greatlearning

AUTOMATED TICKET ASSIGNMENT

AIML JUNE GROUP 5A-NLP
MENTOR:
Mr. AMIT JAIN
SUBMITTED BY:
NAMAGIRI LAKSHMI TB
ANAND KAKARADDI
ADITYA KUMAR

CONTENTS

1.SUMMARY OF PROBLEM STATEMENT,DATA AND FINDINGS

One of the key activities of any IT function is to “Keep the lights on” to ensure there is no impact to the Business operations. IT leverages Incident Management process to achieve the above Objective. An incident is something that is unplanned interruption to an IT service or reduction in the quality of an IT service that affects the Users and the Business. The main goal of Incident Management process is to provide a quick fix / workarounds or solutions that resolves the interruption and restores the service to its full capacity to ensure no business impact. In most of the organizations, incidents are created by various Business and IT Users, End Users/ Vendors if they have access to ticketing systems, and from the integrated monitoring systems and tools. Assigning the incidents to the appropriate person or unit in the support team has critical importance to provide improved user satisfaction while ensuring better allocation of support resources. The assignment of incidents to appropriate IT groups is still a manual process in many of the IT organizations. Manual assignment of incidents is time consuming and requires 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

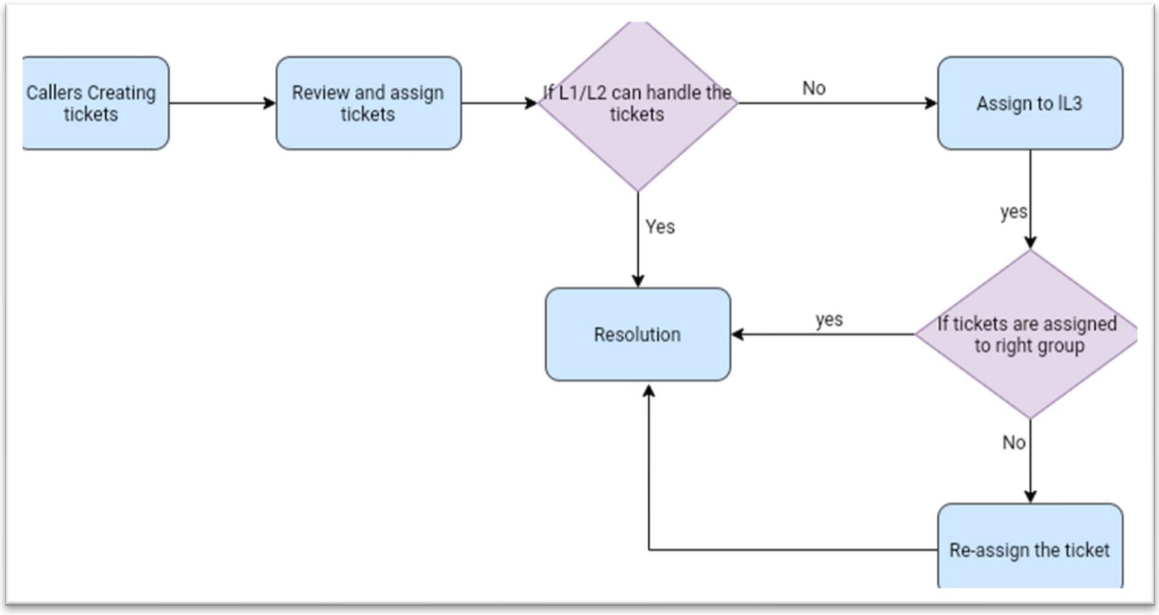
In the support process, incoming incidents are analysed and assessed by organization’s support teams to fulfil the request. In many organizations, better allocation and effective usage of the valuable support resources will directly result in substantial cost savings. Currently the incidents are created by various stakeholders (Business Users, IT Users and Monitoring Tools) within IT Service Management Tool and are assigned to Service Desk teams (L1 / L2 teams). This team will review the incidents for right ticket categorization, priorities and then carry out initial diagnosis to see if they can resolve. Around ~54% of the incidents are resolved by L1 / L2 teams. In case 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. In case 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. 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. Project Description In this capstone project, the goal is to build a classifier that can classify the tickets by analysing text.

PROBLEM INTERPRETATION

Classification of tickets to right functional groups which helps to reduce the resolving time of the issue.

Here, assuming the L1/L2 are not able to resolve the issue and hence it has to be passed to functional group, sometimes instead of assigning an incident ticket to group 1, it might be assigned to group 2 which creates a lot of chaos and results in waste of time. Hence, to avoid this confusion we are using automated ticket classification.



ADVANTAGES OF USING AUTOMATED TICKET CLASSIFIER.

- Reducing the time taken to assign the tickets to particular group without having much human intervention.
- Avoids confusion by automatically assigning tickets to functional groups.
- Increases user-satisfaction and improves customer service.
- Better allocation and effective usage of resources.

STEPS IN INCIDENT MANAGEMENT

- Collecting relevant details at the time of ticket creation.
- Sorting the tickets and prioritizing, ensuring that the ticket is sent to suitable group to work on the same and provide the customer with a solution.

PROBLEMS FACED WHEN A TICKET IS SENT TO AN INAPPROPRIATE GROUP

- No detailed record of past incidents and decreased customer satisfaction.
- If an incident is marked important and doesn't get resolved customer gets frustrated.

MILESTONES

Milestone 1:

Pre-Processing, Data Visualisation and EDA

Overview

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 pre-processing
i) Creating word vocabulary from the corpus of report text data
j) Creating tokens as required.

Milestone 2:

Model Building

Overview

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.

Milestone 3:

Test the Model, Fine-tuning and Repeat

Overview

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.

INFORMATION ABOUT THE DATASET

Goal is to build a classifier that can classify the tickets by analysing the text.

ATTRIBUTES OF THE DATASET:

Name of the dataset: **input_data.xlsx**

- **Assignment Group** - Various Functional Groups handling different category of incidents.

- **Caller -** Callers/Users raising the incident tickets.
- **Short Description -** a short description for the raised incident ticket provided the user.
- **Description -** a detailed description for the raised incident ticket provided the user.

Based on the Description, Assignment group has to be selected automatically.
The dataset has 8500 rows of data, 4 columns and **74** assignment groups.

	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

DATA FINDINGS.

a) Highly imbalanced dataset in terms of ticket distribution across **Assignment group.**
b) **8** null values present in Short Description column and 1 null value present in Description column.

```
[ ]  #checking for null values in our dataset
     df.isna().sum()
```

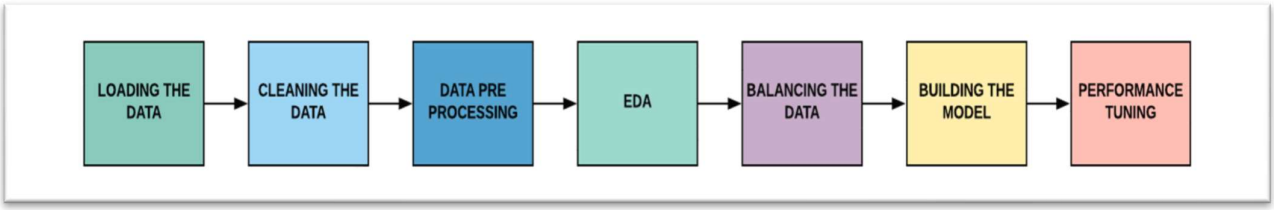
```
Short description    8
Description          1
Caller               0
Assignment group     0
dtype: int64
```

Group 0 occupies around 46% of total distribution of tickets. Hence the model will predict very poorly for other groups and prediction will be excellent for **Group 0.**

c) **Hence, merging the groups with less than 20 tickets under the name "Manual".**
d) After merging we have a total of **42 groups.**

2. OVERVIEW OF THE FINAL PROCESS

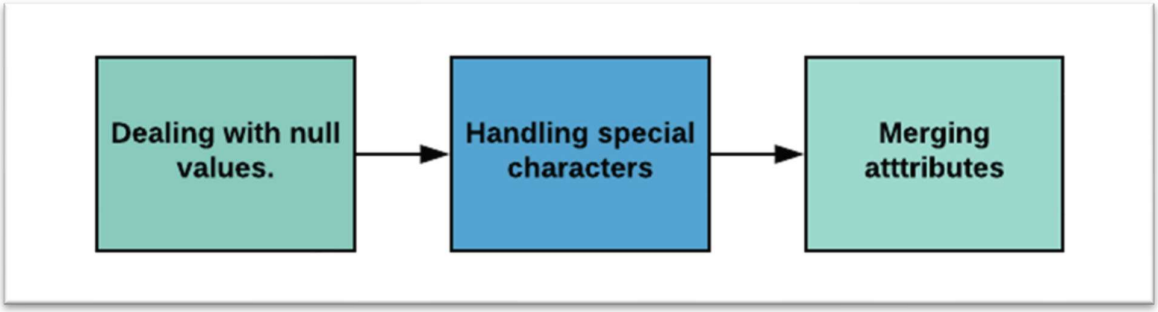
DESIGN



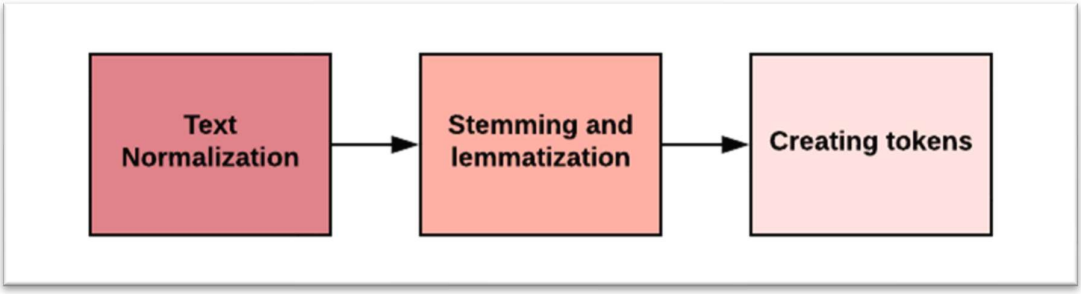
- Loading the data
- Cleaning the data
- Data pre-processing

- Eda
- Balancing the data
- Building the model
- Performance tuning

DATA CLEAN UP SUB-FLOW:



DATA PRE-PROCESSING SUB-FLOW:



1. Text Normalization
   - Converting to lower case
   - Removal of hyperlinks
   - Removal of URL
   - Removal of e-mail
   - Removal of special characters
   - Expanding the text
   - Removal of single char and extra spaces
   - Removal of stop words

2. Performing stemming and Lemmatization.
3. Creating tokens.

DATA BALANCING SUB-FLOW

- Down sampling the majority topics under GRP_0.
- Club groups with lesser than 20 tickets assigned.
- Join and prepare balanced dataset.

Traditional Machine Learning Models:

- Logistic Regression
- Naïve Bayes
- K Nearest Neighbours
- Support Vector Machine
- Random Forest

Deep Learning Models:

- Simple Neural network
- LSTM
- Bi-LSTM
- GRU

We are planning to achieve an accuracy of above 90% as the domain we are dealing with is not critical. in case of critical domain (medical, related to life) an accuracy of 95% and above is good.

3. STEP BY STEP WALK THROUGH OF THE SOLUTION

FILES USED IN PROJECT

FILE 1-NLP_EDA

- Contains all the EDA performed.

FILE2-NLP_ML_MODELS

- Contains traditional ML models built on raw data, up sampled and down sampled data.

FILE3- NLP_ML_MODELS_UPSAMPLED

- Contains traditional ML models up sampled using Random Over Sampler technique

FILE4-NLP_NN_Raw_data

- Has deep learning models built on raw data(pre-processed)

FILE5-NLP_NN_Upsampled

- Deep learning models on up sampled data.

FILE6-NLP_NN_DOWNSAMPLED

- Deep learning models on down sampled data

3.1. LOADING AND EXPLORING THE DATASET

- Mount the drive and set the project path to current working directory, when using Google Colabs. This step is not needed if you are running on your local machine.
- Import the necessary libraries.

```
[ ] #setting project path
    import os
    os.chdir('/content/drive/My Drive')

[ ] #to check if the current working directory is same as mentioned above
    path = os.getcwd()
    print(path)

    /content/drive/My Drive

[ ] import numpy as np
    import pandas as pd
    df = pd.read_excel("input_data.xlsx",encoding='utf-8')

    #checking few records from dataset to understand the structure of data
    df.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

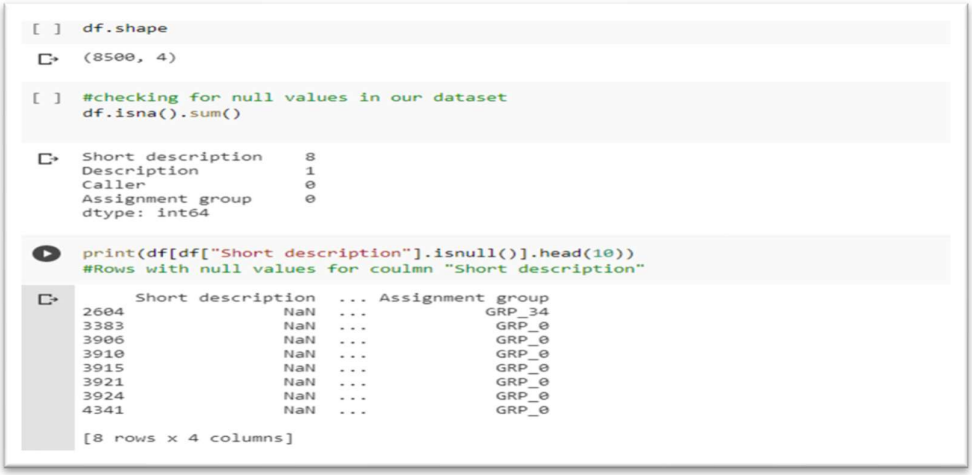
```
[ ] df.shape

    (8500, 4)

[ ] #checking for null values in our dataset
    df.isna().sum()

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

3.2. CLEANING THE DATA

3.2.1 DEALING WITH NULL VALUES

I. There are many ways to deal with missing/null values.
II. Replacing them with a stop word.
III. Dropping the records which contain them.
IV. Replacing them with an empty string.

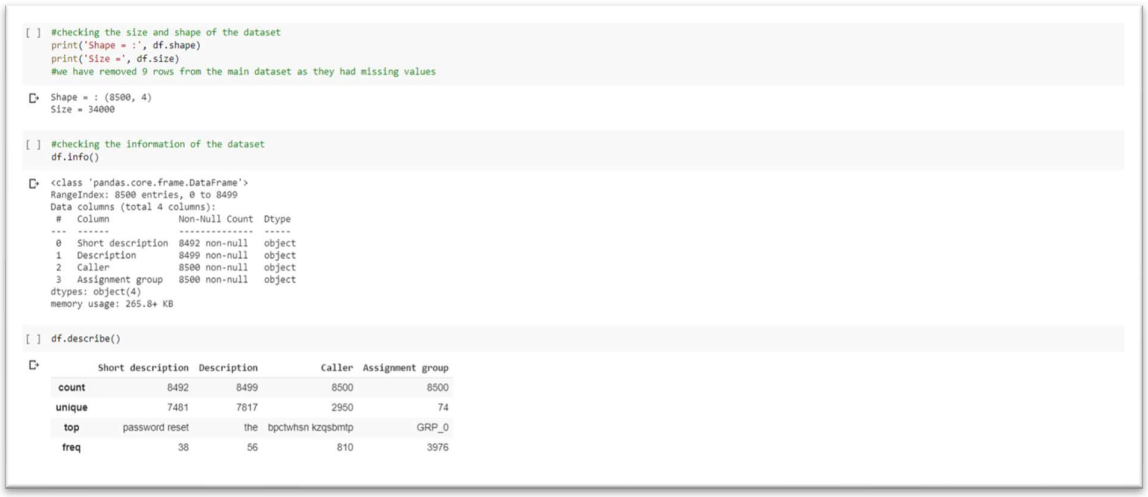
- We did not want to drop the records as it results in loss of data.
- We notice we have 8500 rows and 4 columns in which we have a total of 9 null values. Since the null values are not present in common in both the rows of Description and Short Description, we will not be replacing them with any values. However, the columns Description and Short Description will be merged so the null values will be taken care of while merging and we will be left with no null values. Replacing them with stop words was avoided as we will be removing the stop words.



```
[ ]  df.shape

     (8500, 4)

[ ]  #checking for null values in our dataset
     df.isna().sum()

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

     print(df[df["Short description"].isnull()].head(10))
     #Rows with null values for coulmn "Short description"

          Short description  ... Assignment group
     2604               NaN  ...          GRP_34
     3383               NaN  ...           GRP_0
     3906               NaN  ...           GRP_0
     3910               NaN  ...           GRP_0
     3915               NaN  ...           GRP_0
     3921               NaN  ...           GRP_0
     3924               NaN  ...           GRP_0
     4341               NaN  ...           GRP_0

     [8 rows x 4 columns]
```

3.2.2. DESCRIBING THE DATASET WITH VARIOUS SUMMARY AND STATISTICS.

We notice we have 8500 rows and 4 columns in which we have a total of 9 null values.



```
[ ]  #checking the size and shape of the dataset
     print('Shape = :', df.shape)
     print('Size =', df.size)
     #we have removed 9 rows from the main dataset as they had missing values

     Shape = : (8500, 4)
     Size = 34000

[ ]  #checking the information of the dataset
     df.info()

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

[ ]  df.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

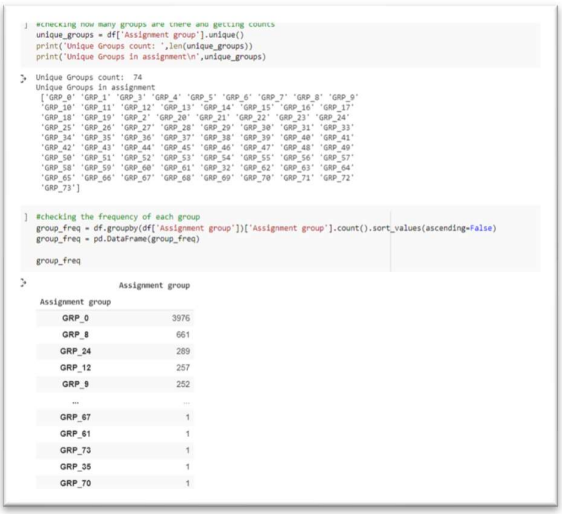
OBSERVATIONS:

- Shape of the data is 8500,4.
- Size of the data is 34000.
- Total Description count is 8500 out of which we have 7817 unique values.
- Total Short Description count is 8492 out of which 7841 are unique.
- The datatype of all the columns are of “object” type.
- The most used word in Short Description is ‘password reset’ and ‘the’ in Description.
- Top caller **is bpctwhsn kzqsbmtp** and top Assignment group is **GRP_0.**

3.3. ANALYSIS ON **CALLER** COLUMN.

- We notice a total of 74 Assignment group raging from GRP_0 to GRP_72.
- Top 5 groups are GRP_0, GRP_8, GRP_24, GRP_12, GRP_9.

- Most of the tickets are assigned to GRP_0, our model will predict the best on the group because it is the majority group which says our group distribution is highly imbalanced.

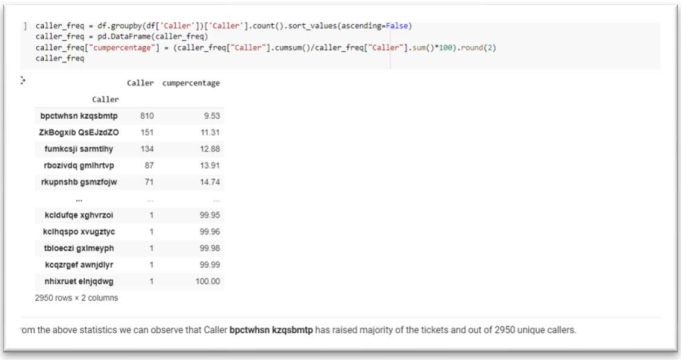


