National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Sejal Shah
Student ID: x18196292

School of Computing
National College of Ireland

Supervisor: Dr. Muhammad Iqbal

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Sejal Shah
Student ID:	x18196292
Programme:	Data Analytics
Year:	2018
Module:	MSc Research Project
Supervisor:	Dr. Muhammad Iqbal
Submission Due Date:	17 August 2020
Project Title:	Configuration Manual
Word Count:	1031
Page Count:	19

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	17th August 2020

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Sejal Shah
x18196292
MSc in Data Analytics

17 August 2020

1 Introduction

This configuration manual would present the system setup,hardware and software requirement. Also, the codes which have been used in programming for implementing below research work:

''Automatic Ticket Assignment using Machine Learning and Deep Learning Techniques"

2 System Configuration

2.1 Hardware

- **Processor:** Intel(R) Core(TM) i5-8300H CPU @ 2.30 GHz 2.30 GHz
- **GPU:** NVIDIA GeForce GTX 1050Ti
- **RAM:** 8GB
- **Storage:** 1 TB HDD
- **Operting System:** Windows 10, 64-bit

2.2 Software

- **Google Collaboratory:**A free cloud service which provides free service on GPU for running machine learning and deep learning models on its environment. Also, for hyper parameter tuning and evaluation.
- **Microsoft Excel 2016:** used for data storing and data ploting.
- **Jupyter Notebook by Anaconda Distribution:** An open source software which has been used for Exploratory Data Analysis and data visualization using python verson 3.6.5 on jupyter notebook in this research.

3 Project Development

Python programming language has been used for implementation of this research study. Firstly, Data Analysis, Data pre-processing stage includes detecting and fixing mojibake, translation, data cleaning, lemmatization. Followed by data preparation and applying classification models.Also, K-fold cross validation has been applied on the machine learning models used in this research study.

In the next section codes develop for this research study has been depicted in detail.

3.1 Data collection

The data has been downloaded from public data repository kaggle [1] and loaded into excel for further exploration.

3.1.1 Importing libraries into colab

```
pip install plotly wordcloud ftfy langdetect iso-language-codes #Installing additional packages
```

```
Requirement already satisfied: plotly in /usr/local/lib/python3.6/dist-packages (4.4.1)
Requirement already satisfied: wordcloud in /usr/local/lib/python3.6/dist-packages (1.5.0)
Requirement already satisfied: ftfy in /usr/local/lib/python3.6/dist-packages (5.8)
Requirement already satisfied: langdetect in /usr/local/lib/python3.6/dist-packages (1.0.8)
Requirement already satisfied: iso-language-codes in /usr/local/lib/python3.6/dist-packages (1.1.0)
Requirement already satisfied: retrying>=1.3.3 in /usr/local/lib/python3.6/dist-packages (from plotly) (1.3.3)
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (from plotly) (1.15.0)
Requirement already satisfied: numpy>=1.6.1 in /usr/local/lib/python3.6/dist-packages (from wordcloud) (1.18.5)
Requirement already satisfied: pillow in /usr/local/lib/python3.6/dist-packages (from wordcloud) (7.0.0)
Requirement already satisfied: wcwidth in /usr/local/lib/python3.6/dist-packages (from ftfy) (0.2.5)
```

Figure 1: Installation of Libraries

```
import pandas as pd #import pandas
import numpy as np #import numpy
from ftfy import fix_encoding, fix_text, badness #import ftfy
from wordcloud import WordCloud, STOPWORDS#import word Colud
import matplotlib.pyplot as plt #import pyplot
import seaborn as sns #import seaborn
import plotly as py #import plotly
import plotly.graph_objs as go #import plotly graph object
from plotly.offline import download_plotlyjs,init_notebook_mode,plot,iplot #import plotly for offline use
from sklearn.pipeline import Pipeline #import pipeline
from sklearn.naive_bayes import MultinomialNB #import multinomial Naive bayes
from sklearn.svm import SVC, LinearSVC #import SVM classes
from sklearn.tree import DecisionTreeClassifier #import Decision Tree
from sklearn.ensemble import RandomForestClassifier #import Random Forest
from sklearn.neighbors import KNeighborsClassifier  # import KNN
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer, TfidfTransformer  # import Countvectorizer
from sklearn.linear_model import LogisticRegression   # import Logistic REgression
import keras.backend as K  #  import keras
from keras.models import Sequential, Model  #  import models
from keras.layers import Input, Dropout, Flatten, Dense, Embedding, LSTM, GRU, concatenate, Bidirectional  #  import layers
from keras.layers import BatchNormalization, TimeDistributed, Conv1D, MaxPooling1D, GlobalMaxPool1D, GlobalMaxPooling1D  #  import layers
from keras.constraints import max_norm, unit_norm  #  import layers
from keras.preprocessing.sequence import pad_sequences  #  import pad sequences
from keras.preprocessing.text import Tokenizer, text_to_word_sequence  #  import tokenizer
from keras.layers.merge import Concatenate  #  import concatenate
from keras.callbacks import EarlyStopping, ModelCheckpoint  #  import callbacks
from keras import optimizers, regularizers  #  import optimizers
from sklearn.model_selection import train_test_split  #  import train test split
from sklearn.metrics import confusion_matrix, classification_report, auc  #  evalution metrics
from sklearn.metrics import roc_curve, accuracy_score, precision_recall_curve  #  evalution metrics
import re #  regular expression
import string # string related manupulations
from spacy.lang.en.stop_words import STOP_WORDS #  import stop words
from nltk.tokenize import word_tokenize #  import word_tokenizer
import nltk #  import nltk
#nltk.download('punkt') #  download corpus
```

Figure 2: Importing required python libraries

[1] https://www.kaggle.com/aviskumar/automatic-ticket-assignment-using-nlp

Refer code from Figure 1 and Figure 2 for all the python libraries which have been used in this research.

