train, y_test,pipelineRequired = False,isDeepModel=True, arch_name
='LSTM')
```

```
Found 15207 unique tokens.
(8500, 500)
Total 400000 word vectors.
Model: "sequential_5"
```

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

MODEL PERFORMANCE

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

ADVICE FOR BUSINESS

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.

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

Considering SOP review take 15 minutes for an Incident, 8500 Incidents require 2125 Person Hours. Automated Ticket Assignment can save 15 minutes per Incident.

REFERENCES

- Text-Preprocessing – Translate - https://pypi.org/project/googletrans/
- Data Visualization - https://www.datacamp.com/community/tutorials/wordcloud-python
- NLP - https://towardsdatascience.com/machine-learning-nlp-text-classification-using-scikit-learn-python-and-nltk-c52b92a7c73a
- NLP - https://dzone.com/articles/nlp-tutorial-using-python-nltk-simple-examples
- Text Classification - https://www.analyticsvidhya.com/blog/2018/04/a-comprehensive-guide-to-understand-and-implement-text-classification-in-python/
- Text Classification - https://machinelearningmastery.com/develop-n-gram-multichannel-convolutional-neural-network-sentiment-analysis/
- Text Preprocessing – Translate - https://stackoverflow.com/questions/49497391/googletrans-api-error-expecting-value-line-1-column-1-char-0
- NLP - https://github.com/aboSamoor/polyglot
- Text Processing - https://medium.com/starschema-blog/text-preprocessing-in-different-languages-for-natural-language-processing-in-python-fb106f70b554
- Text Processing - https://www.kdnuggets.com/2018/11/text-preprocessing-python.html
- NLP - https://towardsdatascience.com/nlp-text-preprocessing-a-practical-guide-and-template-d80874676e79
- Text Preprocessing - https://www.geeksforgeeks.org/text-preprocessing-in-python-set-1/
- NLP - https://mlwhiz.com/blog/2019/01/17/deeplearning_nlp_preprocess/
- NLP-EDA - https://www.kaggle.com/wil2210/eda-nlp-ml
- Text Preprocessing-EDA - https://towardsdatascience.com/a-complete-exploratory-data-analysis-and-visualization-for-text-data-29fb1b96fb6a
- Translate - https://pypi.org/project/translate/
- Sample Projects
    - https://towardsdatascience.com/it-support-ticket-classification-and-deployment-using-machine-learning-and-aws-lambda-8ef8b82643b6
    - https://worldconferences.net/proceedings/aics2014/PAPER%20AICS/A044%20-%20MACHINE%20LEARNING%20BASED%20TICKET%20CLASSIFICATION%20IN%20ISSUE%20TRACKING%20SYSTEMS%20-mucahit.pdf
    - https://arxiv.org/pdf/1808.02636.pdf
    - https://www.kaggle.com/rohitayinaparthy/text-analytics-using-nlp-nltk-it-incidents
    - https://github.com/karolzak/support-tickets-classification