```
#checking how many groups are there and getting counts
unique_groups = df['Assignment group'].unique()
print('Unique Groups count: ',len(unique_groups))
print('Unique Groups in assignment\n',unique_groups)
```

```
Unique Groups count:  74
Unique Groups in assignment
 ['GRP_0' 'GRP_1' 'GRP_3' 'GRP_4' 'GRP_5' 'GRP_6' 'GRP_7' 'GRP_8' 'GRP_9'
 'GRP_10' 'GRP_11' 'GRP_12' 'GRP_13' 'GRP_14' 'GRP_15' 'GRP_16' 'GRP_17'
 'GRP_18' 'GRP_19' 'GRP_2' 'GRP_20' 'GRP_21' 'GRP_22' 'GRP_23' 'GRP_24'
 'GRP_25' 'GRP_26' 'GRP_27' 'GRP_28' 'GRP_29' 'GRP_30' 'GRP_31' 'GRP_33'
 'GRP_34' 'GRP_35' 'GRP_36' 'GRP_37' 'GRP_38' 'GRP_39' 'GRP_40' 'GRP_41'
 'GRP_42' 'GRP_43' 'GRP_44' 'GRP_45' 'GRP_46' 'GRP_47' 'GRP_48' 'GRP_49'
 'GRP_50' 'GRP_51' 'GRP_52' 'GRP_53' 'GRP_54' 'GRP_55' 'GRP_56' 'GRP_57'
 'GRP_58' 'GRP_59' 'GRP_60' 'GRP_61' 'GRP_32' 'GRP_62' 'GRP_63' 'GRP_64'
 'GRP_65' 'GRP_66' 'GRP_67' 'GRP_68' 'GRP_69' 'GRP_70' 'GRP_71' 'GRP_72'
 'GRP_73']
```

```
#checking the frequency of each group
group_freq = df.groupby(df['Assignment group'])['Assignment group'].count().sort_values(ascending=False)
group_freq = pd.DataFrame(group_freq)

group_freq
```

Assignment group	Assignment group
GRP_0	3976
GRP_8	661
GRP_24	289
GRP_12	257
GRP_9	252
...	...
GRP_67	1
GRP_61	1
GRP_73	1
GRP_35	1
GRP_70	1



```
caller_freq = df.groupby(df['Caller'])['Caller'].count().sort_values(ascending=False)
caller_freq = pd.DataFrame(caller_freq)
caller_freq["cumpercentage"] = (caller_freq["Caller"].cumsum()/caller_freq["Caller"].sum()*100).round(2)
caller_freq
```

Caller	Caller	cumpercentage
bpctwhsn kzqsbmtp	810	9.53
ZkBogxib QsEJzdZO	151	11.31
fumkcsji sarmtlhy	134	12.88
rbozivdq gmlhrtvp	87	13.91
rkupnshb gsmzfojw	71	14.74
...	...	...
kcldufqe xghvrzoi	1	99.95
kcihqspo xvugztyc	1	99.96
tbloeczi gxlmeyph	1	99.98
kcqzrgef awnjdlyr	1	99.99
nhixruet elnjqdwg	1	100.00

2950 rows × 2 columns

om the above statistics we can observe that Caller **bpctwhsn kzqsbmtp** has raised majority of the tickets and out of 2950 unique callers.

Analysis for Top 10 Callers

Top10 Callers

```
#checking the frequency of each group
Caller_freq = df.groupby(df['Caller'])['Caller'].count().sort_values(ascending=False)
Caller_freq = pd.DataFrame(Caller_freq)

Caller_freq
Top_10_Callers=Caller_freq.head(10)
```

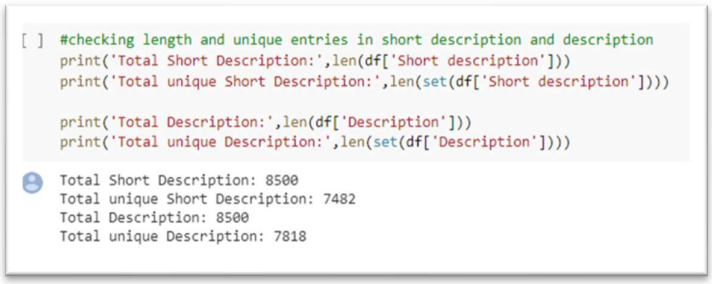
```
Top_10_Callers["cumpercentage"] = Top_10_Callers["Caller"].cumsum()/Top_10_Callers["Caller"].sum()*100
#visualizing the tickets assigned to each group
fig, ax = plt.subplots(figsize=(15,5))
plt.xticks(rotation='vertical')
plt.tick_params(labelsize=13)
plt.title('Top_10_callers')
ax.bar(Top_10_Callers.index, Top_10_Callers["Caller"], color="C0")
print(Top_10_Callers)


plt.show()
```

```
                Caller  cumpercentage
Caller
bpctwhsn kzqsbmtp     810      52.529183
ZkBogxib QsEJzdZO     151      62.321660
fumkcsji sarmtlhy     134      71.011673
rbozivdq gmlhrtvp      87      76.653696
rkupnshb gsmzfojw      71      81.258106
jloygrwh acvztedi      64      85.408500
spxqmiry zpwgoqju      63      89.494163
oldrctiu bxurpsyi      57      93.190661
olckhmvx pcqobjnd      54      96.692607
```

- We notice that around 810 tickets have been raised by the caller *bpctwhsn kzqsbmtp* which makes him to the caller.

The groups to which *bpctwhsn kzqsbmtp* has raised tickets.

3.4. ANALYSIS ON 'SHORT DESCRIPTION' AND 'DESCRIPTION' COLUMN.



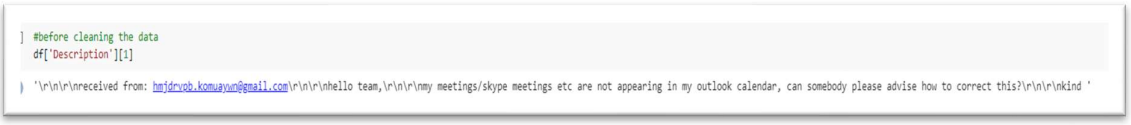
We have visualized the most commonly used words (top 20) along with the unique count of words.

FINAL OBSERVATIONS:

1. From the above analysis we can conclude that our dataset contained 8500 rows and 4 calls which contained null values.
2. We notice null values but since we are going to merge columns, we are not removing it.
3. We have a total of 74 unique groups and 2948 unique callers.
4. We see a major contribution from GROUP_0 of assignment group which handles almost 50% of the input tickets which results in low accuracy for those groups having less percentage.
5. We notice caller **bpctwhsn kzqsbmtp** contributes to 10% of the total tickets which tells that a single caller has many issues.

3.5. DATA PRE-PROCESSING

- Before performing the pre-processing:



We notice many extra characters and spaces present, so we shall remove them using the below code.

Stemming

Stemming is the process of producing morphological variants of a root/base word. Stemming programs are commonly referred to as stemming algorithms or stemmers. A stemming algorithm reduces the words “chocolates”, “chocolatey”, “choco” to the root word, “chocolate” and “retrieval”, “retrieved”, “retrieves” reduce to the stem “retrieve”.

Lemmatization

Lemmatization is the process of grouping together the different inflected forms of a word so they can be analysed as a single item. Lemmatization is similar to stemming but it brings context to the words. So, it links words with similar meaning to one word.

Code snippet:

```
lemmatizer = WordNetLemmatizer()
stemmer = PorterStemmer()

def preprocess(text):
    sentence     = str(text)

#Converting to lower case
    lower = sentence.lower()
    clean        = re.compile('<.*?>')
#Remove hyperlink
    hyperlink    = re.sub(clean, '',lower)
 #Remove Web link
    url      = re.sub(r'http:\S+', ' ',hyperlink)
 #removing maillid
    mail     = re.sub(r"\S*@\S*.com\s?",' ',url)
 #expanding text
    expand   = contractions.fix(mail)
 #special characters
    specialchar  = re.sub(r"[!';:@#$%^&*()~?<>,\.:+-=`""'\'_]",' ',expand)
 #numbers
    num      = re.sub(r'\d', '', specialchar)
    spcl     = re.sub("([^\x00-\x7F])+"," ",num)
 #singlechar
    single   = re.sub(r"\s+[a-zA-Z]\s+", ' ',spcl)
 #extraspaces
    spaces       = re.sub(r"\s+", ' ',single)
    tokenizer    = RegexpTokenizer(r'\w+')
    text_token   = tokenizer.tokenize(spaces)  #Text tokenized
#Removal of stop words
    filter_words = [w for w in text_token if len(w) > 2 if not w in stopwords.words('english')]
#Word Stemming
    words_stem   = [stemmer.stem(w) for w in filter_words]
#Word Lemmatization
    clean_text   = [lemmatizer.lemmatize(w) for w in words_stem]
    return " ".join(clean_text)
```

Steps followed:

- Converting to lower case
- Removal of hyperlinks
- Removal of URL
- Removal of e-mail
- Removal of special characters
- Expanding the text
- Removal of single char and extra spaces

- Removal of stop words
- Performing stemming and Lemmatization

- After pre-processing:

```
[ ] #after cleaning the data
    df['Description'][1]

    'receiv hello team meet skype meet etc appear outlook calendar somebodi plea advis correct kind'
```

OBSERVATIONS:

- Entire dataset is converted to lower case.
- User e-mail id did not add any value to our analysis, despite the fact that the user id is also given in the caller column. Hence, all the e-mail id is removed from the dataset.
- All numerals removed because they were dominating.
- All punctuation marks removed as they were a hinderance lemmatization.
- All occurrences of more than one blank space, new line breaks, blank spaces, special char etc. have been replaced with a single blank space.

3.5.1. MERGING THE DESCRIPTION, CALLER AND SHORT DESCRIPTION COLUMN

- Merging the columns mentioned above as we notice certain rows have same description and short description and the caller names are present in both in caller column and description column.
- Merging of description and short description under the name 'Description'

```
df['isSubstr'] =df.apply(lambda x: x['Short description'] not in x['Description'], axis=1)

#we notice redundancy in short description hence we will append with description wherever the redundancy is present.
df_merge=df.copy()
df_merge.loc[df_merge['isSubstr'] == True, 'Description'] =df_merge['Short description'] + " "+ df_merge['Description']
#Drop Short description and issubstr
df_merge =df_merge.drop(['Short description'],axis=1)
df_merge =df_merge.drop(['isSubstr'],axis=1)
df_merge.tail()
```

	Description	Caller	Assignment group
8495	email come mail receiv good afternoon receiv e...	avglmrts vhqmtiua	GRP_29
8496	telephoni softwar issu	rbozivdq gmlhrtvp	GRP_0
8497	vip window password reset tifpdchb pedxruyf	oybwdsgx oxyhwrfz	GRP_0
8498	machin est funcionando unabl access machin uti...	ufawcgob aowtxjky	GRP_62
8499	mehreren lassen sich verschieden prgramdntym n...	kqvbrspl jyzoklfx	GRP_49

```
df.head()
```

	Short description	Description	Caller	Assignment group	isSubstr
0	login issu	verifi user detail employe manag name check us...	spxjnwir pjlcoqds	GRP_0	False
1	outlook	receiv hello team meet skype meet etc appear o...	hmjdrvpb komuaywn	GRP_0	False
2	log vpn	receiv cannot log vpn best	eylqgodm ybqkwiam	GRP_0	False
3	unabl access tool page	unabl access tool page	xbkucsvz gcpydteq	GRP_0	False
4	skype error	skype error	owlgqjme qhcozdfx	GRP_0	False

- Merging of caller into Description

```
#append caller into description if not already present

df_merge['isCallerNotSubstr'] =df_merge.apply(lambda x: x['Caller'] not in x['Description'], axis=1)

#We will append caller to description if True only then drop short description
df_merge_caller = df_merge.copy()
df_merge_caller.loc[df_merge_caller['isCallerNotSubstr'] == True, 'Description'] =df_merge_caller['Description'] + " " +df_merge_caller['Caller']
#Drop Caller
df_merge_caller = df_merge_caller.drop(['Caller'],axis=1)
df_merge_caller = df_merge_caller.drop(['isCallerNotSubstr'],axis=1)
df_merge_caller.tail(10)
```

- Dataset after merging:

	Description	Assignment group
8490	check statu purchas plea contact pasgryowski p...	GRP_29
8491	vpn laptop receiv need vpn new laptop name llv...	GRP_34
8492	tool etim option visitbl tmopbken ibzougsd	GRP_0
8493	erp two account ad sorri anoth two account nee...	GRP_10
8494	tablet need reimag due multipl issu crm wifi e...	GRP_3
8495	email come mail receiv good afternoon receiv e...	GRP_29
8496	telephoni softwar issu rbozivdq gmlhrtvp	GRP_0
8497	vip window password reset tifpdchb pedxruyf oy...	GRP_0
8498	machin est funcionando unabl access machin uti...	GRP_62
8499	mehreren lassen sich verschieden prgramdntym n...	GRP_49

- After merging we have a total of two columns which is **Description** and **Assignment group.**
- **Assignment group** will be our target.

```
#Let us make a copy of description as nndes for neural network and mldes for traditonal ml models
df_merge_caller['nndes'] = df_merge_caller['Description'].copy()
df_merge_caller['mldes'] = df_merge_caller['Description'].copy()
df_merge_caller.head()
```

	Description	Assignment group	nndes	mldes
0	verifi user detail employe manag name check us...	GRP_0	verifi user detail employe manag name check us...	verifi user detail employe manag name check us...
1	receiv hello team meet skype meet etc appear o...	GRP_0	receiv hello team meet skype meet etc appear o...	receiv hello team meet skype meet etc appear o...
2	receiv cannot log vpn best eylqgodm ybqkwiam	GRP_0	receiv cannot log vpn best eylqgodm ybqkwiam	receiv cannot log vpn best eylqgodm ybqkwiam
3	unabl access tool page xbkucsvz gcpydteq	GRP_0	unabl access tool page xbkucsvz gcpydteq	unabl access tool page xbkucsvz gcpydteq
4	skype error owlgqjme qhcozdfx	GRP_0	skype error owlgqjme qhcozdfx	skype error owlgqjme qhcozdfx

- Creating two new columns called **mldes** and **nndes** for model building where nndes will be used to neural network and mldes used for machine learning models.

3.5.2. CRAETING A NEW GROUP CALLED **"MANUAL".**

- We noticed that there are groups which have very less number of tickets, those groups with less number of tickets can be put under a separate group called **Manual.**
- **Manual** group contains the groups which have less than 20 tickets**.**

```
#for 95% significant data, we can merge the groups that have lesser representation
group_elimination_treshold = 20

filtered_group = dict(filter(lambda x: x[1] >= group_elimination_treshold, group_frequency.items()))
print (filtered_group)
valid_group = list(filtered_group)
print(len(valid_group))
```

```
{'GRP_0': 3976, 'GRP_8': 661, 'GRP_24': 289, 'GRP_12': 257, 'GRP_9': 252, 'GRP_2': 241, 'GRP_19': 215, 'GRP_3': 200, 'GRP_6': 184, 'GRP_13': 145, 'GRP_10': 140, 'GRP_5': 129, 'GRP_14': 118, 'GRP_25': 116, 'GRP_33': 107, 'GRP_4': 100, 'GRP_2
41
```

```
df_merge_caller.describe()
```

	Description	Assignment group	nndes	mldes
count	8500	8500	8500	8500
unique	7411	74	7411	7411
top	receiv job job fail job schedul bpctwhsn kzqsbmtp	GRP_0	receiv job job fail job schedul bpctwhsn kzqsbmtp	receiv job job fail job schedul bpctwhsn kzqsbmtp
freq	460	3976	460	460

Will replace the group with value "Manual" if the group is not in the valid list

```
a=df_merge_caller

for ind in a.index:
    a['Assignment group'][ind]= a['Assignment group'][ind] if a['Assignment group'][ind] in (valid_group) else "Manual"
     #print(a['Assignment group'][ind])

df_filtered_m=a

df_filtered_m
```

Before merging we had a total of **74** groups.
After merging we have a total of **42** groups which will be used for model building.

```
#number of groups should be valid-groupcount  +1 (Manual)
```

	Description	Assignment group	nndes	mldes
count	8500	8500	8500	8500
unique	7411	42	7411	7411
top	receiv job job fail job schedul bpctwhsn kzqsbmtp	GRP_0	receiv job job fail job schedul bpctwhsn kzqsbmtp	receiv job job fail job schedul bpctwhsn kzqsbmtp
freq	460	3976	460	460

After merging we notice we don't have any null/missing values present.