4 Data Pre-Processing and EDA

The outcomes of predictive models are dependent on the type of applied data pre-processing technique. Generally, the dataset contains noise (irrelevant data) with information. It is important to minimize the noise for obtaining high quality information from the data. In this research, techniques namely text cleaning, fixing mojibake, translating text, lemmatization and data cleaning have been applied. Furthermore, to transform the data one-hot encoding, TF-IDF vectorization and synthetic attribute construction have been applied which enabled model training.

4.1 Initial analysis of data set

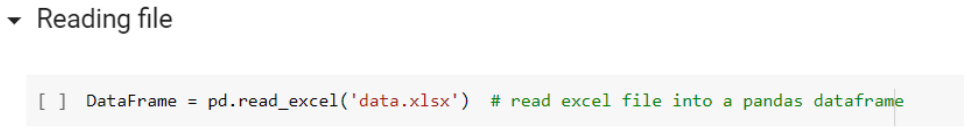

Figure 3: Data loading from excel

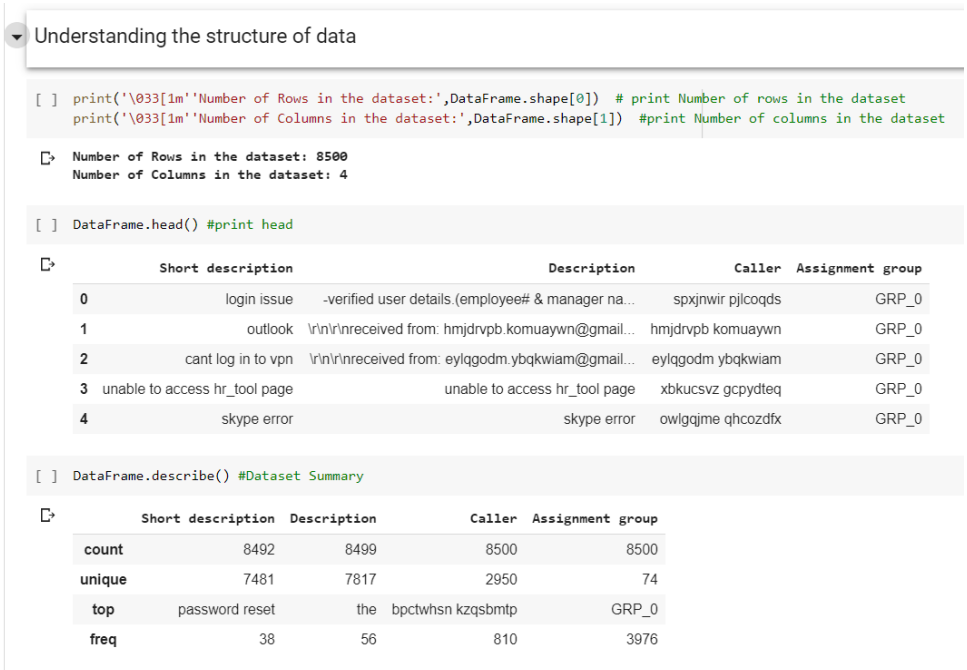

Figure 4: Head and summary of the dataset

Figure 3 depicts the data loading i.e reading of excel file into pandas dataframe whereas, Figure 4 represents top 5 rows and summary of the dataset. From summary it can be seen that 'password reset' issues are majorly faced while an inconsistency in Description column can be observed. Top caller is 'bpctwhsn kzqsbmtp' and 'GRP_0' has been assigned most incidents.

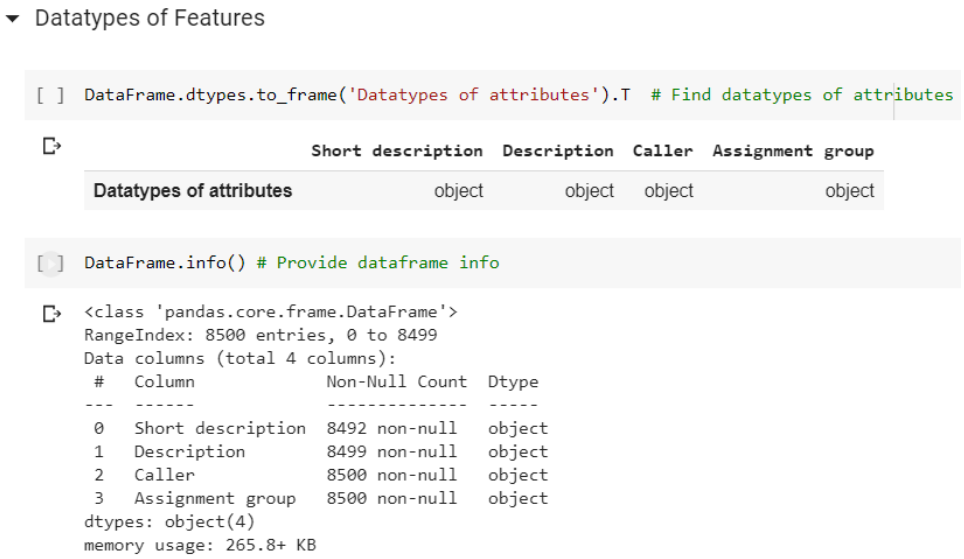

Figure 5: Additional information about dataset

Figure 5 shows that The data set has 4 columns named 'short description', 'description', 'Caller' and 'Assignment group'. The datatype of all features is object.