```
#checking for null values
df_filtered_m.isna().sum()
```

```
Description         0
Assignment group    0
nndes               0
mldes               0
dtype: int64
```

3.5.3. VISULAZING THE TOP WORDS AND BI-GRAMS.

- Checking the Top 20 most frequently occurring uni-gram words.

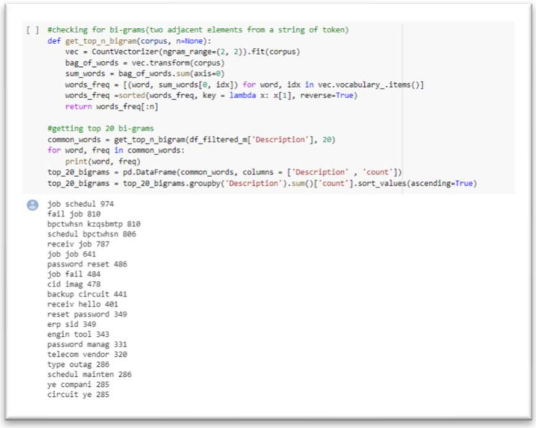
Code snippet:

```
from sklearn.feature_extraction.text import CountVectorizer
#to get the most commonly used words

def get_top_n_words(corpus, n=None):
    vec = CountVectorizer().fit(corpus) #converts it to a matrix of text token
    bag_of_words = vec.transform(corpus) #tokenizes string by extracting words
    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) #adding the second element x[1] to sort func here
    return words_freq[:n]
common_words = get_top_n_words(df_filtered_m['Description'], 20) #20 as we want top20

#checking for word and its frequency
for word, freq in common_words:
    print(word, freq)

#having a look at top 20 words
top_20_words = pd.DataFrame(common_words, columns = ['Description' , 'count'])
top_20_words = top_20_words.groupby('Description').sum()['count'].sort_values(ascending=True)
```

```
job 2611
receiv 2541
plea 2063
ye 2027
password 1912
erp 1624
user 1521
tool 1431
compani 1405
schedul 1301
issu 1298
access 1274
sid 1232
reset 1146
ticket 1099
work 998
fail 970
error 942
unabl 937
```

- We notice job and receive are the top 2 most occurred words.

MOST FREQUENTLY OCCURRED BI-GRAMS (TOP 20).



```
[ ] #checking for bi-grams(two adjacent elements from a string of token)
    def get_top_n_bigram(corpus, n=None):
        vec = CountVectorizer(ngram_range=(2, 2)).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[:n]

    #getting top 20 bi-grams
    common_words = get_top_n_bigram(df_filtered_m['Description'], 20)
    for word, freq in common_words:
        print(word, freq)
    top_20_bigrams = pd.DataFrame(common_words, columns = ['Description' , 'count'])
    top_20_bigrams = top_20_bigrams.groupby('Description').sum()['count'].sort_values(ascending=True)

job schedul 974
fail job 810
bpctwhsn kzqsbmtp 810
schedul bpctwhsn 806
receiv job 787
job job 641
password reset 486
job fail 484
cid imag 478
backup circuit 441
receiv hello 401
reset password 349
erp sid 349
engin tool 343
password manag 331
telecom vendor 320
type outag 286
schedul mainten 286
ye compari 285
circuit ye 285
```

- Storing the above performed steps in a file called **eda.csv** which will be used for model building . storing it in a different file as we need not re-run if the runtime gets disconnected.
- **eda.csv** file contains all the eda performed on the dataset.

4. MODEL EVALUATION.

- The data provided had huge imbalance in the dataset in terms of distribution of tickets among Assignment Groups.
- Merged groups which had less than 20 tickets into a single group called ‘Manual’
- We performed Up sampling on the train data using SMOTE.
- We also tried other up sampling methods like RandomOverSampler.
- We down sampled the data and found good accuracies.

Initial analysis gave us insights that KNN performed well on the raw data(pre-processed) data with a train accuracy of **72%** and test accuracy of **64%**. However, in spite of its advantages it has some limitations.

However, with deep learning models built on down sampled data we noticed an accuracy of 94 on train and 93 on test rendered by LSTM, BI-LSTM AND GRU by using transfer learning.

MODEL ARCHITECTURE.

Build multi-class classifier than can classify the tickets by analysing the text. Text present in Short Description and Description are concatenated prior to model building.

4.1. **MODEL BUILDING**- MILESTONE 1

We have tried the following models and choose the most optimal one in Milestone 1.
Splitting the data into train and test set in the ratio 80:20.

- 80% Training data
- 20% testing data

Code snippet:

```
def Raw_model_performance():

    Result=pd.DataFrame(columns={"Model","Test acc","Train acc"})
    train_acc,test_acc = run_classification(MultinomialNB(), X_train, X_test, y_train, y_test)
    new_row={{"Model":"Naive bayes","Test acc":test_acc,"Train acc":train_acc}}
    Result=Result.append(new_row,ignore_index=True)


    #using kNN
    train_acc,test_acc=run_classification(KNeighborsClassifier(), X_train, X_test, y_train, y_test)
    new_row={{"Model":"kNN","Test acc":test_acc,"Train acc":train_acc}}
    Result=Result.append(new_row,ignore_index=True)


    # SVM ()
    train_acc,test_acc=run_classification(SVC(kernel='rbf'), X_train, X_test, y_train, y_test)
    new_row={{"Model":"SVM()","Test acc":test_acc,"Train acc":train_acc}}
    Result=Result.append(new_row,ignore_index=True)


    #decision tree classifier
    train_acc,test_acc=run_classification(DecisionTreeClassifier(), X_train, X_test, y_train, y_test)
    new_row={{"Model":"decision tree classifier","Test acc":test_acc,"Train acc":train_acc}}
    Result=Result.append(new_row,ignore_index=True)

    #using random forest
    train_acc,test_acc=run_classification(RandomForestClassifier(n_estimators=100), X_train, X_test, y_train, y_test)
    new_row={{"Model":"RandomForestClassifier","Test acc":test_acc,"Train acc":train_acc}}
    Result=Result.append(new_row,ignore_index=True)
    train_acc,test_acc=run_classification(LogisticRegression(), X_train, X_test, y_train, y_test)
    new_row={{"Model":"LogisticRegression","Test acc":test_acc,"Train acc":train_acc}}
    Result=Result.append(new_row,ignore_index=True)


    return Result
```

Using a function named Raw_model_performance () to build various models.

- Multinomial Naïve Bayes
- KNN
- SVM
- Decision tree classifier
- Random forest

MULTINOMIAL NAÏVE BAYES:

Naïve Bayes classifier assumes that the value of a particular feature is independent of the value of any other feature, given class variable.

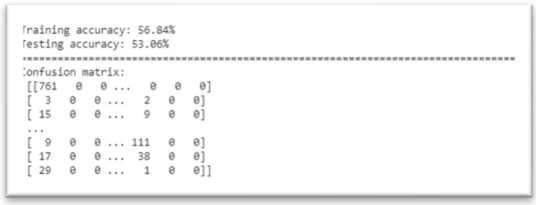
Advantages:

- Works well with text data.
- Easy to implement.
- Fast when compared to other algorithms.

Disadvantage:

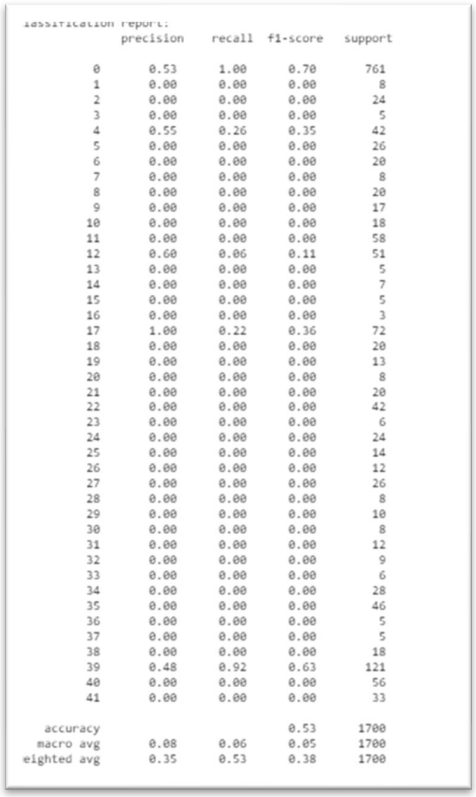
- A strong assumption about the shape of the distribution of the data.
- It cannot learn interactions between the features.
- Limited by data scarcity.

Accuracy.



```
Training accuracy: 56.84%
Testing accuracy: 53.06%
--------------------------------------------------------------------------------
Confusion matrix:
 [[761   0   0 ...   0   0   0]
 [  3   0   0 ...   2   0   0]
 [ 15   0   0 ...   9   0   0]
 ...
 [  9   0   0 ... 111   0   0]
 [ 17   0   0 ...  38   0   0]
 [ 29   0   0 ...   1   0   0]]
```

Classification report.



Classification report:

	precision	recall	f1-score	support
0	0.53	1.00	0.70	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	0.55	0.26	0.35	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	0.60	0.06	0.11	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	0.22	0.36	72
18	0.00	0.00	0.00	20
19	0.00	0.00	0.00	13
20	0.00	0.00	0.00	8
21	0.00	0.00	0.00	20
22	0.00	0.00	0.00	42
23	0.00	0.00	0.00	6
24	0.00	0.00	0.00	24
25	0.00	0.00	0.00	14
26	0.00	0.00	0.00	12
27	0.00	0.00	0.00	26
28	0.00	0.00	0.00	8
29	0.00	0.00	0.00	10
30	0.00	0.00	0.00	8
31	0.00	0.00	0.00	12
32	0.00	0.00	0.00	9
33	0.00	0.00	0.00	6
34	0.00	0.00	0.00	28
35	0.00	0.00	0.00	46
36	0.00	0.00	0.00	5
37	0.00	0.00	0.00	5
38	0.00	0.00	0.00	18
39	0.48	0.92	0.63	121
40	0.00	0.00	0.00	56
41	0.00	0.00	0.00	33
accuracy			0.53	1700
macro avg	0.08	0.06	0.05	1700
weighted avg	0.35	0.53	0.38	1700

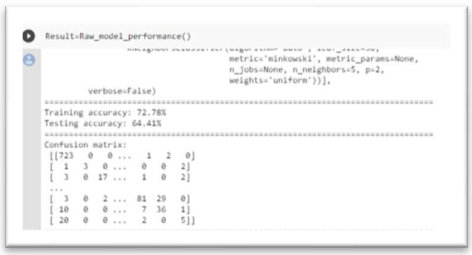
Our model is highly overfit and our test accuracy is too low.

KNN- K NEAREST NEIGHBOURS.

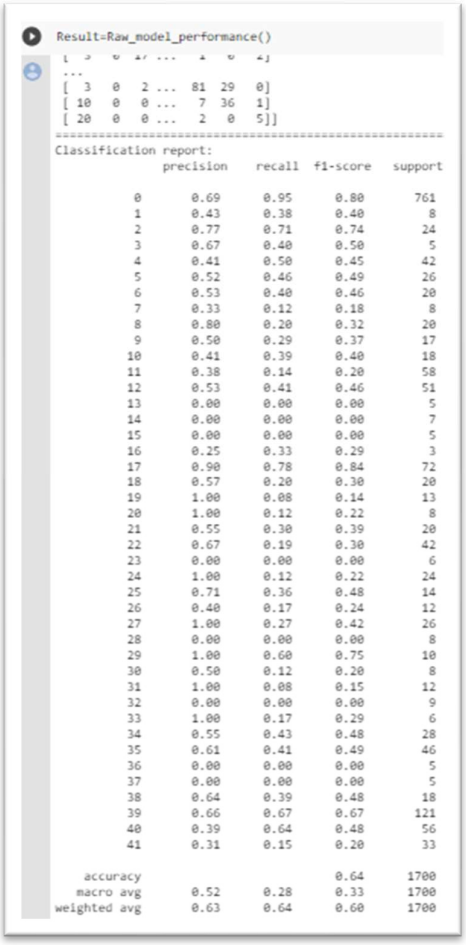
Both for classification and regression, a useful technique can be to assign weights to the contributions of the neighbours, so that the nearer neighbours contribute more to the average than the more distant ones.

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

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



CLASSIFICATION REPORT



SVM

The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space (N — the number of features) that distinctly classifies the data points.

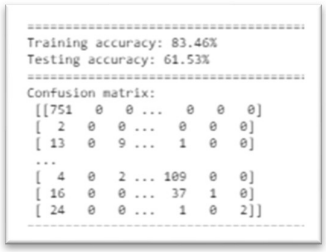
The advantages of support vector machines are based on scikit-learn page:

- Effective in high dimensional spaces.
- Still effective in cases where number of dimensions is greater than the number of samples

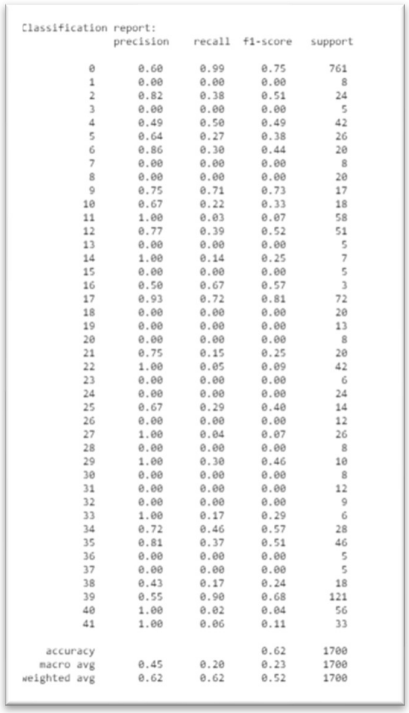
- Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
- Versatile: different Kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels.

<u>The Disadvantages of support vector machines include:</u>

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



```
==========================================
Training accuracy: 83.46%
Testing accuracy: 61.53%
==========================================
Confusion matrix:
 [[751   0   0 ...   0   0   0]
 [  2   0   0 ...   0   0   0]
 [ 13   0   9 ...   1   0   0]
 ...
 [  4   0   2 ... 109   0   0]
 [ 16   0   0 ...  37   1   0]
 [ 24   0   0 ...   1   0   2]]
------------------------------------------
```



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	0.50	0.49	42
5	0.64	0.27	0.38	26
6	0.86	0.30	0.44	20
7	0.00	0.00	0.00	8
8	0.00	0.00	0.00	20
9	0.75	0.71	0.73	17
10	0.67	0.22	0.33	18
11	1.00	0.03	0.07	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	0.50	0.67	0.57	3
17	0.93	0.72	0.81	72
18	0.00	0.00	0.00	20
19	0.00	0.00	0.00	13
20	0.00	0.00	0.00	8
21	0.75	0.15	0.25	20
22	1.00	0.05	0.09	42
23	0.00	0.00	0.00	6
24	0.00	0.00	0.00	24
25	0.67	0.29	0.40	14
26	0.00	0.00	0.00	12
27	1.00	0.04	0.07	26
28	0.00	0.00	0.00	8
29	1.00	0.30	0.46	10
30	0.00	0.00	0.00	8
31	0.00	0.00	0.00	12
32	0.00	0.00	0.00	9
33	1.00	0.17	0.29	6
34	0.72	0.46	0.57	28
35	0.81	0.37	0.51	46
36	0.00	0.00	0.00	5
37	0.00	0.00	0.00	5
38	0.43	0.17	0.24	18
39	0.55	0.90	0.68	121
40	1.00	0.02	0.04	56
41	1.00	0.06	0.11	33
accuracy			0.62	1700
macro avg	0.45	0.20	0.23	1700
weighted avg	0.62	0.62	0.52	1700

The model is overfit again.

RANDOM FOREST

Random forests or random decision forests technique is an ensemble learning method for text classification.

Advantages:

- High predictive accuracy.
- Efficient on large datasets.
- Ability to handle multiple input features without need for feature deletion.

- Prediction is based on input features considered important for classification.
- Works well with missing data still giving a better predictive accuracy.

Disadvantages:

- Not easily interpretable.
- Random forest overfit with noisy classification or regression.

Random forest classifier:

```
==================================
Training accuracy: 95.51%
Testing accuracy: 63.65%
==================================
Confusion matrix:
 [[757   0   0 ...   0   0   0]
 [  1   0   0 ...   1   0   0]
 [ 14   0   9 ...   0   1   0]
 ...
 [  5   1   2 ... 100   8   0]
 [ 16   0   0 ...  29   9   0]
 [ 24   0   0 ...   2   0   1]]
==================================
```

	precision	recall	f1-score	support
0	0.62	0.99	0.76	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	0.55	0.52	42
5	0.58	0.27	0.37	26
6	1.00	0.30	0.46	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	0.60	0.33	0.43	18
11	0.50	0.03	0.06	58
12	0.83	0.39	0.53	51
13	1.00	0.20	0.33	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	0.91	0.85	0.88	72
18	1.00	0.05	0.10	20
19	0.00	0.00	0.00	13
20	0.00	0.00	0.00	8
21	1.00	0.20	0.33	20
22	0.75	0.14	0.24	42
23	1.00	0.33	0.50	6
24	0.50	0.08	0.14	24
25	0.56	0.36	0.43	14
26	1.00	0.08	0.15	12
27	1.00	0.08	0.14	26
28	0.00	0.00	0.00	8
29	1.00	0.50	0.67	10
30	0.00	0.00	0.00	8
31	0.00	0.00	0.00	12
32	0.00	0.00	0.00	9
33	1.00	0.50	0.67	6
34	0.78	0.50	0.61	28
35	0.84	0.35	0.49	46
36	1.00	0.20	0.33	5
37	0.00	0.00	0.00	5
38	1.00	0.11	0.20	18
39	0.57	0.83	0.68	121
40	0.39	0.16	0.23	56
41	1.00	0.03	0.06	33
accuracy			0.64	1700
macro avg	0.59	0.24	0.30	1700
weighted avg	0.64	0.64	0.56	1700

DECISION TREE

Decision Tree is a very popular machine learning algorithm. Decision Tree solves the problem of machine learning by transforming the data into tree representation. Each internal node of the tree representation denotes an attribute and each leaf node denotes a class label.

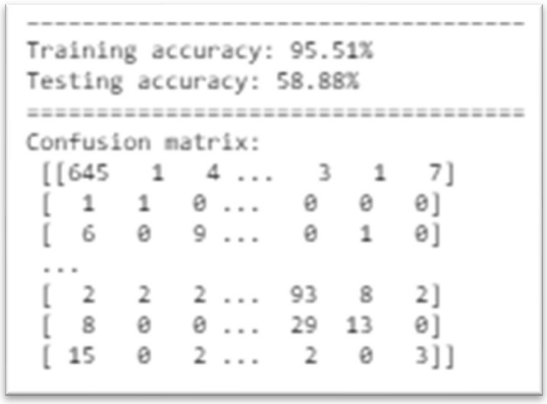
Decision tree algorithm can be used to solve both regression and classification problems:

Advantages:

- Compared to other algorithms decision trees requires less effort for data preparation during pre-processing.
- A decision tree does not require normalization of data.
- A decision tree does not require scaling of data as well.
- Missing values in the data also does NOT affect the process of building decision tree to any considerable extent.
- A Decision trees model is very intuitive and easy to explain to technical teams as well as stakeholders.

Disadvantages:

- A small change in the data can cause a large change in the structure of the decision tree causing instability.
- For a Decision tree sometimes, calculation can go far more complex compared to other algorithms.
- Decision tree often involves higher time to train the model.
- Decision tree training is relatively expensive as complexity and time taken is more.
- Decision Tree algorithm is inadequate for applying regression and predicting continuous values.



Classification report:

	precision	recall	f1-score	support
0	0.71	0.85	0.77	761
1	0.25	0.12	0.17	8
2	0.31	0.38	0.34	24
3	0.00	0.00	0.00	5
4	0.45	0.45	0.45	42
5	0.28	0.27	0.27	26
6	0.24	0.20	0.22	20
7	0.17	0.12	0.14	8
8	0.06	0.05	0.06	20
9	0.88	0.82	0.85	17
10	0.33	0.33	0.33	18
11	0.39	0.22	0.29	58
12	0.52	0.43	0.47	51
13	0.14	0.20	0.17	5
14	0.25	0.14	0.18	7
15	0.00	0.00	0.00	5
16	0.50	0.33	0.40	3
17	0.80	0.78	0.79	72
18	0.32	0.35	0.33	20
19	0.00	0.00	0.00	13
20	0.00	0.00	0.00	8
21	0.60	0.30	0.40	20
22	0.26	0.26	0.26	42
23	0.67	0.33	0.44	6
24	0.50	0.08	0.14	24
25	0.23	0.21	0.22	14
26	0.33	0.17	0.22	12
27	0.33	0.12	0.17	26
28	0.00	0.00	0.00	8
29	0.67	0.60	0.63	10
30	0.25	0.25	0.25	8
31	0.00	0.00	0.00	12
32	0.50	0.22	0.31	9
33	0.75	0.50	0.60	6
34	0.71	0.61	0.65	28
35	0.67	0.35	0.46	46
36	0.50	0.20	0.29	5
37	0.00	0.00	0.00	5
38	0.57	0.44	0.50	18
39	0.56	0.77	0.65	121
40	0.39	0.23	0.29	56
41	0.12	0.09	0.10	33
accuracy			0.59	1700
macro avg	0.36	0.28	0.31	1700
weighted avg	0.56	0.59	0.56	1700

LOGISTIC REGRESSION

Logistic regression is a predictive analysis technique.

Advantages:

- Performs well when the data is linearly separable.
- Less prone to overfitting but can overfit in high dimensional datasets, however it can be overcome by using L1 and L2 regularization techniques.

Disadvantages:

- Main limitation of Logistic Regression is the assumption of linearity between the dependent variable and the independent variables. In the real world, the data is rarely linearly separable. Most of the time data would be a jumbled mess.
- If the number of observations is lesser than the number of features, Logistic Regression should not be used, otherwise it may lead to overfit.
- Logistic Regression can only be used to predict discrete functions. Therefore, the dependent variable of Logistic Regression is restricted to the discrete number set. This restriction itself is problematic, as it is prohibitive to the prediction of continuous data.