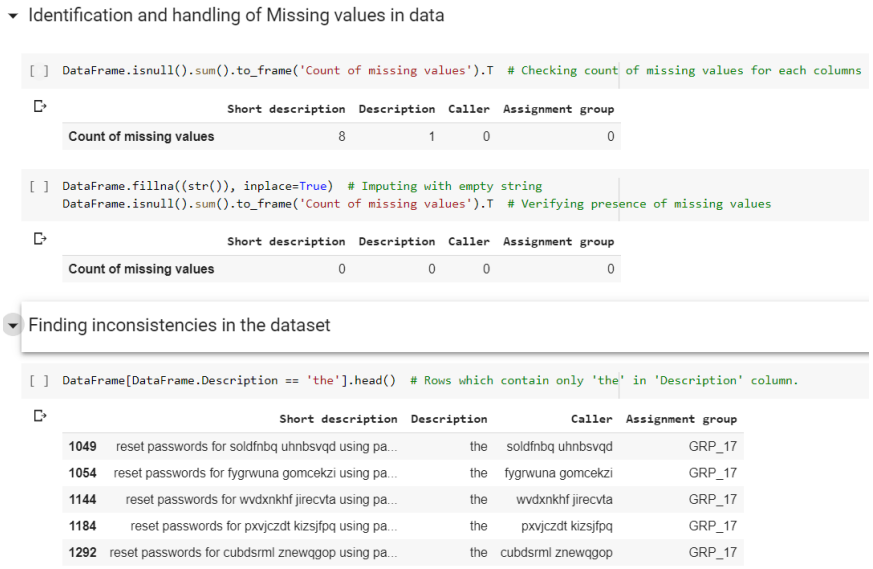

Figure 6: Missing value and inconsistency in dataset

There are 8 missing values in 'Short Description', 1 in 'Description' and no missing values are present in 'Caller' and 'Assignment Group' and one of the inconsistencies can be observed from Figure 6.

Detecting and Fixing Mojibake

```
[ ]  #False = Mojibake Alert, True = The text is not Mojibake affected
     def mojibake_test(data): #Define function for detecting Mojibake
       if badness.sequence_weirdness(data): #check if affected
         return False #return false if text is affected(Mojibake alert)
       try:
         data.encode('windows-1252') #encode if exception is thrown
       except UnicodeEncodeError: #Windows-1252 encodable(Mojibake alert)
         return False
       else: #the text is not affected by Wrong Decoding
         return True
```

Figure 7: Function for mojibake identification

For identification of mojibake, function is written in python which can be observed from Figure 7

Translation

```
[ ]  from langdetect import detect
     from iso_language_codes import *
     def detect_lang(text):
       try:
         tokens = text.split()
         for token in tokens:
           if detect(token) != 'en': # Checking for presence of non english text
             lang = detect(token)
             return language(lang)["Name"]
         return language(detect(text))["Name"] # if no other language present returns english
       except:
         return language('en')["Name"]
     DataFrame['Language Before Translation'] = DataFrame["Description"].apply(detect_lang)
```

```
from google.cloud import translate_v2 as translate
translate_client = translate.Client.from_service_account_json('TicketAssignmentTranslation-4e6e83c85d08.json')
from time import sleep


for index , row in DataFrame.iterrows():
  result = translate_client.translate([row['Short description'],row["Description"]], target_language='en')
  DataFrame["Short description"].iloc[index] = result[0]['translatedText']
  DataFrame["Description"].iloc[index] = result[1]['translatedText']
  print("Translation completed for row {}".format(index))
  sleep(0.5)
```

Figure 8: Function for translation

Figure 8 represents the function written for translation of text. Also, for generation of Google Translation API key refer [2]

[2] https://translatepress.com/docs/automatic-translation/generate-google-api-key/

4.2 Data Cleaning

```
def get_step1_regex_list():
    ''' Removes the mentioned regex text from the document.'''
    regexList = []

    # received from: xxxx
    regexList += ['received from:[\ ]?[a-z0-9]+[\._]?[a-z0-9]+[@]\w+[.]\w{2,3}']
    regexList += ['^received from:[\ ]?.*']
    # from - mail address or names
    regexList += ['^from:[\ ]?.*']
    # to - mail addresses
    regexList += ['^to:[\ ]?.*']
    # cc - mail addresses
    regexList += ['^cc:[\ ]?.*']
    # sent: xxx
    regexList += ['^sent:[\ ]?.*']
    # email: xxxx
    regexList += ['^email:[\ ]?.*']
    # name: xxxx
    regexList += ['^name:[\ ]?.*']
    # customer number: xxxx
    regexList += ['^customer number:[\ ]?.*']
    # browser: xxxx
    regexList += ['^browser:[\ ]?.*']
    # language: xxxx
    regexList += ['^language:[\ ]?.*']
    # user: xxxx
    regexList += ['^user:[\ ]?.*']
    # user id: xxxx
    regexList += ['^user id:[\ ]?.*']
    # login id: xxxx
    regexList += ['^login id:[\ ]?.*']
    # computer: xxxx
    regexList += ['^computer:[\ ]?.*']
    # computer name: xxxx
    regexList += ['^computer name:[\ ]?.*']
    # service tag: xxxx
    regexList += ['^service tag:[\ ]?.*']
    # service tag: xxxx
    regexList += ['^model details:[\ ]?.*']
    # images
    regexList += ['\[cid:(.*)\]']
    # telephone: xxxx
    regexList += ['^telephone[:\ ].*']
    # id: xxxx
    regexList += ['^id[:\ ].*']
    # email
    regexList += ['[a-z0-9]+[\._]?[a-z0-9]+[@]\w+[.]\w{2,3}']
    # timestamp 12/12/2020 12:12:12
    regexList += ['\d{2}/\d{2}/\d{4}[\ ]?\d{2}:\d{2}:\d{2}']
    # timestamp 10/25/2016 1:47 pm
    regexList += ['\d{2}/\d{2}/\d{4}[\ ]?\d{1,2}:\d{1,2}[\ ]?(am|pm)']
    # date 10/25/2016 or 10-25-2010
    regexList += ['\d{1,2}[/-]?\d{1,2}[/-]?\d{4}']
    # time 01:12 am
    regexList += ['\d{1,2}:\d{1,2}[\ ]?(am|pm)']
    # time 12:11:23
    regexList += ['\d{2}:\d{2}:\d{2}']
    # subject
    regexList += ['subject:']
    #summary
    regexList += ['summary:']
    # forward
    regexList += ['(fwd:|fw:|re:)']
    # <mailto: >
    regexList += ['\<mailto.*\>']
    # hostname_xxxx
    # regexList += ['hostname_\d*']
    # ticket number
    regexList += ['ticket_no\d*']
    regexList += ['tck no[\.\d]*']

    return regexList
```

Figure 9: Regular Expression

▾ Data Cleaning

```
[ ]  DataFrame_copy = DataFrame  #copy of original dataframe
     def apply_regex(text, regexList):  # Apply regex to the text using regexList
         ''' Apply list of regex expressions on to text '''
         for regex_ in regexList:
             text = re.sub(regex_, '', text)
         return text  # Return clean text
```

Figure 10: Applied regular expression

In this research study of incident reporting there is no need to concern about emails, or the sender, the main cognizance of language processing here is to concentrate on content and subject of incident ticket, the message it is trying to deliver, that being stated email id's, to, cc, bcc, problem, footers, computer names, ip addresses, numbers all are inappropriate and may be removed using regular expressions expressions which is represented by Figure 9 and Figure 10.