```
==========================================================
Training accuracy: 69.40%
Testing accuracy: 62.00%
==========================================================
Confusion matrix:
 [[751   0   0 ...   0   1   0]
 [  2   0   0 ...   0   0   1]
 [ 12   0   9 ...   1   0   1]
 ...
 [  5   0   2 ... 110   0   0]
 [ 15   0   0 ...  37   2   0]
 [ 23   0   0 ...   1   0   1]]
```

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

           0       0.61      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      0.52      0.51        42
           5       0.56      0.38      0.45        26
           6       0.86      0.30      0.44        20
           7       0.00      0.00      0.00         8
           8       0.67      0.10      0.17        20
           9       1.00      0.65      0.79        17
          10       0.57      0.22      0.32        18
          11       0.78      0.12      0.21        58
          12       0.70      0.37      0.49        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       1.00      0.33      0.50         3
          17       0.91      0.72      0.81        72
          18       1.00      0.20      0.33        20
          19       0.00      0.00      0.00        13
          20       0.00      0.00      0.00         8
          21       0.33      0.05      0.09        20
          22       0.80      0.10      0.17        42
          23       0.00      0.00      0.00         6
          24       0.00      0.00      0.00        24
          25       0.67      0.29      0.40        14
          26       0.00      0.00      0.00        12
          27       1.00      0.08      0.14        26
          28       0.00      0.00      0.00         8
          29       1.00      0.30      0.46        10
          30       0.00      0.00      0.00         8
          31       0.00      0.00      0.00        12
          32       0.00      0.00      0.00         9
          33       0.00      0.00      0.00         6
          34       0.75      0.43      0.55        28
          35       0.85      0.37      0.52        46
          36       0.00      0.00      0.00         5
          37       0.00      0.00      0.00         5
          38       0.00      0.00      0.00        18
          39       0.56      0.91      0.69       121
          40       0.67      0.04      0.07        56
          41       0.14      0.03      0.05        33

    accuracy                           0.62      1700
   macro avg       0.40      0.19      0.22      1700
weighted avg       0.59      0.62      0.53      1700
```

Performance of ML models without HyperParameters:

	Train acc	Test acc	Model
0	56.838235	53.058824	Naive bayes
1	72.779412	64.411765	kNN
2	83.455882	61.529412	SVM()
3	95.514706	58.882353	decision tree classifier
4	95.514706	63.647059	RandomForestClassifier
5	69.397059	62.000000	LogisticRegression

We notice KNN and random forest giving good accuracy
KNN has good precision scores compared to random forest.

4.1.2. HYPERPARAMETER TUNING

Since we did not get good accuracies, we will now perform hyperparameter tuning.

HyperParameter

```
[ ]
def Raw_model_performance_Hyper():

    Result=pd.DataFrame(columns={"Model","Test acc","Train acc"})

    #using kNN
    train_acc,test_acc=run_classification(KNeighborsClassifier(weights='distance'), X_train, X_test, y_train, y_test)
    new_row={{"Model":"kNN_hyperTuned","Test acc":test_acc,"Train acc":train_acc}}
    Result=Result.append(new_row,ignore_index=True)

    # SVM with Linear kernel
    train_acc,test_acc=run_classification(LinearSVC(), X_train, X_test, y_train, y_test)
    new_row={{"Model":"SVM with Linear kernel","Test acc":test_acc,"Train acc":train_acc}}
    Result=Result.append(new_row,ignore_index=True)

    # SVM with RBF kernel
    train_acc,test_acc=run_classification(SVC(kernel='rbf'), X_train, X_test, y_train, y_test)
    new_row={{"Model":"SVM with RBF kernel","Test acc":test_acc,"Train acc":train_acc}}
    Result=Result.append(new_row,ignore_index=True)


    train_acc,test_acc=run_classification(LogisticRegression(solver='liblinear',C=5, penalty='l2',max_iter=1000), X_train, X_test, y_train, y_test)
    new_row={{"Model":"LogisticRegression_hyperTuned","Test acc":test_acc,"Train acc":train_acc}}
    Result=Result.append(new_row,ignore_index=True)


    return Result
```

Side by side comparison

Performance of ML models with HyperParameters:

	Model	Test acc	Train acc
0	kNN_hyperTuned	65.941176	94.411765
1	SVM with Linear kernel	68.411765	93.808824
2	SVM with RBF kernel	61.529412	83.455882
3	LogisticRegression_hyperTuned	66.529412	88.382353

With hyper parameters

Performance of ML models without HyperParameters:

	Model	Test acc	Train acc
0	Naive bayes	53.058824	56.838235
1	kNN	64.411765	72.779412
2	SVM()	61.529412	83.455882
3	decision tree classifier	58.882353	95.514706
4	RandomForestClassifier	63.647059	95.514706
5	LogisticRegression	62.000000	69.397059

Without hyper parameters

- Using hyper parameter tuning we increased the performance of Logistic Regression and KNN
- We notice that SVM with linear kernel has performed well with an accuracy of 68% on test set and 93% on train set. But still we have not overcome the problem of overfitting. Models are highly overfit.

4.2. **MILESTONE 2**- UPSAMPLING ON MACHINE LEARNING MODELS.

- Since we did not get good accuracies let us try up sampling the data.
- Up sampling to be performed on the train data.

Up sampling using the ***SMOTE*** method.

One approach to addressing imbalanced datasets is to oversample the minority class. The simplest approach involves duplicating examples in the minority class, although these examples don't add any new information to the model. Instead, new examples can be synthesized from the existing examples. This is a type of data augmentation for the minority

class and is referred to as the Synthetic Minority Oversampling Technique, or ***SMOTE*** for short.

Advantages:

- Alleviates overfitting caused by random oversampling as synthetic examples are generated rather than replication of instances.
- No loss of information.
- It's simple to implement and interpret.

Disadvantages:

- While generating synthetic examples, SMOTE does not take into consideration neighbouring examples can be from other classes. This can increase the overlapping of classes and can introduce additional noise.
- SMOTE is not very practical for high dimensional data.

```
[ ] #Upsampling data With the SMOTE function
    #Upsampling on the Train data not the test data
    sm = SMOTE(random_state = 42)

    X_res, y_res = sm.fit_resample(X_train_, y_train_)
```

LogisticRegression

```
[ ] #Running LogisticReg on Upsampled data.

[ ] Result_upsample_ML=pd.DataFrame(columns={"Model","Test acc","Train acc"})

[ ] from sklearn.linear_model import LogisticRegression
    lg=LogisticRegression()
    lg.fit(X_res,y_res)

    LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                       intercept_scaling=1, l1_ratio=None, max_iter=100,
                       multi_class='auto', n_jobs=None, penalty='l2',
                       random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                       warm_start=False)

[ ] lg.score(X_test_,y_test_)

    0.04941176470588235

[ ] train_acc= lg.score(X_res,y_res)
    test_acc = lg.score(X_test_,y_test_)
    new_row=({"Model":"LogisticRegression()","Test acc":test_acc,"Train acc":train_acc})
    Result_upsample_ML=Result_upsample_ML.append(new_row,ignore_index=True)

[ ] #Logistic Model Didnot perform well on the Upsampled data

[ ] Result_upsample_ML
```

	Train acc	Test acc	Model
0	0.317441	0.049412	LogisticRegression()

On Logistic Regression it is performing very poorly.

```
Performance of ML models On Upsampled SMOTE trained data:
```

	Model	Test acc	Train acc
0	LogisticRegression()	0.049412	0.317441
1	KNN()	0.276471	0.919559
2	Ramdom Forest	0.400000	0.915426

Noticed very poor performance on up sampling using SMOTE.

Another technique which can be used for up sampling is Random Over Sampler.

Random oversampling involves randomly duplicating examples from the minority class and adding them to the training dataset.

Advantages:

- This technique can be effective for those machine learning algorithms that are affected by a skewed distribution and where multiple duplicate examples for a given class can influence the fit of the model.
- this method leads to no information loss.

Disadvantages:

- It increases the likelihood of overfitting since it replicates the minority class events.

```
from imblearn.over_sampling import RandomOverSampler
ROS = RandomOverSampler(sampling_strategy='auto', random_state=42)
X_res, y_res  = ROS.fit_resample(X_train_, y_train_)

#X_res = pd.DataFrame(X_res)
#y_res = pd.DataFrame(y_res)
```

Performance of the ML models on Upsampled RandomOverSample Data:

	Model	Test acc	Train acc
0	LogisticRegression()	0.067647	0.293927
1	KNN()	0.338824	0.922980
2	Ramdom Forest	0.540000	0.946886

When Compared with the performance of model on SMOTE data and the RandOversample data, The RandomOverdSampled data performed quite good.

But the downsampled data perfromed very well among Upsampled (SMOTE & RandOverSample) ,downsampled and Raw data(Pre-Processed)

Noticed poor performance on this as well.

4.2.1. DOWNSAMPLING TECHNIQUE.

- We noticed very poor performance on up sampling using SMOTE and RandomOverSampler.
- We will now check how our model performs when we Down Sample them.

Downsampling on ML models

```
[13] from sklearn.utils import resample

     df_majority = df.loc[df['Assignment group'] == 'GRP_0']
     df_minority = df.loc[df['Assignment group'] != 'GRP_0']
     df_majority_downsampled = resample(df_majority,
                                replace=False,     # sample with replacement
                                n_samples=600,    # to match majority class
                                random_state=42) # reproducible results
     df = pd.concat([df_majority_downsampled, df_minority])
     max_size = df['Assignment group'].value_counts().max()

     lst = [df]
     for class_index, group in df.groupby('Assignment group'):
         lst.append(group.sample(max_size-len(group), replace=True))
     df = pd.concat(lst)
```

KNN DOWNSAMPLED.

	precision	recall	f1-score	support
0	0.52	0.37	0.43	160
1	0.91	1.00	0.95	169
2	0.97	0.82	0.89	169
3	0.99	1.00	0.99	167
4	0.70	0.55	0.61	161
5	0.84	0.87	0.85	180
6	0.96	0.92	0.94	173
7	0.97	1.00	0.98	156
8	0.95	1.00	0.97	171
9	0.96	1.00	0.98	163
10	0.90	0.98	0.94	177
11	0.69	0.70	0.70	173
12	0.86	0.60	0.71	168
13	0.99	1.00	0.99	169
14	0.99	1.00	0.99	179
15	0.95	1.00	0.98	145
16	0.99	1.00	1.00	160
17	0.70	0.72	0.71	163
18	0.85	0.91	0.88	170
19	0.91	1.00	0.95	151
20	0.96	1.00	0.98	175
21	0.93	0.95	0.94	172
22	0.77	0.76	0.76	155
23	0.95	1.00	0.98	157
24	0.94	0.98	0.96	162
25	0.90	0.93	0.92	170
26	0.93	1.00	0.96	174
27	0.87	0.96	0.91	146
28	0.99	1.00	1.00	171
29	0.99	1.00	0.99	164
30	0.98	1.00	0.99	177
31	0.97	0.80	0.88	140
32	0.98	0.70	0.82	183
33	0.99	1.00	0.99	166
34	0.29	0.84	0.44	167
35	0.54	0.54	0.54	170
36	0.99	0.70	0.82	157
37	0.97	1.00	0.99	172
38	0.92	1.00	0.96	153
39	0.77	0.35	0.48	178
40	0.77	0.27	0.40	159
41	0.79	0.69	0.74	149
ccuracy			0.86	6941
cro avg	0.88	0.86	0.85	6941
ted avg	0.88	0.86	0.85	6941

RANDOM FOREST DOWNSAMPLED

	precision	recall	f1-score	support
0	0.78	0.62	0.69	160
1	0.88	1.00	0.94	169
2	0.96	0.86	0.91	169
3	1.00	1.00	1.00	167
4	0.92	0.83	0.87	161
5	0.94	0.97	0.95	180
6	0.97	0.98	0.97	173
7	0.97	1.00	0.98	156
8	0.98	1.00	0.99	171
9	0.99	1.00	1.00	163
10	1.00	0.98	0.99	177
11	0.94	0.90	0.92	173
12	0.92	0.90	0.91	168
13	1.00	1.00	1.00	169
14	1.00	1.00	1.00	179
15	0.99	1.00	1.00	145
16	1.00	1.00	1.00	160
17	0.95	0.97	0.96	163
18	0.96	1.00	0.98	170
19	0.99	1.00	0.99	151
20	0.98	1.00	0.99	175
21	0.97	0.97	0.97	172
22	0.95	0.96	0.96	155
23	0.98	1.00	0.99	157
24	0.95	0.98	0.96	162
25	0.99	1.00	0.99	170
26	0.98	1.00	0.99	174
27	0.96	1.00	0.98	146
28	0.97	1.00	0.99	171
29	0.97	1.00	0.98	164
30	0.99	1.00	1.00	177
31	0.99	0.80	0.89	140
32	0.73	0.95	0.83	183
33	0.99	1.00	1.00	166
34	0.90	0.66	0.77	167
35	1.00	0.50	0.67	170
36	0.97	0.83	0.89	157
37	0.98	1.00	0.99	172
38	0.98	1.00	0.99	153
39	0.99	0.43	0.60	178
40	0.36	0.84	0.50	159
41	0.93	0.89	0.91	149
accuracy			0.92	6941
macro avg	0.94	0.92	0.93	6941
weighted avg	0.94	0.92	0.93	6941

PERFORMANCE COMPARISON

Performance of ML models On Downsampled trained data:

	Model	Test acc	Train acc
0	LogisticRegression()	0.215963	0.241343
1	KNN()	0.855352	0.908698
2	Ramdom Forest	0.924074	0.950723

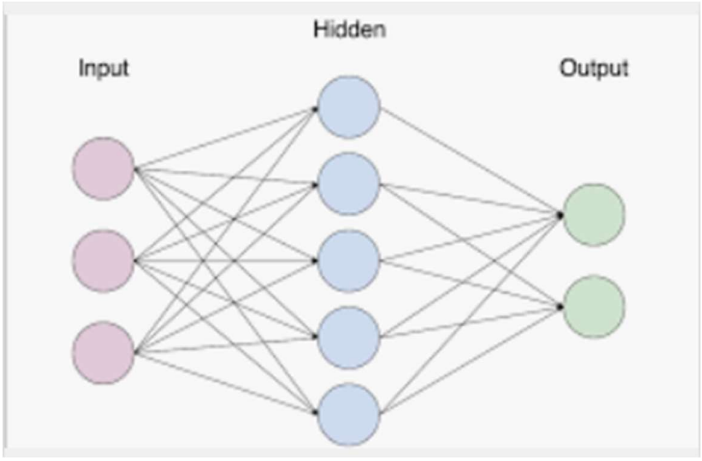
- Logistic regression is again performing poorly on down sampling as well.
- However, we notice significant improvement in KNN and random Forest.
- So, we will consider **random forest** to be our Final ML model with good accuracy and precision score.

DEEP LEARNING MODELS ON RAW-PRE-PROCESSED DATA

SIMPLE NEURAL NETWROK

Splitting the data into train and test set in the ratio 80:20

A neural network is a series of algorithms that endeavours to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. In this sense, neural networks refer to systems of neurons, either organic or artificial in nature. Neural networks can adapt to changing input; so, the network generates the best possible result without needing to redesign the output criteria. The concept of neural networks, which has its roots in artificial intelligence, is swiftly gaining popularity in the development of trading systems.



ADVANTAGES OF USING NEURAL NETWORK

1. They have the ability to learn and model non-linear and complex relationships.
2. Parallel processing ability: Artificial neural networks have numerical strength that can perform more than one job at the same time.
3. ANNs can generalize — After learning from the initial inputs and their relationships, it can infer unseen relationships on unseen data as well, thus making the model generalize and predict on unseen data.
4. Unlike many other prediction techniques, ANN does not impose any restrictions on the input variables (like how they should be distributed).
5. Gradual corruption: A network slows over time and undergoes relative degradation. The network problem does not immediately corrode.

DISADVANTAGES:

1. Hardware dependence: Artificial neural networks require processors with parallel processing power, by their structure. For this reason, the realization of the equipment is dependent.
2. Unexplained functioning of the network: This is the most important problem of ANN. When ANN gives a probing solution, it does not give a clue as to why and how. This reduces trust in the network.