Removal of stop words

▾ Removal of Stop Words

"Stop Words" is a technical term to represent high frequency words which are widely used in a language and corpus which add little or no value in vocabulary. We need to remove one's words before training any of our models. Most of the NLP libraries obtainable has a well-known set of stop words defined, in our case we are going to use "NLTK's" stop words.

```
[ ]  import nltk
     nltk.download('punkt')

     def remove_stop_words(text):  # Function to remove stop words
       return ' '.join([word for word in word_tokenize(text) if word not in STOP_WORDS])
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

```
[ ]  DataFrame['Clean Description'] = DataFrame['Clean Description'].apply(lambda x: remove_stop_words(x))
     DataFrame['Clean Short Description'] = DataFrame['Clean Short Description'].apply(lambda x: remove_stop_words(x))
```

Figure 11: Removing of stop words

Stop words removal is necessary before any of the model Figure 11 explains the function return for stop words.

Data Normalization

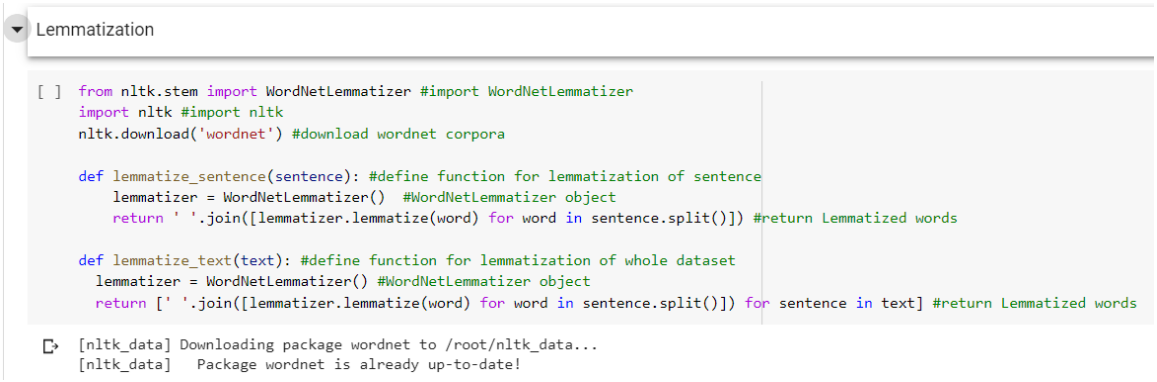

Figure 12: Lemmatization code

Data Normalization has been performed using lemmatization to get root word, code has shown in Figure 12. [3]

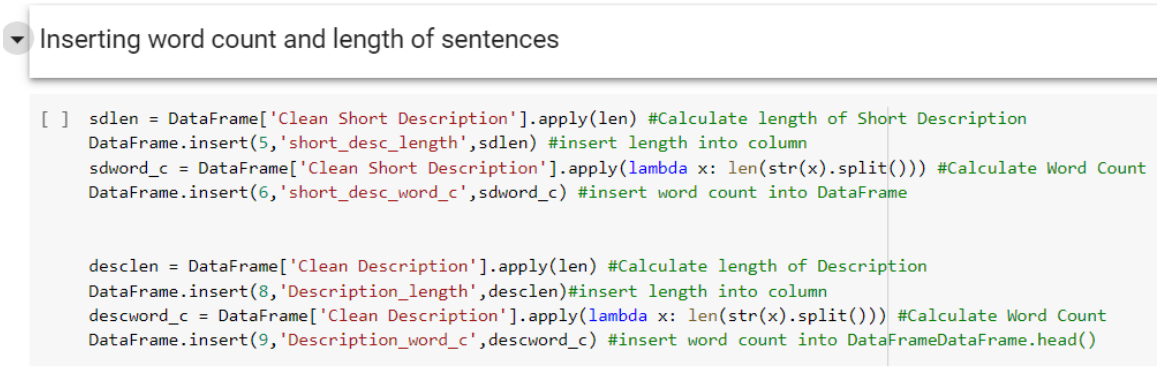

Figure 13: Word count and length

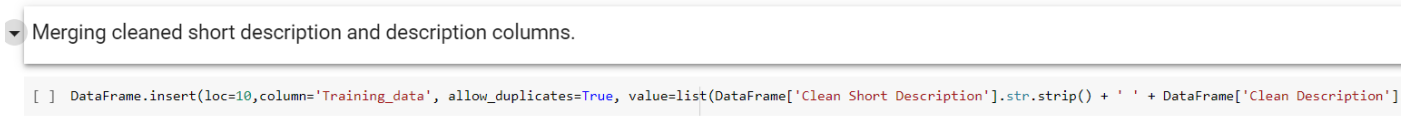

Figure 14: Short description and description merged

[3] https://www.tutorialspoint.com/natural_language_toolkit/natural_language_toolkit_stemming_lemmatization.htm#:~:text=Difference%20between%20Stemming%20%26%20Lemmatization&text=In%20simple%20words%2C%20stemming%20technique,always%20get%20a%20valid%20word

Figure 15: Target column for training

Adding word count and length, merging of short description and description column and creating target column from 'Assignment_ group' has been explained in Figure 13, Figure 14 and Figure 15. Also, Dataset summary after pre-processing has been shown in Figure 16.