3. Assurance of proper network structure: There is no specific rule for determining the structure of artificial neural networks. The appropriate network structure is achieved through experience and trial and error.
4. The difficulty of showing the problem to the network: ANNs can work with numerical information. Problems have to be translated into numerical values before being introduced to ANN. The display mechanism to be determined here will directly influence the performance of the network. This depends on the user's ability.
5. The duration of the network is unknown: The network is reduced to a certain value of the error on the sample means that the training has been completed. This value does not give us optimum results.

```
[22] # Splitting into training and test set
     from sklearn.model_selection import train_test_split

     X_train, X_test, y_train, y_test = train_test_split(X,y, random_state = 42)
```

```
     #Model buliding
     #Initialize variables

     embedding_size=50

     maxlen=769
     model = Sequential() #Sequential model

     model.add(Embedding(max_features, 50, input_length=maxlen))  #Init embedding layer with no pretrained wts
     #embedding layer takes input of vocabulary size i.e 13000, embedding dimension i.e 50 and input sequence i.e number of columns of dataset i.e 300
     model.add(Flatten()) #Use flatten layer
     model.add(Dropout(0.5)) #use dropout for regularization
     model.add(Dense(10)) #Hidden layer
     model.add(Dropout(0.3)) #use dropout for regularization
     model.add(Dense(42, activation='sigmoid')) #Output layer
     model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 769, 50)           808900
_________________________________________________________________
flatten_1 (Flatten)          (None, 38450)             0
_________________________________________________________________
dropout_1 (Dropout)          (None, 38450)             0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                384510
_________________________________________________________________
dropout_2 (Dropout)          (None, 10)                0
_________________________________________________________________
dense_2 (Dense)              (None, 42)                462
=================================================================
Total params: 1,193,872
Trainable params: 1,193,872
Non-trainable params: 0
_________________________________________________________________
```

```
[26] #Evaluate train set and then print accuracy
     results = model.evaluate(X_train, y_train)
     print('Train accuracy: ', results[1])
```

```
6375/6375 [==============================] - 0s 65us/step
Train accuracy:  0.9276862740516663
```

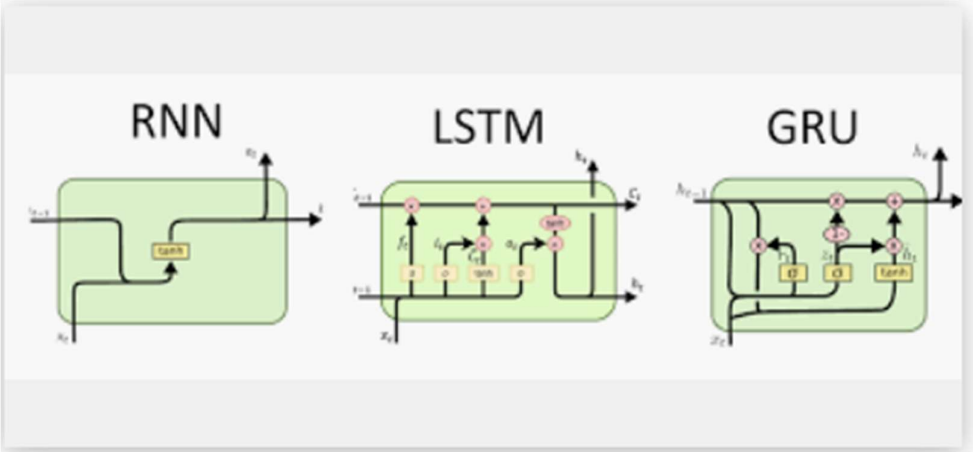
```
[27] #Evaluate test set and then print accuracy
     results = model.evaluate(X_test, y_test)
     print('Test accuracy: ', results[1])
```

```
2125/2125 [==============================] - 0s 77us/step
Test accuracy:  0.62729412317276
```

Model is highly overfit with poor performance on test set.

TO OVERCOME THE CHALLENGE OF UNDERSTANDING THE PREVIOUS OUTPUT WE USE RNNs.

LSTM is a special kind of RNN's, capable of Learning Long-term dependencies. LSTM's have a Nature of Remembering information for a long period of time is their Default behaviour.



Advantages of Recurrent Neural Network

- The main advantage of RNN over ANN is that RNN can model sequence of data (i.e. time series) so that each sample can be assumed to be dependent on previous ones.
- Recurrent neural network is even used with convolutional layers to extend the effective pixel neighbourhood.

Disadvantages of Recurrent Neural Network

- Gradient vanishing and exploding problems.
- Training an RNN is a very difficult task.
- It cannot process very long sequences if using tanh or relu as an activation function.

SIMPLE LSTM

```
[29] #Model buliding
     embedding_size=50
     model2 = Sequential()  #Sequential model
     model2.add(Embedding(max_features, embedding_size, input_length=maxlen)) #Init embedding layer with no pretrained wts
     #embedding layer takes input of vocabulary size i.e 10000, embedding dimension i.e 50 and input sequence i.e number of columns of d
     #Here I am not using pretrained glove vector
     model2.add(Dropout(0.3)) #use dropout for regularization
     model2.add(LSTM(100,return_sequences=True, dropout=0.3))  #Hidden layer with dropout Here return_sequences=true as I want to get ou
     model2.add(GlobalMaxPool1D()) #This is to get values from all states and not missing important info
     model2.add(Dense(42, activation="sigmoid")) #Final dense layer

     #Note:GlobalMaxPool1D requires output from all lSTMs therfore reurn_sequences = True, this returns output from all LSTM not just fi
     #ALso GlobalMaxPool1D takes max of all output in very special way this step is done so that every words wether at start or at The e
```

```
[30] #model summary
     model2.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 769, 50)           808900
_________________________________________________________________
dropout_3 (Dropout)          (None, 769, 50)           0
_________________________________________________________________
lstm_1 (LSTM)                (None, 769, 100)          60400
_________________________________________________________________
global_max_pooling1d_1 (Glob (None, 100)               0
_________________________________________________________________
dense_3 (Dense)              (None, 42)                4242
=================================================================
Total params: 873,542
Trainable params: 873,542
Non-trainable params: 0
_________________________________________________________________
```

```
Train on 6375 samples, validate on 2125 samples
Epoch 1/12
6375/6375 [==============================] - 205s 32ms/step - loss: 2.4471 - accuracy: 0.4717 - val_loss: 2.5143 - val_accuracy: 0.4560
Epoch 2/12
6375/6375 [==============================] - 205s 32ms/step - loss: 2.4294 - accuracy: 0.4717 - val_loss: 2.4790 - val_accuracy: 0.4560
Epoch 3/12
6375/6375 [==============================] - 205s 32ms/step - loss: 2.3909 - accuracy: 0.4717 - val_loss: 2.4384 - val_accuracy: 0.4560
Epoch 4/12
6375/6375 [==============================] - 204s 32ms/step - loss: 2.3465 - accuracy: 0.4717 - val_loss: 2.3914 - val_accuracy: 0.4560
Epoch 5/12
6375/6375 [==============================] - 204s 32ms/step - loss: 2.2769 - accuracy: 0.4717 - val_loss: 2.3152 - val_accuracy: 0.4560
Epoch 6/12
6375/6375 [==============================] - 203s 32ms/step - loss: 2.1984 - accuracy: 0.4717 - val_loss: 2.2333 - val_accuracy: 0.4560
Epoch 7/12
6375/6375 [==============================] - 203s 32ms/step - loss: 2.0982 - accuracy: 0.4867 - val_loss: 2.1033 - val_accuracy: 0.5191
Epoch 8/12
6375/6375 [==============================] - 203s 32ms/step - loss: 1.9125 - accuracy: 0.5642 - val_loss: 2.0094 - val_accuracy: 0.5459
Epoch 9/12
6375/6375 [==============================] - 203s 32ms/step - loss: 1.7908 - accuracy: 0.5787 - val_loss: 1.9129 - val_accuracy: 0.5567
Epoch 10/12
6375/6375 [==============================] - 203s 32ms/step - loss: 1.6713 - accuracy: 0.5933 - val_loss: 1.8510 - val_accuracy: 0.5628
Epoch 11/12
6375/6375 [==============================] - 202s 32ms/step - loss: 1.5708 - accuracy: 0.6063 - val_loss: 1.8265 - val_accuracy: 0.5548
Epoch 12/12
6375/6375 [==============================] - 204s 32ms/step - loss: 1.4876 - accuracy: 0.6162 - val_loss: 1.7943 - val_accuracy: 0.5652
```

```
[34] #Evaluate test set and then print accuracy
     results2 = model2.evaluate(X_test, y_test)
     print('Test accuracy: ', results2[1])
```

```
2125/2125 [==============================] - 9s 4ms/step
Test accuracy:  0.5651764869689941
```

```
[35] #Evaluate train set and then print accuracy
     results2 = model2.evaluate(X_train, y_train)
     print('Train accuracy: ', results2[1])
```

```
6375/6375 [==============================] - 26s 4ms/step
Train accuracy:  0.6240000128746033
```

LSTM WITH PRE-TRAINED GLOVE EMBEDDINGS.

GloVe

GloVe is a word vector technique that rode the wave of word vectors after a brief silence. Just to refresh, word vectors put words to a nice vector space, where similar words cluster together and different words repel.

Advantages:

1. The goal of Glove is very straightforward, i.e., to enforce the word vectors to capture sub-linear relationships in the vector space. Thus, it proves to perform better than Word2vec in the word analogy tasks.
2. Glove adds some more practical meaning into word vectors by considering the relationships between word pair and word pair rather than word and word.
3. Glove gives lower weight for highly frequent word pairs so as to prevent the meaningless stop words like “the”, “an” will not dominate the training progress.

Disadvantages:

The model is trained on the co-occurrence matrix of words, which takes a lot of memory for storage. Especially, if you change the hyper-parameters related to the co-occurrence matrix, you have to reconstruct the matrix again, which is very time-consuming.

```
[40] #model building
     model3 = Sequential()  #Sequential model
     model3.add(Embedding(max_features, 200, input_length=maxlen, weights=[embedding_matrix], trainable=Tru
     model3.add(Dropout(0.5)) #use dropout for regularization embedding_matrix
     model3.add(LSTM(100,return_sequences=True, dropout=0.4))  #Hidden layer with dropout Here return_seque
     model3.add(GlobalMaxPool1D()) #This is to get values from all states and not missing important info
     model3.add(Dense(42, activation="sigmoid")) #Final dense layer
     #Note:GlobalMaxPool1D requires output from all lSTMs therfore return_sequences = True, this returns ou
     #ALso GlobalMaxPool1D takes max of all output in very special way this step is done so that every word
```

```
[41] #To see what our model have, number of layer , output shape ,etc
     #model summary
     model3.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, 769, 200)          3235600
_________________________________________________________________
dropout_4 (Dropout)          (None, 769, 200)          0
_________________________________________________________________
lstm_2 (LSTM)                (None, 769, 100)          120400
_________________________________________________________________
global_max_pooling1d_2 (Glob (None, 100)               0
_________________________________________________________________
dense_4 (Dense)              (None, 42)                4242
=================================================================
Total params: 3,360,242
Trainable params: 3,360,242
Non-trainable params: 0
_________________________________________________________________
```

```
Train on 6375 samples, validate on 2125 samples
Epoch 1/12
6375/6375 [==============================] - 201s 31ms/step - loss: 2.8940 - acc: 0.4552 - val_loss: 2.4905 - val_acc: 0.4560
Epoch 2/12
6375/6375 [==============================] - 201s 32ms/step - loss: 2.3681 - acc: 0.4717 - val_loss: 2.3592 - val_acc: 0.4560
Epoch 3/12
6375/6375 [==============================] - 201s 32ms/step - loss: 2.2406 - acc: 0.4717 - val_loss: 2.2418 - val_acc: 0.4560
Epoch 4/12
6375/6375 [==============================] - 201s 31ms/step - loss: 2.1160 - acc: 0.4717 - val_loss: 2.1086 - val_acc: 0.4560
Epoch 5/12
6375/6375 [==============================] - 200s 31ms/step - loss: 1.9288 - acc: 0.5445 - val_loss: 1.8752 - val_acc: 0.5482
Epoch 6/12
6375/6375 [==============================] - 201s 32ms/step - loss: 1.7841 - acc: 0.5735 - val_loss: 1.7820 - val_acc: 0.5624
Epoch 7/12
6375/6375 [==============================] - 202s 32ms/step - loss: 1.6369 - acc: 0.5945 - val_loss: 1.6819 - val_acc: 0.5831
Epoch 8/12
6375/6375 [==============================] - 200s 31ms/step - loss: 1.5406 - acc: 0.6082 - val_loss: 1.6169 - val_acc: 0.5882
Epoch 9/12
6375/6375 [==============================] - 201s 32ms/step - loss: 1.4412 - acc: 0.6202 - val_loss: 1.5581 - val_acc: 0.5901
Epoch 10/12
6375/6375 [==============================] - 201s 31ms/step - loss: 1.3580 - acc: 0.6340 - val_loss: 1.5096 - val_acc: 0.5953
Epoch 11/12
6375/6375 [==============================] - 201s 32ms/step - loss: 1.2740 - acc: 0.6515 - val_loss: 1.4945 - val_acc: 0.6061
Epoch 12/12
6375/6375 [==============================] - 202s 32ms/step - loss: 1.2021 - acc: 0.6729 - val_loss: 1.4789 - val_acc: 0.6024
```

```
[48] #Evaluate test set and then print accuracy
     results3 = model3.evaluate(X_test, y_test)
     print('Test accuracy: ', results3[1])
```

```
2125/2125 [==============================] - 9s 4ms/step
Test accuracy:  0.6023529171943665
```

```
[49] #Evaluate train set and then print accuracy
     results3 = model2.evaluate(X_train, y_train)
     print('Train accuracy: ', results3[1])
```

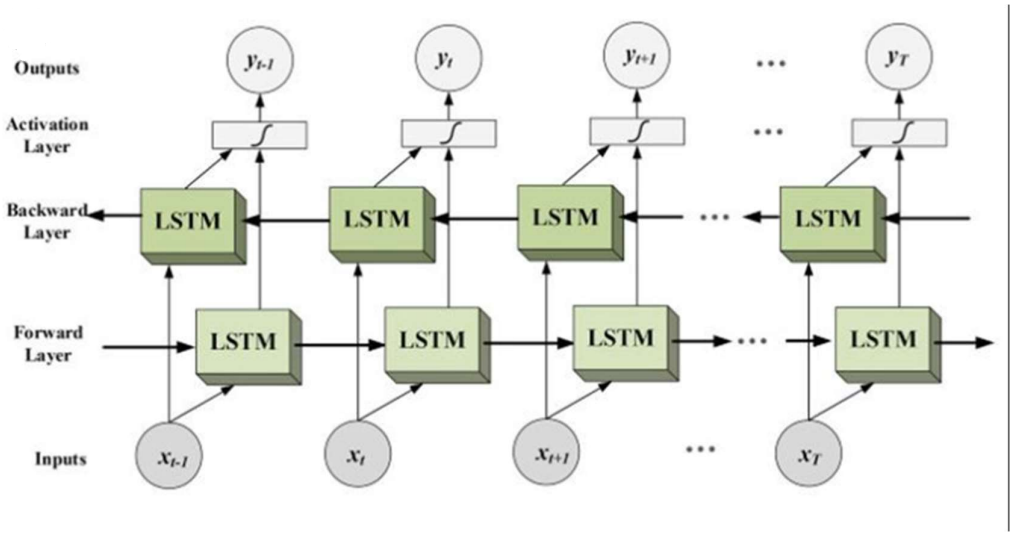
```
6375/6375 [==============================] - 26s 4ms/step
Train accuracy:  0.6240000128746033
```

Accuracies are still low

BI-LSTM WITH PRE-TRAINED GLOVE EMBEDDINGS.

Bi-LSTM is the possibility for Bi-LSTM to leverage future context chunks to learn better representations of single words. There is no special training step or units added, the idea is just to read a sentence forward and backward to capture more information.

A bidirectional LSTM (BiLSTM) layer learns bidirectional long-term dependencies between time steps of time series or sequence data. These dependencies can be useful when you want the network to learn from the complete time series at each time step.



```
[52] #Model buliding
     model4 = Sequential()  #Sequential model
     model4.add(Embedding(max_features, 200, input_length = maxlen, weights = [embedding_matrix]))
     #embedding layer takes input of vocabulary size, embedding dimension i.e 200 and input sequence i.e number of
     #Here I am using pretrained glove vector stored in variable
     model4.add(Dropout(0.5)) #use dropout for regularization embedding_matrix
     model4.add(Bidirectional(LSTM(100, dropout=0.5)))  #Hidden layer with dropout Here return_sequences=true as I
     model4.add(Dense(42, activation="sigmoid")) #Final dense layer
```

```
[53] #print model summary
     model4.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_4 (Embedding)      (None, 769, 200)          3235600
_________________________________________________________________
dropout_5 (Dropout)          (None, 769, 200)          0
_________________________________________________________________
bidirectional_1 (Bidirection (None, 200)               240800
_________________________________________________________________
dense_5 (Dense)              (None, 42)                8442
=================================================================
Total params: 3,484,842
Trainable params: 3,484,842
Non-trainable params: 0
_________________________________________________________________
```

```
Train on 5100 samples, validate on 1275 samples
Epoch 1/12
5100/5100 [==============================] - 318s 62ms/step - loss: 2.6399 - acc: 0.4676 - val_loss: 2.1684 - val_acc: 0.4878
Epoch 2/12
5100/5100 [==============================] - 320s 63ms/step - loss: 2.1561 - acc: 0.4676 - val_loss: 1.9054 - val_acc: 0.4878
Epoch 3/12
5100/5100 [==============================] - 320s 63ms/step - loss: 2.0242 - acc: 0.4676 - val_loss: 1.8526 - val_acc: 0.4878
Epoch 4/12
5100/5100 [==============================] - 321s 63ms/step - loss: 1.9155 - acc: 0.4676 - val_loss: 1.7564 - val_acc: 0.4878
Epoch 5/12
5100/5100 [==============================] - 321s 63ms/step - loss: 1.8186 - acc: 0.4676 - val_loss: 1.6977 - val_acc: 0.4878
Epoch 6/12
5100/5100 [==============================] - 323s 63ms/step - loss: 1.7451 - acc: 0.4676 - val_loss: 1.6441 - val_acc: 0.4878
Epoch 7/12
5100/5100 [==============================] - 319s 63ms/step - loss: 1.6528 - acc: 0.5049 - val_loss: 1.5511 - val_acc: 0.5976
Epoch 8/12
5100/5100 [==============================] - 321s 63ms/step - loss: 1.5161 - acc: 0.5900 - val_loss: 1.4968 - val_acc: 0.6047
Epoch 9/12
5100/5100 [==============================] - 322s 63ms/step - loss: 1.4124 - acc: 0.6125 - val_loss: 1.4518 - val_acc: 0.6141
Epoch 10/12
5100/5100 [==============================] - 319s 63ms/step - loss: 1.2897 - acc: 0.6429 - val_loss: 1.4404 - val_acc: 0.6149
Epoch 11/12
5100/5100 [==============================] - 319s 63ms/step - loss: 1.1987 - acc: 0.6680 - val_loss: 1.4292 - val_acc: 0.6259
Epoch 12/12
5100/5100 [==============================] - 320s 63ms/step - loss: 1.1113 - acc: 0.6904 - val_loss: 1.4261 - val_acc: 0.6282
```

```
[58] #Evaluate test set and then print accuracy
     results4 = model4.evaluate(X_test, y_test)
     print('Test accuracy: ', results4[1])
```

```
2125/2125 [==============================] - 17s 8ms/step
Test accuracy:  0.5868235230445862
```

```
[59] #Evaluate train set and then print accuracy
     results4 = model4.evaluate(X_train, y_train)
     print('Train accuracy: ', results4[1])
```

```
6375/6375 [==============================] - 49s 8ms/step
Train accuracy:  0.7105882167816162
```