```
DataFrame.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8500 entries, 0 to 8499
Data columns (total 13 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   Short description           8500 non-null   object
 1   Description                 8500 non-null   object
 2   Caller                      8500 non-null   object
 3   Assignment group            8500 non-null   object
 4   Language Before Translation 8500 non-null   object
 5   short_desc_length           8500 non-null   int64
 6   short_desc_word_c           8500 non-null   int64
 7   Clean Description           8500 non-null   object
 8   Description_length          8500 non-null   int64
 9   Description_word_c          8500 non-null   int64
 10  Training_data               8500 non-null   object
 11  Clean Short Description     8500 non-null   object
 12  Target                      8500 non-null   int8
dtypes: int64(4), int8(1), object(8)
memory usage: 805.3+ KB
```

Figure 16: Dataset summary after pre-processing

Feature selection

▾ PCA and Resampling of dataset

```
from imblearn.over_sampling import ADASYN, SMOTE, SVMSMOTE, RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler
tv = TfidfVectorizer(stop_words=None, max_features=20000)
test_tfidf = tv.fit_transform(DataFrame['Training_data'])
rus = RandomUnderSampler(sampling_strategy={0:600})
ros = RandomOverSampler()
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/externals/six.py:31: FutureWarning:

The module is deprecated in version 0.21 and will be removed in version 0.23 since we've

/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:144: FutureWarning:

The sklearn.neighbors.base module is  deprecated in version 0.22 and will be removed in
```

```
[ ] print('Shape of TFIDF vector:', test_tfidf.shape)
```

```
Shape of TFIDF vector: (8500, 10736)
```

```
[ ] X = pd.DataFrame(test_tfidf.todense(), columns=tv.get_feature_names())
```

Figure 17: Resampling and PCA

▾ Principal Component Analysis (PCA)

```
[ ] from sklearn.preprocessing import StandardScaler
    X = StandardScaler().fit_transform(X)
```

```
[ ] from sklearn.decomposition import PCA
    pca = PCA(0.95)
    X = pca.fit_transform(X)
```

```
[ ] import pickle as pk
    pk.dump(X, open("pca.pkl","wb"))
```

```
[ ] import pickle as pk
    X_pca = pk.load(open("pca.pkl",'rb'))
    X_pca.shape
```

```
(8500, 3468)
```

```
X_under_sample, y_under_sample = rus.fit_resample(X_pca, DataFrame['Target'])
X_over_sample,y_over_sample = ros.fit_resample(X_under_sample, y_under_sample)
```

Figure 18: Principal component Analysis

The use of resampling techniques has been depicted in Figure 17 which at first undersamples and then oversamples the data set to handle the class imbalance. Figure 18 depicts the use of PCA with 95% variance which has reduced the dimensions of data set from 10736 to 3468.

5 Code used for Machine Learning and Deep Learning models

This section consist of code used for modelling which has been done using scikit libraries. Models such as SVM, Random Forest classifier, K-Nearest Neighbours, ANN and BLSTM have been applied.

5.1 Data Splitting

Data has been split into train and test i.e 80:20 ratio. Also, K- fold cross validation has been used for generating efficient and better results (K. S. Manjunatha and Guru; 2019).

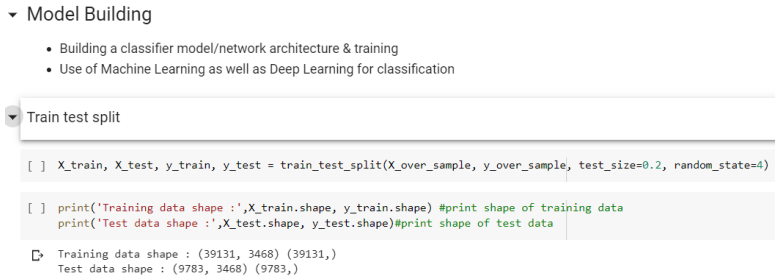

Figure 19: Train- test split

Train and test split for model training has been described by Figure 19.

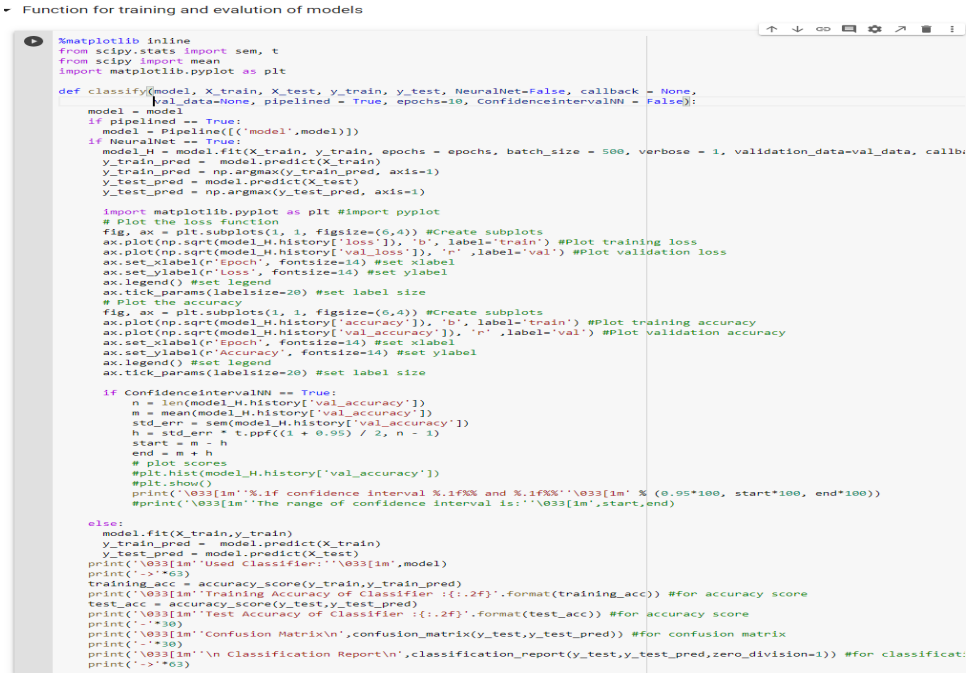

Figure 20: Code for training and evaluation of models

Figure 20 shows the all the function used in this research for training and evaluation of the model.

Importing Libraries

```
from sklearn.pipeline import Pipeline #import pipeline
from sklearn.naive_bayes import MultinomialNB #import multinomial Naive bayes
from sklearn.svm import SVC, LinearSVC #import SVM classes
from sklearn.tree import DecisionTreeClassifier #import Decision Tree
from sklearn.ensemble import RandomForestClassifier #import Random Forest
from sklearn.neighbors import KNeighborsClassifier #import KNN
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer, TfidfTransformer #import Countvectorizer
from sklearn.linear_model import LogisticRegression  #import Logistic Regression
```

Figure 21: importing Libraries for models

Libraries which been imported for running the classification models are represented in Figure 21.

5.2 K-Nearest Neighbours

K-Nearest Neighbour has been applied using default as well as optimized parameters.

KNeighborsClassifier

```
knn=KNeighborsClassifier() #K-neighbors classifier
classify(knn, X_train, X_test, y_train, y_test) #classify using KNN
```

```
Used Classifier: Pipeline(memory=None,
         steps=[('model',
                 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 of Classifier :0.96
Test Accuracy of Classifier :0.93
------------------------------
Confusion Matrix
 [[ 43   0   2 ...   0   0   0]
 [  0 128   0 ...   0   0   0]
 [  0   0  99 ...   0   0   0]
 ...
 [  0   0   0 ... 131   0   0]
 [  0   2   6 ...   0  61   3]
 [  2   0   3 ...   0   6  56]]
------------------------------
```

Figure 22: KNN with default parameters

The code has been shown in Figure 22 for KNN with default parameters. Also, confusion matrix has been described in same.

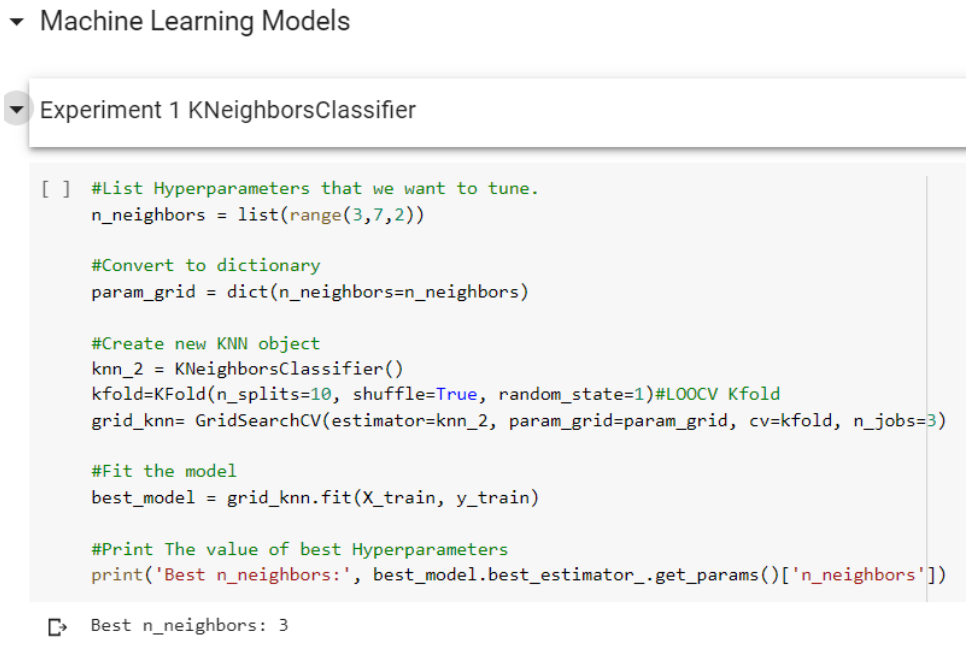

Figure 23: KNN - GridSearchCV and 10 fold cross validation

Refer Figure 23 for KNN with GridSeacrhCV optimization and 10 fold cross validation.

Figure 24: Model training using tuned hyper parameters

5.3 Random Forest Classifier

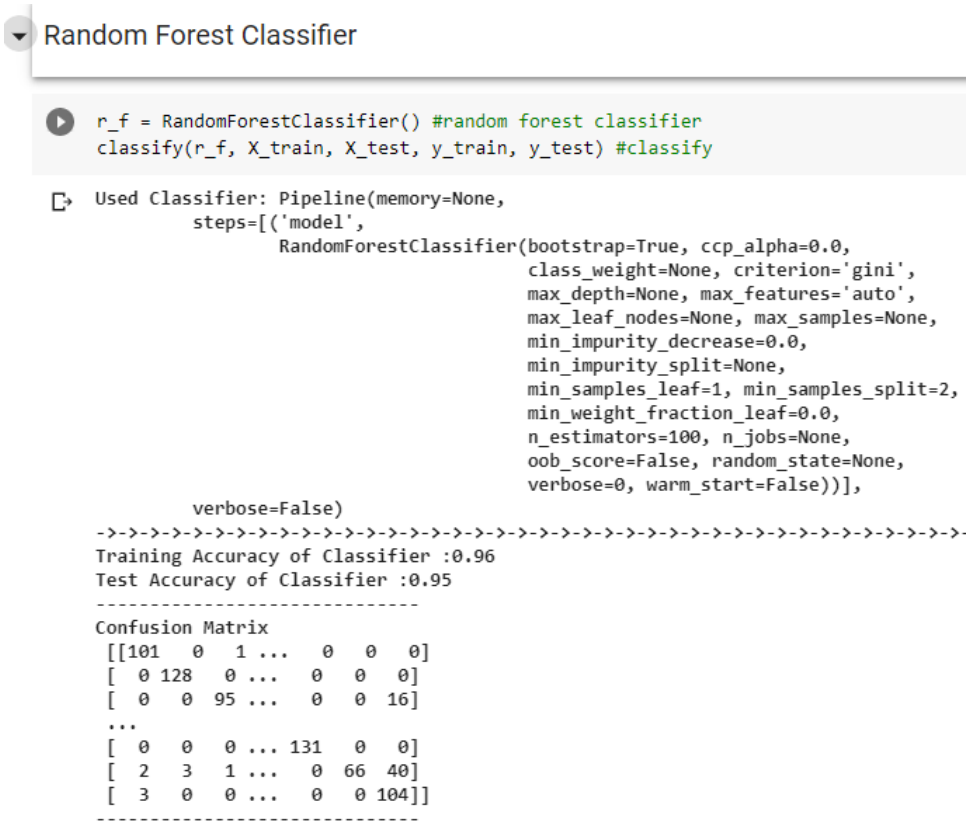

Figure 25: Random Forest with default parameters

The code has been describe in Figure 25 for Random Forest with default parameters and Figure 26 for Random Forest using GridSearchCV with 10 fold cross validation.

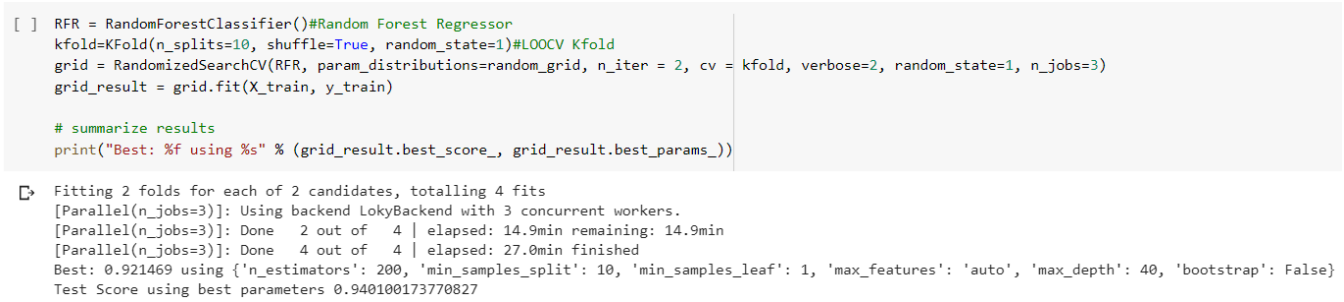

Figure 26: Random Forest- RandomizedSearchCV and 10 fold cross validation