4.3. NEURAL NETWORKS ON UP SAMPLED DATA USING SMOTE.

Using simple neural network to build our first model.

```
#Model buliding
#Initialize variables

embedding_size=100
batch_size =100
epochs = 10
maxlen=769
model = Sequential() #Sequential model

model.add(Embedding(MAX_NB_WORDS, embedding_size, input_length=maxlen))  #Init embedding layer with no pretrained wts
#embedding layer takes input of vocabulary size i.e 13000, embedding dimension i.e 50 and input sequence i.e number of columns of dataset i.e 300
model.add(Flatten()) #Use flatten layer
model.add(Dropout(0.5)) #use dropout for regularization
model.add(Dense(10)) #Hidden layer
model.add(Dropout(0.3)) #use dropout for regularization
model.add(Dense(42, activation='sigmoid')) #Output layer
```

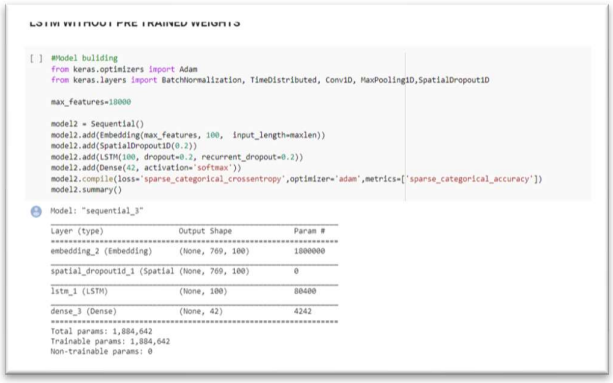
```
#To see what our model have, number of layer , output shape ,etc
#model summary
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 769, 100)          1800000
_________________________________________________________________
flatten_1 (Flatten)          (None, 76900)             0
_________________________________________________________________
dropout_1 (Dropout)          (None, 76900)             0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                769010
_________________________________________________________________
dropout_2 (Dropout)          (None, 10)                0
_________________________________________________________________
dense_2 (Dense)              (None, 42)                462
=================================================================
Total params: 2,569,472
Trainable params: 2,569,472
Non-trainable params: 0
```

```
Train on 135030 samples, validate on 1700 samples
Epoch 1/15
135030/135030 [==============================] - 13s 100us/step - loss: 2.4909 - accuracy: 0.3013 - val_loss: 2.4680 - val_accuracy: 0.3782
Epoch 2/15
135030/135030 [==============================] - 13s 99us/step - loss: 2.4060 - accuracy: 0.3238 - val_loss: 2.3754 - val_accuracy: 0.4171
Epoch 3/15
135030/135030 [==============================] - 13s 99us/step - loss: 2.3357 - accuracy: 0.3425 - val_loss: 2.4751 - val_accuracy: 0.3759
Epoch 4/15
135030/135030 [==============================] - 13s 98us/step - loss: 0.8243 - accuracy: 0.1449 - val_loss: 1.1921e-07 - val_accuracy: 0.4476
Epoch 5/15
135030/135030 [==============================] - 13s 99us/step - loss: 1.1921e-07 - accuracy: 0.0238 - val_loss: 1.1921e-07 - val_accuracy: 0.4476
Epoch 6/15
135030/135030 [==============================] - 13s 98us/step - loss: 1.1921e-07 - accuracy: 0.0238 - val_loss: 1.1921e-07 - val_accuracy: 0.4476
Epoch 7/15
135030/135030 [==============================] - 13s 98us/step - loss: 1.1921e-07 - accuracy: 0.0238 - val_loss: 1.1921e-07 - val_accuracy: 0.4476
Epoch 8/15
135030/135030 [==============================] - 13s 99us/step - loss: 1.1921e-07 - accuracy: 0.0238 - val_loss: 1.1921e-07 - val_accuracy: 0.4476
Epoch 9/15
135030/135030 [==============================] - 13s 100us/step - loss: 1.1921e-07 - accuracy: 0.0238 - val_loss: 1.1921e-07 - val_accuracy: 0.4476
Epoch 10/15
135030/135030 [==============================] - 13s 99us/step - loss: 1.1921e-07 - accuracy: 0.0238 - val_loss: 1.1921e-07 - val_accuracy: 0.4476
Epoch 11/15
135030/135030 [==============================] - 13s 99us/step - loss: 1.1921e-07 - accuracy: 0.0238 - val_loss: 1.1921e-07 - val_accuracy: 0.4476
Epoch 12/15
135030/135030 [==============================] - 13s 99us/step - loss: 1.1921e-07 - accuracy: 0.0238 - val_loss: 1.1921e-07 - val_accuracy: 0.4476
Epoch 13/15
135030/135030 [==============================] - 13s 97us/step - loss: 1.1921e-07 - accuracy: 0.0238 - val_loss: 1.1921e-07 - val_accuracy: 0.4476
Epoch 14/15
135030/135030 [==============================] - 13s 98us/step - loss: 1.1921e-07 - accuracy: 0.0238 - val_loss: 1.1921e-07 - val_accuracy: 0.4476
Epoch 15/15
135030/135030 [==============================] - 13s 97us/step - loss: 1.1921e-07 - accuracy: 0.0238 - val_loss: 1.1921e-07 - val_accuracy: 0.4476
```

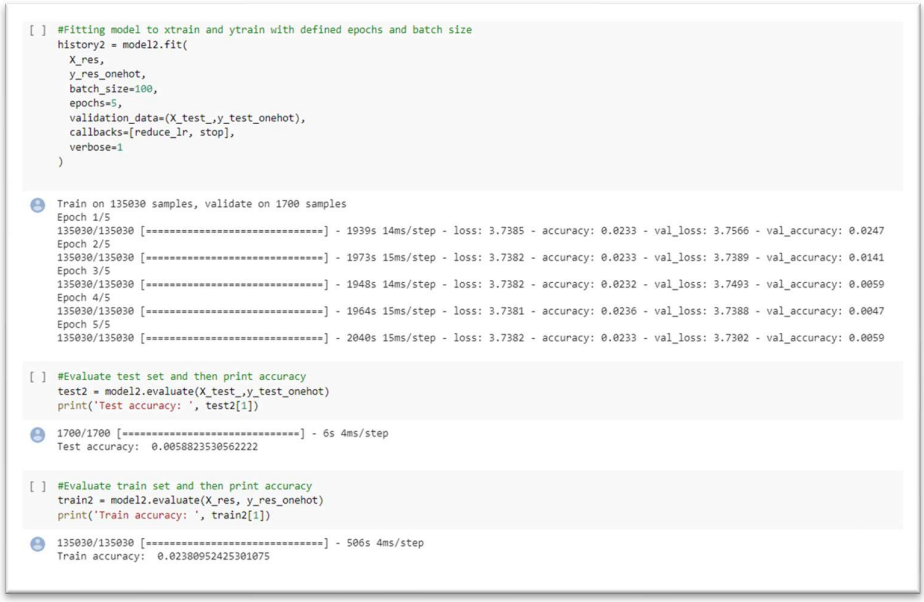
We achieved a test accuracy of about 44% which is not great so let us try other models.

LSTM WITHOUT PRE-TRAINED WEIGHTS.



```
[ ] #Model bulding
    from keras.optimizers import Adam
    from keras.layers import BatchNormalization, TimeDistributed, Conv1D, MaxPooling1D,SpatialDropout1D

    max_features=18000

    model2 = Sequential()
    model2.add(Embedding(max_features, 100,  input_length=maxlen))
    model2.add(SpatialDropout1D(0.2))
    model2.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2))
    model2.add(Dense(42, activation='softmax'))
    model2.compile(loss='sparse_categorical_crossentropy',optimizer='adam',metrics=['sparse_categorical_accuracy'])
    model2.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 769, 100)          1800000
_________________________________________________________________
spatial_dropout1d_1 (Spatial (None, 769, 100)          0
_________________________________________________________________
lstm_1 (LSTM)                (None, 100)               80400
_________________________________________________________________
dense_3 (Dense)              (None, 42)                4242
=================================================================
Total params: 1,884,642
Trainable params: 1,884,642
Non-trainable params: 0
```



```
[ ] #Fitting model to xtrain and ytrain with defined epochs and batch size
    history2 = model2.fit(
      X_res,
      y_res_onehot,
      batch_size=100,
      epochs=5,
      validation_data=(X_test_,y_test_onehot),
      callbacks=[reduce_lr, stop],
      verbose=1
    )
```

```
Train on 135030 samples, validate on 1700 samples
Epoch 1/5
135030/135030 [==============================] - 1939s 14ms/step - loss: 3.7385 - accuracy: 0.0233 - val_loss: 3.7566 - val_accuracy: 0.0247
Epoch 2/5
135030/135030 [==============================] - 1973s 15ms/step - loss: 3.7382 - accuracy: 0.0233 - val_loss: 3.7389 - val_accuracy: 0.0141
Epoch 3/5
135030/135030 [==============================] - 1948s 14ms/step - loss: 3.7382 - accuracy: 0.0232 - val_loss: 3.7493 - val_accuracy: 0.0059
Epoch 4/5
135030/135030 [==============================] - 1964s 15ms/step - loss: 3.7381 - accuracy: 0.0236 - val_loss: 3.7388 - val_accuracy: 0.0047
Epoch 5/5
135030/135030 [==============================] - 2040s 15ms/step - loss: 3.7382 - accuracy: 0.0233 - val_loss: 3.7302 - val_accuracy: 0.0059
```

```
[ ] #Evaluate test set and then print accuracy
    test2 = model2.evaluate(X_test_,y_test_onehot)
    print('Test accuracy: ', test2[1])
```

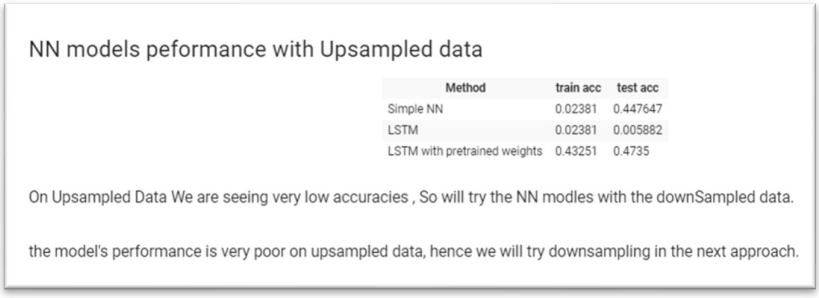
```
1700/1700 [==============================] - 6s 4ms/step
Test accuracy:  0.0058823530562222
```

```
[ ] #Evaluate train set and then print accuracy
    train2 = model2.evaluate(X_res, y_res_onehot)
    print('Train accuracy: ', train2[1])
```

```
135030/135030 [==============================] - 506s 4ms/step
Train accuracy:  0.02380952425301075
```

very low accuracies observed from the model.

FINAL COMPARISION



NN models peformance with Upsampled data

Method	train acc	test acc
Simple NN	0.02381	0.447647
LSTM	0.02381	0.005882
LSTM with pretrained weights	0.43251	0.4735

On Upsampled Data We are seeing very low accuracies , So will try the NN modles with the downSampled data.

the model's performance is very poor on upsampled data, hence we will try downsampling in the next approach.

Deep learning models are not performing well here too.

Let us try another approach.

4.3.1. DEEP LEARNING MODELS BY DOWN SAMPLING.

As we did not achieve good accuracies in the previous approach, we will performing down sampling in this approach.

```
# Downsample majority class i.e GRP_0
from sklearn.utils import resample

df_majority = df.loc[df['Assignment group'] == 'GRP_0']
df_minority = df.loc[df['Assignment group'] != 'GRP_0']
df_majority_downsampled = resample(df_majority,
                                   replace=False,     # sample with replacement
                                   n_samples=600,     # to match majority class
                                   random_state=42) # reproducible results
```

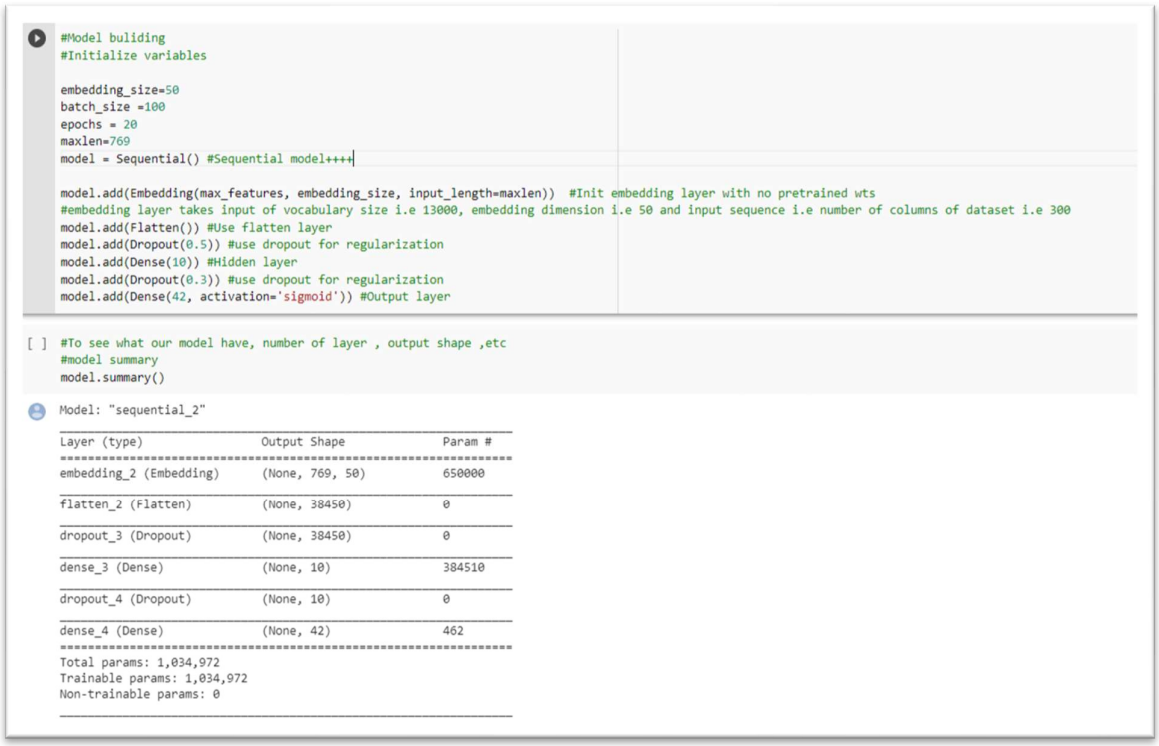
```
#concatinating majority and minority class
df = pd.concat([df_majority_downsampled, df_minority])
```

```
max_size = df['Assignment group'].value_counts().max()

lst = [df]
for class_index, group in df.groupby('Assignment group'):
    lst.append(group.sample(max_size-len(group), replace=True))
df = pd.concat(lst)
```

- We are down sampling group 0 and creating a balanced dataset by concatenating the df_majority and df_minority, setting the limit as 600 to match the majority class.
- Splitting the data into train and test set in the ratio 80:20

All the models built in this approach are saved in h5 files.

SIMPLE NEURAL NETWORK MODEL



```
#Model buliding
#Initialize variables

embedding_size=50
batch_size =100
epochs = 20
maxlen=769
model = Sequential() #Sequential model+++

model.add(Embedding(max_features, embedding_size, input_length=maxlen))  #Init embedding layer with no pretrained wts
#embedding layer takes input of vocabulary size i.e 13000, embedding dimension i.e 50 and input sequence i.e number of columns of dataset i.e 300
model.add(Flatten()) #Use flatten layer
model.add(Dropout(0.5)) #use dropout for regularization
model.add(Dense(10)) #Hidden layer
model.add(Dropout(0.3)) #use dropout for regularization
model.add(Dense(42, activation='sigmoid')) #Output layer
```

```
#To see what our model have, number of layer , output shape ,etc
#model summary
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 769, 50)           650000
_________________________________________________________________
flatten_2 (Flatten)          (None, 38450)             0
_________________________________________________________________
dropout_3 (Dropout)          (None, 38450)             0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                384510
_________________________________________________________________
dropout_4 (Dropout)          (None, 10)                0
_________________________________________________________________
dense_4 (Dense)              (None, 42)                462
=================================================================
Total params: 1,034,972
Trainable params: 1,034,972
Non-trainable params: 0
_________________________________________________________________
```

```
/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/indexed_slices.py:434: UserWarning: Converting sparse IndexedSlices to a dense
  "Converting sparse IndexedSlices to a dense Tensor of unknown shape. "
Train on 20821 samples, validate on 6941 samples
Epoch 1/15
20821/20821 [==============================] - 16s 766us/step - loss: 3.5870 - accuracy: 0.0729 - val_loss: 3.2279 - val_accuracy: 0.1359
Epoch 2/15
20821/20821 [==============================] - 16s 761us/step - loss: 2.8677 - accuracy: 0.1804 - val_loss: 2.1541 - val_accuracy: 0.5760
Epoch 3/15
20821/20821 [==============================] - 16s 759us/step - loss: 2.1732 - accuracy: 0.3658 - val_loss: 1.3553 - val_accuracy: 0.7686
Epoch 4/15
20821/20821 [==============================] - 16s 757us/step - loss: 1.7340 - accuracy: 0.4755 - val_loss: 0.9525 - val_accuracy: 0.8336
Epoch 5/15
20821/20821 [==============================] - 16s 755us/step - loss: 1.4849 - accuracy: 0.5571 - val_loss: 0.7193 - val_accuracy: 0.8604
Epoch 6/15
20821/20821 [==============================] - 16s 756us/step - loss: 1.3165 - accuracy: 0.6135 - val_loss: 0.5796 - val_accuracy: 0.8721
Epoch 7/15
20821/20821 [==============================] - 16s 751us/step - loss: 1.1986 - accuracy: 0.6505 - val_loss: 0.4879 - val_accuracy: 0.8921
Epoch 8/15
20821/20821 [==============================] - 15s 744us/step - loss: 1.1189 - accuracy: 0.6733 - val_loss: 0.4417 - val_accuracy: 0.8957
Epoch 9/15
20821/20821 [==============================] - 16s 757us/step - loss: 1.0544 - accuracy: 0.6926 - val_loss: 0.3991 - val_accuracy: 0.9065
Epoch 10/15
20821/20821 [==============================] - 16s 755us/step - loss: 0.9959 - accuracy: 0.7056 - val_loss: 0.3693 - val_accuracy: 0.9071
Epoch 11/15
20821/20821 [==============================] - 16s 761us/step - loss: 0.9424 - accuracy: 0.7278 - val_loss: 0.3420 - val_accuracy: 0.9130
Epoch 12/15
20821/20821 [==============================] - 16s 765us/step - loss: 0.9084 - accuracy: 0.7359 - val_loss: 0.3207 - val_accuracy: 0.9172
Epoch 13/15
20821/20821 [==============================] - 16s 759us/step - loss: 0.8774 - accuracy: 0.7434 - val_loss: 0.3079 - val_accuracy: 0.9206
Epoch 14/15
20821/20821 [==============================] - 16s 767us/step - loss: 0.8409 - accuracy: 0.7515 - val_loss: 0.2948 - val_accuracy: 0.9223
Epoch 15/15
20821/20821 [==============================] - 16s 771us/step - loss: 0.8328 - accuracy: 0.7552 - val_loss: 0.2894 - val_accuracy: 0.9231
```

```
[ ] #Evaluate test set and then print accuracy
    test = model.evaluate(X_test, y_test)
    print('Test accuracy: ', test[1])
```

```
6941/6941 [==============================] - 1s 78us/step
Test accuracy:  0.9230658411979675
```

```
[ ] #Evaluate train set and then print accuracy
    train = model.evaluate(X_train, y_train)
    print('Train accuracy: ',train[1])
```

```
20821/20821 [==============================] - 2s 77us/step
Train accuracy:  0.9382354617118835
```

```
[ ] #Store the accuracy results for each model in a dataframe for final comparison
    results = pd.DataFrame({'Method':['NN'], 'train acc': train[1], 'test acc':test[1]},index={'1'})
    results = results[['Method', 'train acc','test acc']]
    results
```