```
r_f = RandomForestClassifier(n_estimators= 200, max_features= 'auto', max_depth=40,
                             min_samples_split=10, min_samples_leaf=1, bootstrap=False)
classify(r_f, X_train, X_test, y_train, y_test) #classify

Used Classifier: Pipeline(memory=None,
         steps=[('model',
                 RandomForestClassifier(bootstrap=False, ccp_alpha=0.0,
                                        class_weight=None, criterion='gini',
                                        max_depth=40, 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=10,
                                        min_weight_fraction_leaf=0.0,
                                        n_estimators=200, n_jobs=None,
                                        oob_score=False, random_state=None,
                                        verbose=0, warm_start=False))],
         verbose=False)
->->->->->->->->->->->->->->->->->->->->->->->->->->->->->->->->->->->->->->->->->->->
Training Accuracy of Classifier :0.95
Test Accuracy of Classifier :0.94
------------------------------
Confusion Matrix
 [[ 71   0   0 ...   0   0   0]
 [  0 128   0 ...   0   0   0]
 [  0   0  92 ...   0   0  19]
 ...
 [  0   0   0 ... 131   0   0]
 [  4   3   1 ...   0  65  39]
 [  0   0   0 ...   0   0  99]]
------------------------------
```

Figure 27: Model training using tuned hyper parameters

Figure 27 shows code used for model training using the tuned hyper parameters.

5.4 Support Vector Machine

SVM has been applied with default refer Figure 28, GridSearchCV and 10 fold cross validation refer Figure 29 and finally model training with obtained hyper parameter Figure 30.

```
SVM Classifier

sv = SVC() #SVM with RBF kernel
classify(sv, X_train, X_test, y_train, y_test) #classify

Used Classifier: Pipeline(memory=None,
         steps=[('model',
                 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 of Classifier :0.92
Test Accuracy of Classifier :0.91
------------------------------
Confusion Matrix
 [[ 96   0   0 ...   0   0   0]
 [  0 116   0 ...   0   0   0]
 [  4   0  90 ...   0   0  19]
 ...
 [  0   0   0 ... 131   0   0]
 [  5   3   1 ...   0  44  59]
 [  3   0   0 ...   0   0 129]]
------------------------------
```

Figure 28: Support Vector Machine with default parameters

Experiment 3 Support Vector Classifier

```
lsv = SVC()
param_grid = {'C': [1, 100, 1000]}
kfold=KFold(n_splits=10, shuffle=True, random_state=1)
grid = GridSearchCV(estimator=lsv, param_grid=param_grid, cv=kfold, n_jobs=3)
grid_result = grid.fit(X_train, y_train)
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
```

```
Best: 0.938105 using {'C': 1000}
```

Figure 29: SVM - GridSearchCV and 10 fold cross validation

```
sv = SVC(C=1000)
classify(sv, X_train, X_test, y_train, y_test)
```

```
Used Classifier: Pipeline(memory=None,
         steps=[('model',
                 SVC(C=1000, 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 of Classifier :0.96
Test Accuracy of Classifier :0.95
------------------------------
Confusion Matrix
 [[ 64   0   0 ...   0   0   0]
 [  0 128   0 ...   0   0   0]
 [  0   0  93 ...   0   0  19]
 ...
 [  0   0   0 ... 131   0   0]
 [  4   3   1 ...   0  54  41]
 [  2   0   0 ...   0   0 105]]
------------------------------
```

Figure 30: Model training using tuned hyper parameter

Importing libraries for Neural networks

NEURAL NETWORK MODELS

Importing libraries