	Method	train acc	test acc
1	NN	0.938235	0.923066

	precision	recall	f1-score	support
0	0.72	0.76	0.74	160
1	0.88	1.00	0.94	169
2	1.00	0.88	0.93	169
3	1.00	1.00	1.00	167
4	0.84	0.88	0.86	161
5	0.97	0.99	0.98	180
6	0.96	0.94	0.95	173
7	0.99	1.00	0.99	156
8	0.99	1.00	1.00	171
9	1.00	1.00	1.00	163
10	0.97	0.97	0.97	177
11	0.97	0.87	0.92	173
12	0.97	0.85	0.90	168
13	1.00	1.00	1.00	169
14	1.00	1.00	1.00	179
15	1.00	1.00	1.00	145
16	1.00	1.00	1.00	160
17	0.99	0.98	0.99	163
18	0.99	1.00	0.99	170
19	0.99	1.00	1.00	151
20	0.99	1.00	1.00	175
21	0.99	0.95	0.97	172
22	0.95	0.99	0.97	155
23	0.99	1.00	0.99	157
24	1.00	0.98	0.99	162
25	1.00	0.97	0.99	170
26	0.98	1.00	0.99	174
27	0.93	1.00	0.96	146
28	1.00	1.00	1.00	171
29	0.99	1.00	1.00	164
30	1.00	1.00	1.00	177
31	0.98	0.78	0.87	140
32	0.78	0.95	0.86	183
33	1.00	1.00	1.00	166
34	0.82	0.69	0.75	167
35	1.00	0.51	0.67	170
36	0.97	0.80	0.88	157
37	1.00	0.97	0.98	172
38	0.99	1.00	1.00	153
39	0.92	0.38	0.53	178
40	0.37	0.86	0.51	159
41	0.70	0.85	0.77	149
accuracy			0.92	6941
macro avg	0.94	0.92	0.92	6941
weighted avg	0.94	0.92	0.93	6941

We have noticed good accuracies but the precision scores for the model is not great. Saving the model and its weights.

```
[ ]  # keras library import  for Saving and loading model and weights

    from keras.models import model_from_json
    from keras.models import load_model

    # serialize model to JSON
    #  the keras model which is trained is defined as 'model' in this example
    model_json = model.to_json()


    with open("model_num.json", "w") as json_file:
        json_file.write(model_json)

    # serialize weights to HDF5
    model.save_weights("nn.h5")
```

LSTM WITHOUT PRE-TRAINED WEIGHTS

```
[ ] #Model buliding

    from keras.optimizers import Adam
    from keras.layers import BatchNormalization, TimeDistributed, Conv1D, MaxPooling1D,SpatialDropout1D
    model2 = Sequential()
    model2.add(Embedding(max_features, 100,  input_length=maxlen))
    model2.add(SpatialDropout1D(0.2))
    model2.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2))
    model2.add(Dense(42, activation='softmax'))
    model2.compile(loss='sparse_categorical_crossentropy',optimizer='adam',metrics=['sparse_categorical_accuracy'])
    model2.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_6 (Embedding)      (None, 769, 100)          1300000
_________________________________________________________________
spatial_dropout1d_1 (Spatial (None, 769, 100)          0
_________________________________________________________________
lstm_3 (LSTM)                (None, 100)               80400
_________________________________________________________________
dense_7 (Dense)              (None, 42)                4242
=================================================================
Total params: 1,384,642
Trainable params: 1,384,642
Non-trainable params: 0
_________________________________________________________________
```

```
Train on 20821 samples, validate on 6941 samples
Epoch 1/20
20821/20821 [==============================] - 453s 22ms/step - loss: 1.0467 - accuracy: 0.7344 - val_loss: 0.6251 - val_accuracy: 0.8441
Epoch 2/20
20821/20821 [==============================] - 453s 22ms/step - loss: 0.5578 - accuracy: 0.8529 - val_loss: 0.4227 - val_accuracy: 0.8803
Epoch 3/20
20821/20821 [==============================] - 460s 22ms/step - loss: 0.3799 - accuracy: 0.8940 - val_loss: 0.3388 - val_accuracy: 0.8980
Epoch 4/20
20821/20821 [==============================] - 462s 22ms/step - loss: 0.3029 - accuracy: 0.9108 - val_loss: 0.3144 - val_accuracy: 0.9088
Epoch 5/20
20821/20821 [==============================] - 452s 22ms/step - loss: 0.2644 - accuracy: 0.9190 - val_loss: 0.2801 - val_accuracy: 0.9179
Epoch 6/20
20821/20821 [==============================] - 453s 22ms/step - loss: 0.2313 - accuracy: 0.9274 - val_loss: 0.2672 - val_accuracy: 0.9170
Epoch 7/20
20821/20821 [==============================] - 454s 22ms/step - loss: 0.2172 - accuracy: 0.9300 - val_loss: 0.2590 - val_accuracy: 0.9244
Epoch 8/20
20821/20821 [==============================] - 456s 22ms/step - loss: 0.2038 - accuracy: 0.9325 - val_loss: 0.2501 - val_accuracy: 0.9234
Epoch 9/20
20821/20821 [==============================] - 452s 22ms/step - loss: 0.1934 - accuracy: 0.9361 - val_loss: 0.2508 - val_accuracy: 0.9251
Epoch 10/20
20821/20821 [==============================] - 458s 22ms/step - loss: 0.1874 - accuracy: 0.9357 - val_loss: 0.2511 - val_accuracy: 0.9249
Epoch 11/20
20821/20821 [==============================] - 453s 22ms/step - loss: 0.1806 - accuracy: 0.9365 - val_loss: 0.2482 - val_accuracy: 0.9268
Epoch 12/20
20821/20821 [==============================] - 452s 22ms/step - loss: 0.1794 - accuracy: 0.9378 - val_loss: 0.2540 - val_accuracy: 0.9257
Epoch 13/20
20821/20821 [==============================] - 453s 22ms/step - loss: 0.1720 - accuracy: 0.9383 - val_loss: 0.2547 - val_accuracy: 0.9255
Epoch 14/20
20821/20821 [==============================] - 453s 22ms/step - loss: 0.1726 - accuracy: 0.9391 - val_loss: 0.2502 - val_accuracy: 0.9213

Epoch 00014: ReduceLROnPlateau reducing learning rate to 0.00020000000949949026.
Epoch 15/20
20821/20821 [==============================] - 451s 22ms/step - loss: 0.1674 - accuracy: 0.9404 - val_loss: 0.2456 - val_accuracy: 0.9277
Epoch 16/20
20821/20821 [==============================] - 452s 22ms/step - loss: 0.1629 - accuracy: 0.9420 - val_loss: 0.2461 - val_accuracy: 0.9280
Epoch 17/20
20821/20821 [==============================] - 452s 22ms/step - loss: 0.1629 - accuracy: 0.9414 - val_loss: 0.2448 - val_accuracy: 0.9290
Epoch 18/20
20821/20821 [==============================] - 453s 22ms/step - loss: 0.1615 - accuracy: 0.9420 - val_loss: 0.2483 - val_accuracy: 0.9272
Epoch 19/20
20821/20821 [==============================] - 457s 22ms/step - loss: 0.1599 - accuracy: 0.9427 - val_loss: 0.2478 - val_accuracy: 0.9271
Epoch 20/20
20821/20821 [==============================] - 453s 22ms/step - loss: 0.1606 - accuracy: 0.9418 - val_loss: 0.2491 - val_accuracy: 0.9262

Epoch 00020: ReduceLROnPlateau reducing learning rate to 4.0000001899898055e-05.
```

We notice slight improvement in accuracy compared to model1(neural network)

```
[ ] #Evaluate test set and then print accuracy
    test2 = model2.evaluate(X_test, y_test)
    print('Test accuracy: ', test2[1])
```

```
6941/6941 [==============================] - 32s 5ms/step
Test accuracy:  0.9262354373931885
```

```
[ ] #Evaluate train set and then print accuracy
    train2 = model2.evaluate(X_train, y_train)
    print('Train accuracy: ', train2[1])
```

```
20821/20821 [==============================] - 99s 5ms/step
Train accuracy:  0.943326473236084
```

Noticed slight improvement in accuracy and good precision scores too.

	precision	recall	f1-score	support
0	0.92	0.64	0.76	160
1	0.99	0.88	0.93	169
2	1.00	0.89	0.94	169
3	1.00	1.00	1.00	167
4	0.97	0.89	0.93	161
5	0.98	0.96	0.97	180
6	0.99	0.99	0.99	173
7	0.99	1.00	0.99	156
8	0.97	1.00	0.99	171
9	0.98	1.00	0.99	163
10	0.97	0.94	0.95	177
11	0.94	0.94	0.94	173
12	0.93	0.90	0.92	168
13	0.94	1.00	0.97	169
14	1.00	1.00	1.00	179
15	1.00	1.00	1.00	145
16	0.98	1.00	0.99	160
17	1.00	0.98	0.99	163
18	0.95	1.00	0.97	170
19	0.97	1.00	0.99	151
20	0.99	1.00	1.00	175
21	0.98	0.95	0.96	172
22	0.90	0.98	0.94	155
23	1.00	0.96	0.98	157
24	0.96	1.00	0.98	162
25	0.98	1.00	0.99	170
26	0.99	0.97	0.98	174
27	0.90	0.94	0.92	146
28	1.00	1.00	1.00	171
29	0.99	1.00	0.99	164
30	1.00	1.00	1.00	177
31	1.00	0.78	0.88	140
32	0.80	0.95	0.87	183
33	1.00	1.00	1.00	166
34	0.77	0.80	0.78	167
35	0.91	0.53	0.67	170
36	0.96	0.80	0.88	157
37	0.99	1.00	0.99	172
38	0.99	1.00	0.99	153
39	0.80	0.44	0.57	178
40	0.37	0.87	0.52	159
41	0.89	0.93	0.91	149
accuracy			0.93	6941
macro avg	0.94	0.93	0.93	6941
weighted avg	0.94	0.93	0.93	6941

LSTM WITH PRE-TRAINED GLOVE EMBEDDINGS.

Using the GloVe 6B pre trained weights (200D)

```
#model building

model3 = Sequential()  #Sequential model
model3.add(Embedding(max_features, 200, input_length=maxlen, weights=[embedding_matrix], trainable=True)) #Init embedding layer with no pretrained wts
model3.add(Dropout(0.5)) #use dropout for regularization embedding_matrix
model3.add(LSTM(100,return_sequences=True, dropout=0.5))  #Hidden layer with dropout Here return_sequences=true as I want to get output from all layer than apply global max pool over all the output so that I dont miss out important info
model3.add(GlobalMaxPool1D()) #This is to get values from all states and not missing important info
model3.add(Dense(42, activation="sigmoid")) #Final dense layer
#Note:GlobalMaxPool1D requires output from all lSTMs therfore return_sequences = True, this returns output from all LSTM not just final LSTM
#ALso GlobalMaxPool1D takes max of all output in very special way this step is done so that every words wether at start or at he end is given importance.
```

```
[ ] #Compile model with optimizer adam, loss as binary cross entropy and metric is accuracy
    model3.compile(
      loss='categorical_crossentropy',
      #optimizer=Adam(lr=0.01),  #best
      optimizer='adam',
      metrics=['acc']
    )
```

```
Train on 20821 samples, validate on 6941 samples
Epoch 1/25
20821/20821 [==============================] - 633s 30ms/step - loss: 3.1594 - acc: 0.2008 - val_loss: 2.4176 - val_acc: 0.3884
Epoch 2/25
20821/20821 [==============================] - 635s 30ms/step - loss: 2.1470 - acc: 0.4338 - val_loss: 1.6595 - val_acc: 0.5695
Epoch 3/25
20821/20821 [==============================] - 634s 30ms/step - loss: 1.5688 - acc: 0.5941 - val_loss: 1.1717 - val_acc: 0.7061
Epoch 4/25
20821/20821 [==============================] - 635s 31ms/step - loss: 1.1465 - acc: 0.7089 - val_loss: 0.8411 - val_acc: 0.7874
Epoch 5/25
20821/20821 [==============================] - 635s 30ms/step - loss: 0.8586 - acc: 0.7816 - val_loss: 0.6424 - val_acc: 0.8307
Epoch 6/25
20821/20821 [==============================] - 636s 31ms/step - loss: 0.6658 - acc: 0.8270 - val_loss: 0.5114 - val_acc: 0.8562
Epoch 7/25
20821/20821 [==============================] - 637s 31ms/step - loss: 0.5381 - acc: 0.8586 - val_loss: 0.4371 - val_acc: 0.8752
Epoch 8/25
20821/20821 [==============================] - 637s 31ms/step - loss: 0.4467 - acc: 0.8795 - val_loss: 0.3772 - val_acc: 0.8909
Epoch 9/25
20821/20821 [==============================] - 637s 31ms/step - loss: 0.3865 - acc: 0.8911 - val_loss: 0.3294 - val_acc: 0.9048
Epoch 10/25
20821/20821 [==============================] - 636s 31ms/step - loss: 0.3370 - acc: 0.9020 - val_loss: 0.3077 - val_acc: 0.9091
Epoch 11/25
20821/20821 [==============================] - 636s 31ms/step - loss: 0.3023 - acc: 0.9122 - val_loss: 0.2920 - val_acc: 0.9136
Epoch 12/25
20821/20821 [==============================] - 639s 31ms/step - loss: 0.2757 - acc: 0.9182 - val_loss: 0.2724 - val_acc: 0.9173
Epoch 13/25
20821/20821 [==============================] - 637s 31ms/step - loss: 0.2558 - acc: 0.9218 - val_loss: 0.2621 - val_acc: 0.9169
Epoch 14/25
20821/20821 [==============================] - 636s 31ms/step - loss: 0.2547 - acc: 0.9225 - val_loss: 0.2574 - val_acc: 0.9206
Epoch 15/25
20821/20821 [==============================] - 634s 30ms/step - loss: 0.2308 - acc: 0.9288 - val_loss: 0.2431 - val_acc: 0.9225
Epoch 16/25
20821/20821 [==============================] - 634s 30ms/step - loss: 0.2175 - acc: 0.9300 - val_loss: 0.2433 - val_acc: 0.9244
Epoch 17/25
20821/20821 [==============================] - 635s 31ms/step - loss: 0.2105 - acc: 0.9311 - val_loss: 0.2371 - val_acc: 0.9268
Epoch 18/25
20821/20821 [==============================] - 635s 31ms/step - loss: 0.1989 - acc: 0.9330 - val_loss: 0.2404 - val_acc: 0.9261
Epoch 19/25
20821/20821 [==============================] - 636s 31ms/step - loss: 0.1924 - acc: 0.9357 - val_loss: 0.2342 - val_acc: 0.9280
Epoch 20/25
20821/20821 [==============================] - 637s 31ms/step - loss: 0.1859 - acc: 0.9377 - val_loss: 0.2370 - val_acc: 0.9247
Epoch 21/25
20821/20821 [==============================] - 634s 30ms/step - loss: 0.1851 - acc: 0.9371 - val_loss: 0.2316 - val_acc: 0.9297
Epoch 22/25
20821/20821 [==============================] - 638s 31ms/step - loss: 0.1805 - acc: 0.9389 - val_loss: 0.2265 - val_acc: 0.9304
Epoch 23/25
20821/20821 [==============================] - 631s 30ms/step - loss: 0.1771 - acc: 0.9385 - val_loss: 0.2313 - val_acc: 0.9313
Epoch 24/25
20821/20821 [==============================] - 619s 30ms/step - loss: 0.1736 - acc: 0.9403 - val_loss: 0.2345 - val_acc: 0.9310
Epoch 25/25
20821/20821 [==============================] - 616s 30ms/step - loss: 0.1727 - acc: 0.9398 - val_loss: 0.2332 - val_acc: 0.9301

Epoch 00025: ReduceLROnPlateau reducing learning rate to 0.00020000000949949026.
```

```
[ ]  #Evaluate test set and then print accuracy
     test3 = model3.evaluate(X_test, y_test)
     print('Test accuracy: ', test3[1])
```

```
6941/6941 [==============================] - 39s 6ms/step
Test accuracy:  0.93012535572052
```

```
[ ]  #Evaluate train set and then print accuracy
     train3 = model3.evaluate(X_train, y_train)
     print('Train accuracy: ', train3[1])
```

```
20821/20821 [==============================] - 116s 6ms/step
Train accuracy:  0.9453436732292175
```

	precision	recall	f1-score	support
0	0.92	0.67	0.78	160
1	0.89	1.00	0.94	169
2	0.99	0.89	0.93	169
3	0.99	1.00	1.00	167
4	0.91	0.92	0.92	161
5	0.99	0.99	0.99	180
6	0.99	0.98	0.98	173
7	0.99	1.00	0.99	156
8	0.98	1.00	0.99	171
9	1.00	1.00	1.00	163
10	0.94	0.94	0.94	177
11	0.92	0.92	0.92	173
12	0.91	0.90	0.90	168
13	0.99	1.00	1.00	169
14	0.99	1.00	0.99	179
15	1.00	1.00	1.00	145
16	1.00	1.00	1.00	160
17	0.97	0.98	0.98	163
18	0.99	1.00	0.99	170
19	0.99	1.00	0.99	151
20	1.00	1.00	1.00	175
21	0.99	0.95	0.97	172
22	0.94	0.99	0.97	155
23	0.98	1.00	0.99	157
24	0.99	0.99	0.99	162
25	0.99	0.98	0.99	170
26	0.97	1.00	0.99	174
27	0.92	0.93	0.93	146
28	1.00	1.00	1.00	171
29	0.99	1.00	1.00	164
30	1.00	1.00	1.00	177
31	0.94	0.78	0.85	140
32	0.81	0.97	0.88	183
33	0.99	1.00	1.00	166
34	0.91	0.65	0.76	167
35	1.00	0.53	0.69	170
36	0.95	0.80	0.87	157
37	0.99	1.00	0.99	172
38	0.99	1.00	0.99	153
39	0.80	0.49	0.61	178
40	0.37	0.88	0.52	159
41	0.89	0.94	0.92	149
accuracy			0.93	6941
macro avg	0.95	0.93	0.93	6941
weighted avg	0.95	0.93	0.93	6941

COMPARISON ON THE THREE MODELS BUILT SO FAR

	Method	train acc	test acc
1	NN	0.938235	0.923066
2	LSTM	0.943326	0.926235
3	LSTM with weights	0.945344	0.930125

LSTM with pre-trained glove embeddings is performing well.