```
import keras.backend as K #import keras
from keras.models import Sequential, Model #import models
from keras.layers import Input, Dropout, Flatten, Dense, Embedding, LSTM, GRU, concatenate, Bidirectional #import layers
from keras.layers import BatchNormalization, TimeDistributed, Conv1D, MaxPooling1D, GlobalMaxPool1D, GlobalMaxPooling1D #import layers
from keras.constraints import max_norm, unit_norm #import layers
from keras.preprocessing.sequence import pad_sequences #import pad sequences
from keras.preprocessing.text import Tokenizer, text_to_word_sequence #import tokenizer
from keras.layers.merge import Concatenate #import concatenate
from keras.callbacks import EarlyStopping, ModelCheckpoint #import callbacks
from keras import optimizers, regularizers #import optimizers
from sklearn.model_selection import train_test_split #import train test split
from sklearn.metrics import confusion_matrix, classification_report, auc #evalution metrics
from sklearn.metrics import roc_curve, accuracy_score, precision_recall_curve #evalution metrics
```

Figure 31: Important libraries for neural network

Refer figure 31 for importing all libraries

5.5 Artificial Neural Network(ANN)

Deep Neural Network

```
def build_DNN(input_shape, neurons, dropout): # Basic neural network model
  """
  Basic deep neural network model using keras.
  Parameters:
    input_shape : Shape of the input data
    neurons : Number of neurons for the input Dense layer
    dropout : dropout for the neural network
    layers : Number of the layers for the neural network
  Returns:
    model : deep neural network model
  """
  model = Sequential()
  model.add(Dense(neurons, input_dim=input_shape, activation='relu'))
  model.add(Dropout(dropout))
  model.add(Dense(neurons, activation='relu'))
  model.add(Dropout(dropout))
  model.add(BatchNormalization())
  model.add(Dense(neurons, activation='relu'))
  model.add(Dropout(dropout))
  model.add(BatchNormalization())
  model.add(Dense(neurons, activation='relu'))
  model.add(Dropout(dropout))
  model.add(BatchNormalization())
  model.add(Dense(neurons, activation='relu'))
  model.add(Dropout(dropout))
  model.add(BatchNormalization())
  model.add(Dense(128, activation='relu'))
  model.add(Dropout(dropout))
  model.add(BatchNormalization())
  model.add(Dense(74, activation = 'softmax'))
  model.compile(loss='sparse_categorical_crossentropy', metrics = ['accuracy'],
                optimizer = 'adam')
  print(model.summary())
  return model
```

Figure 32: ANN with five hidden dense layer

Refer code from Figure 32 for implementing ANN with fixed 5 hidden layers and fixed number of neurons.

Experiment 4 Neural Network

```
[ ]  def build_DNN(input_shape, neurons, dropout, layers): # Basic neural network model
       """
       Basic deep neural network model using keras.
       Parameters:
         input_shape : Shape of the input data
         neurons : Number of neurons for the input Dense layer
         dropout : dropout for the neural network
         layers : Number of the layers for the neural network
       Returns:
         model : deep neural network model

       """
       model = Sequential()
       model.add(Dense(neurons, input_dim=input_shape, activation='relu'))
       model.add(Dropout(dropout))
       for i in range(0,layers):
         model.add(Dense(neurons, activation='relu'))
         model.add(Dropout(dropout))
         model.add(BatchNormalization())
         neurons = int(neurons/2)
       model.add(Dense(74, activation = 'softmax'))
       model.compile(loss='sparse_categorical_crossentropy', metrics = ['accuracy'],
                     optimizer = 'adam')
       print(model.summary())
       return model
```

Figure 33: ANN with four hidden dense layer

Refer code from Figure 33 for implementing ANN with 4 hidden layers or dynamic number of hidden layers with descending number of neurons per layer, keras library has been used for implementation of ANN, 'Adam' as an optimizer and 'Softmax' activation function (Molino et al.; 2018) .

5.6 Bidirectional Long Short Term Memory (BLSTM)

```
def get_model():
    model = Sequential()
    model.add(Embedding(num_words + 1, embedding_size, input_length=X_train_token.shape[1], weights=[embedding_matrix], trainable=True))
    model.add(Bidirectional(LSTM(128, return_sequences=True)))
    #model.add(LSTM(128, recurrent_dropout=0.2, return_sequences=True))
    model.add(Dense(128, activation='relu'))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(16, activation='relu'))
    model.add(Flatten())
    model.add(Dense(class_count, activation='softmax'))
    model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    print(model.summary())
    return model
```

Figure 34: BLSTM using Five hidden layers

```
Experiment 5 LSTM

[ ] def get_model():
        model = Sequential()
        #model.add(Input(shape=(maxlen,)))
        model.add(Embedding(num_words + 1, embedding_size, input_length=X_train_token.shape[1], weights=[embedding_matrix], trainable=True))
        model.add(Bidirectional(LSTM(128, return_sequences=True)))
        #model.add(LSTM(128, recurrent_dropout=0.2, return_sequences=True))
        model.add(Dense(512, activation='relu'))
        model.add(Dense(256, activation='relu'))
        model.add(Dense(128, activation='relu'))
        model.add(Flatten())
        model.add(Dense(class_count, activation='softmax'))
        model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
        print(model.summary())
        return model
```

Figure 35: BLSTM using four hidden layers

Refer Figure 34 and Figure 35 for code of BLSTM [4].

Note: All the required notebook settings have already been configured in 'Project_x18196292.ipynb' file. If execution is required then import the notebbook file in google colab and execute.
Also, all intermediate supporting files have also been included in code artefacts.

References

K. S. Manjunatha, H. A. and Guru, D. S. (2019). *Data Analytics and Learning*, Vol. 43, Springer Singapore.
URL: *http://link.springer.com/10.1007/978-981-13-2514-4*

[4] `https://medium.com/@jonathan_hui/nlp-word-embedding-glove-5e7f523999f6`

Molino, P., Zheng, H. and Wang, Y. C. (2018). COTA: Improving the speed and accuracy of customer support through ranking and deep networks, *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* pp. 586–595.