LSTM ITERATION 2- TRYING WITH DIFFERENT HIDDEN UNITS.

· **LSTM iteration 3- trying with different hidden units **

```
model7 = Sequential()  #Sequential model
model7.add(Embedding(max_features, 200, input_length=maxlen, weights=[embedding_matrix], trainable=True)) #Init e
model7.add(Dropout(0.5)) #use dropout for regularization embedding_matrix
model7.add(LSTM(128,return_sequences=True, dropout=0.4))  #Hidden layer with dropout Here return_sequences=true a
model7.add(GlobalMaxPool1D()) #This is to get values from all states and not missing important info
model7.add(Dense(42, activation="sigmoid")) #Final dense layer
#Note:GlobalMaxPool1D requires output from all lSTMs therfore return_sequences = True, this returns output from a
#ALso GlobalMaxPool1D takes max of all output in very special way this step is done so that every words wether at
#Compile model with optimizer adam, loss as binary cross entropy and metric is accuracy
model7.compile(
  loss='categorical_crossentropy',
  #optimizer=Adam(lr=0.01),  #best
  optimizer='adam',
  metrics=['acc']
)
#Define checkpoint, early stop and reduced lr
stop = EarlyStopping(monitor="val_loss", patience=5, mode="min")
reduce_lr = ReduceLROnPlateau(monitor="val_loss", factor=0.2, patience=3, min_lr=1e-6, verbose=1, mode="min")
#Fitting model to xtrain and ytrain with defined epochs and batch size
#Here I have used requlaization and model performance tech such as reduced lr, check point and early stop

history7 = model7.fit(
  X_train,
  y_train,
  batch_size=32,
  epochs=30,
  validation_data=(X_test,y_test),
  callbacks=[reduce_lr, stop],
  verbose=1
)
```

```
Train on 20821 samples, validate on 6941 samples
Epoch 1/30
20821/20821 [==============================] - 1244s 60ms/step - loss: 2.7629 - acc: 0.2755 - val_loss: 1.7569 - val_acc: 0.4988
Epoch 2/30
20821/20821 [==============================] - 1247s 60ms/step - loss: 1.5447 - acc: 0.5627 - val_loss: 1.0554 - val_acc: 0.7143
Epoch 3/30
20821/20821 [==============================] - 1241s 60ms/step - loss: 0.9832 - acc: 0.7302 - val_loss: 0.6941 - val_acc: 0.7927
Epoch 4/30
20821/20821 [==============================] - 1240s 60ms/step - loss: 0.6865 - acc: 0.8097 - val_loss: 0.5226 - val_acc: 0.8499
Epoch 5/30
20821/20821 [==============================] - 1242s 60ms/step - loss: 0.5079 - acc: 0.8581 - val_loss: 0.4007 - val_acc: 0.8793
Epoch 6/30
20821/20821 [==============================] - 1237s 59ms/step - loss: 0.4062 - acc: 0.8834 - val_loss: 0.3365 - val_acc: 0.8973
Epoch 7/30
20821/20821 [==============================] - 1236s 59ms/step - loss: 0.3318 - acc: 0.9028 - val_loss: 0.2948 - val_acc: 0.9062
Epoch 8/30
20821/20821 [==============================] - 1247s 60ms/step - loss: 0.2854 - acc: 0.9131 - val_loss: 0.2810 - val_acc: 0.9092
Epoch 9/30
20821/20821 [==============================] - 1243s 60ms/step - loss: 0.2556 - acc: 0.9215 - val_loss: 0.2598 - val_acc: 0.9150
Epoch 10/30
20821/20821 [==============================] - 1236s 59ms/step - loss: 0.2341 - acc: 0.9248 - val_loss: 0.2455 - val_acc: 0.9182
Epoch 11/30
20821/20821 [==============================] - 1241s 60ms/step - loss: 0.2175 - acc: 0.9300 - val_loss: 0.2463 - val_acc: 0.9225
Epoch 12/30
20821/20821 [==============================] - 1239s 60ms/step - loss: 0.2008 - acc: 0.9341 - val_loss: 0.2351 - val_acc: 0.9232
Epoch 13/30
20821/20821 [==============================] - 1236s 59ms/step - loss: 0.1902 - acc: 0.9353 - val_loss: 0.2323 - val_acc: 0.9242
Epoch 14/30
20821/20821 [==============================] - 1238s 59ms/step - loss: 0.1826 - acc: 0.9370 - val_loss: 0.2308 - val_acc: 0.9265
Epoch 15/30
20821/20821 [==============================] - 1249s 60ms/step - loss: 0.1761 - acc: 0.9404 - val_loss: 0.2317 - val_acc: 0.9242
Epoch 16/30
20821/20821 [==============================] - 1251s 60ms/step - loss: 0.1718 - acc: 0.9399 - val_loss: 0.2398 - val_acc: 0.9242
Epoch 17/30
20821/20821 [==============================] - 1251s 60ms/step - loss: 0.1665 - acc: 0.9404 - val_loss: 0.2433 - val_acc: 0.9242

Epoch 00017: ReduceLROnPlateau reducing learning rate to 0.00020000000949949026.
Epoch 18/30
20821/20821 [==============================] - 1248s 60ms/step - loss: 0.1593 - acc: 0.9439 - val_loss: 0.2335 - val_acc: 0.9288
Epoch 19/30
20821/20821 [==============================] - 1247s 60ms/step - loss: 0.1572 - acc: 0.9440 - val_loss: 0.2323 - val_acc: 0.9291
```

```
[ ]  #Evaluate test set and then print accuracy
     test7 = model7.evaluate(X_test, y_test)
     print('Test accuracy: ', test7[1])
```

```
6941/6941 [==============================] - 83s 12ms/step
Test accuracy:  0.9291168451309204
```

```
[ ]  #Evaluate train set and then print accuracy
     train7 = model7.evaluate(X_train, y_train)
     print('Train accuracy: ', train7[1])
```

```
20821/20821 [==============================] - 255s 12ms/step
Train accuracy:  0.9479371905326843
```

	precision	recall	f1-score	support
0	0.89	0.72	0.80	160
1	0.90	1.00	0.95	169
2	0.99	0.85	0.92	169
3	0.99	1.00	0.99	167
4	0.94	0.92	0.93	161
5	0.98	0.96	0.97	180
6	0.99	0.97	0.98	173
7	1.00	1.00	1.00	156
8	0.98	1.00	0.99	171
9	0.99	1.00	1.00	163
10	0.99	0.96	0.97	177
11	0.92	0.93	0.93	173
12	0.94	0.93	0.94	168
13	0.99	1.00	1.00	169
14	1.00	1.00	1.00	179
15	1.00	1.00	1.00	145
16	0.99	1.00	0.99	160
17	0.98	1.00	0.99	163
18	0.98	1.00	0.99	170
19	0.98	1.00	0.99	151
20	0.99	1.00	1.00	175
21	0.97	0.98	0.97	172
22	0.95	0.94	0.94	155
23	0.98	1.00	0.99	157
24	1.00	0.98	0.99	162
25	0.99	1.00	1.00	170
26	0.99	1.00	1.00	174
27	0.91	0.98	0.94	146
28	1.00	1.00	1.00	171
29	0.99	1.00	1.00	164
30	1.00	1.00	1.00	177
31	0.96	0.76	0.85	140
32	0.76	0.97	0.85	183
33	1.00	1.00	1.00	166
34	0.89	0.66	0.76	167
35	0.94	0.45	0.61	170
36	0.96	0.79	0.87	157
37	0.99	1.00	1.00	172
38	0.99	1.00	1.00	153
39	0.86	0.44	0.59	178
40	0.36	0.89	0.51	159
41	0.93	0.95	0.94	149
accuracy			0.93	6941
macro avg	0.95	0.93	0.93	6941
weighted avg	0.95	0.93	0.93	6941

Not much improvement seen. So, we will stick to the first model itself.

BI-DIRECTIONAL LSTM USING GLOVE EMBEDDINGS

```
[ ] #Model buliding
    model4 = Sequential()  #Sequential model
    model4.add(Embedding(max_features, 200, input_length = maxlen, weights = [embedding_matrix]))
    #embedding layer takes input of vocabulary size, embedding dimension i.e 200 and input sequence i.e number of columns of dataset i.e 254
    #Here I am using pretrained glove vector stored in variable
    model4.add(Dropout(0.5)) #use dropout for regularization embedding_matrix
    model4.add(Bidirectional(LSTM(100, dropout=0.5)))  #Hidden layer with dropout Here return_sequences=true as I want to get output from all layer
    model4.add(Dense(42, activation="sigmoid")) #Final dense layer
```

```
[ ] #print model summary
    model4.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, 769, 200)          2600000
_________________________________________________________________
dropout_3 (Dropout)          (None, 769, 200)          0
_________________________________________________________________
bidirectional_3 (Bidirection (None, 200)               240800
_________________________________________________________________
dense_3 (Dense)              (None, 42)                8442
=================================================================
Total params: 2,849,242
Trainable params: 2,849,242
Non-trainable params: 0
_________________________________________________________________
```

```
Train on 16656 samples, validate on 4165 samples
Epoch 1/20
16656/16656 [==============================] - 949s 57ms/step - loss: 3.1093 - acc: 0.1651 - val_loss: 2.2671 - val_acc: 0.3767
Epoch 2/20
16656/16656 [==============================] - 937s 56ms/step - loss: 2.0234 - acc: 0.4295 - val_loss: 1.4127 - val_acc: 0.6031
Epoch 3/20
16656/16656 [==============================] - 955s 57ms/step - loss: 1.3460 - acc: 0.6327 - val_loss: 0.9124 - val_acc: 0.7556
Epoch 4/20
16656/16656 [==============================] - 945s 57ms/step - loss: 0.8992 - acc: 0.7565 - val_loss: 0.6414 - val_acc: 0.8238
Epoch 5/20
16656/16656 [==============================] - 938s 56ms/step - loss: 0.6464 - acc: 0.8222 - val_loss: 0.4995 - val_acc: 0.8564
Epoch 6/20
16656/16656 [==============================] - 941s 57ms/step - loss: 0.5135 - acc: 0.8533 - val_loss: 0.4224 - val_acc: 0.8768
Epoch 7/20
16656/16656 [==============================] - 945s 57ms/step - loss: 0.4288 - acc: 0.8732 - val_loss: 0.3688 - val_acc: 0.8848
Epoch 8/20
16656/16656 [==============================] - 942s 57ms/step - loss: 0.3730 - acc: 0.8877 - val_loss: 0.3469 - val_acc: 0.8946
Epoch 9/20
16656/16656 [==============================] - 932s 56ms/step - loss: 0.3247 - acc: 0.9007 - val_loss: 0.3199 - val_acc: 0.8982
Epoch 10/20
16656/16656 [==============================] - 937s 56ms/step - loss: 0.2974 - acc: 0.9074 - val_loss: 0.3056 - val_acc: 0.9011
Epoch 11/20
16656/16656 [==============================] - 936s 56ms/step - loss: 0.2790 - acc: 0.9131 - val_loss: 0.2918 - val_acc: 0.9088
Epoch 12/20
16656/16656 [==============================] - 939s 56ms/step - loss: 0.2612 - acc: 0.9148 - val_loss: 0.2978 - val_acc: 0.9100
Epoch 13/20
16656/16656 [==============================] - 932s 56ms/step - loss: 0.2477 - acc: 0.9183 - val_loss: 0.2814 - val_acc: 0.9121
Epoch 14/20
16656/16656 [==============================] - 942s 57ms/step - loss: 0.2315 - acc: 0.9226 - val_loss: 0.2823 - val_acc: 0.9157
Epoch 15/20
16656/16656 [==============================] - 936s 56ms/step - loss: 0.2294 - acc: 0.9235 - val_loss: 0.2738 - val_acc: 0.9090
Epoch 16/20
16656/16656 [==============================] - 938s 56ms/step - loss: 0.2203 - acc: 0.9243 - val_loss: 0.2826 - val_acc: 0.9095
Epoch 17/20
16656/16656 [==============================] - 930s 56ms/step - loss: 0.2124 - acc: 0.9280 - val_loss: 0.2782 - val_acc: 0.9121

Epoch 00017: ReduceLROnPlateau reducing learning rate to 0.00020000000949949026.
Epoch 18/20
16656/16656 [==============================] - 928s 56ms/step - loss: 0.1968 - acc: 0.9325 - val_loss: 0.2748 - val_acc: 0.9196
```

```
[ ]  #Evaluate test set and then print accuracy
     test4 = model4.evaluate(X_test, y_test)
     print('Test accuracy: ', test4[1])
```

```
6941/6941 [==============================] - 75s 11ms/step
Test accuracy:  0.9157181978225708
```

```
[ ]  #Evaluate train set and then print accuracy
     train4 = model4.evaluate(X_train, y_train)
     print('Train accuracy: ', train4[1])
```

```
20821/20821 [==============================] - 224s 11ms/step
Train accuracy:  0.9372268319129944
```

	precision	recall	f1-score	support
0	0.83	0.61	0.71	160
1	0.90	1.00	0.95	169
2	0.98	0.86	0.91	169
3	1.00	1.00	1.00	167
4	0.87	0.94	0.90	161
5	0.98	0.92	0.95	180
6	0.98	0.97	0.97	173
7	0.98	1.00	0.99	156
8	0.97	1.00	0.98	171
9	1.00	1.00	1.00	163
10	0.98	0.96	0.97	177
11	0.90	0.89	0.90	173
12	0.90	0.82	0.86	168
13	0.99	1.00	0.99	169
14	0.99	1.00	1.00	179
15	0.99	1.00	0.99	145
16	0.99	1.00	1.00	160
17	0.98	0.98	0.98	163
18	0.92	1.00	0.96	170
19	0.94	1.00	0.97	151
20	0.99	1.00	1.00	175
21	0.99	0.98	0.99	172
22	0.92	0.90	0.91	155
23	0.98	0.98	0.98	157
24	0.98	1.00	0.99	162
25	0.98	0.99	0.99	170
26	0.98	1.00	0.99	174
27	0.81	0.99	0.89	146
28	0.99	1.00	1.00	171
29	0.98	1.00	0.99	164
30	0.99	1.00	0.99	177
31	0.95	0.82	0.88	140
32	0.79	0.94	0.86	183
33	1.00	1.00	1.00	166
34	0.89	0.66	0.76	167
35	0.89	0.45	0.60	170
36	0.97	0.80	0.87	157
37	0.99	1.00	1.00	172
38	0.96	1.00	0.98	153
39	0.90	0.34	0.50	178
40	0.34	0.87	0.49	159
41	0.89	0.79	0.84	149
accuracy			0.92	6941
macro avg	0.93	0.92	0.92	6941
weighted avg	0.94	0.92	0.92	6941

ITERATION 2

Checking with batch size of 32 and adding a global max pooling layer.

```
#CHECKING WITH BATCH SIZE OF 32 AND ADDING A GLOBALMAXPOOOLNG LAYER
#Model buliding
model_bi = Sequential()  #Sequential model
model_bi.add(Embedding(max_features, 200, input_length = maxlen, weights = [embedding_matrix]))
#embedding layer takes input of vocabulary size, embedding dimension i.e 200 and input sequence i.e number of columns of dataset i.e
#Here I am using pretrained glove vector stored in variable
model_bi.add(Dropout(0.5)) #use dropout for regularization embedding_matrix
model_bi.add(Bidirectional(LSTM(100,return_sequences=True, dropout=0.4)))  #Hidden layer with dropout Here return_sequences=true as
model_bi.add(GlobalMaxPool1D())
model_bi.add(Dense(42, activation="sigmoid")) #Final dense layer
```

```
Train on 20821 samples, validate on 6941 samples
Epoch 1/25
20821/20821 [==============================] - 1633s 78ms/step - loss: 2.4976 - acc: 0.3479 - val_loss: 1.4333 - val_acc: 0.6039
Epoch 2/25
20821/20821 [==============================] - 1640s 79ms/step - loss: 1.1587 - acc: 0.6943 - val_loss: 0.7185 - val_acc: 0.8006
Epoch 3/25
20821/20821 [==============================] - 1651s 79ms/step - loss: 0.6771 - acc: 0.8196 - val_loss: 0.4904 - val_acc: 0.8617
Epoch 4/25
20821/20821 [==============================] - 1636s 79ms/step - loss: 0.4713 - acc: 0.8687 - val_loss: 0.4030 - val_acc: 0.8698
Epoch 5/25
20821/20821 [==============================] - 1630s 78ms/step - loss: 0.3815 - acc: 0.8897 - val_loss: 0.3276 - val_acc: 0.8987
Epoch 6/25
20821/20821 [==============================] - 1647s 79ms/step - loss: 0.3107 - acc: 0.9062 - val_loss: 0.3012 - val_acc: 0.9032
Epoch 7/25
20821/20821 [==============================] - 1667s 80ms/step - loss: 0.2690 - acc: 0.9167 - val_loss: 0.2748 - val_acc: 0.9137
Epoch 8/25
20821/20821 [==============================] - 1630s 78ms/step - loss: 0.2400 - acc: 0.9234 - val_loss: 0.2575 - val_acc: 0.9195
Epoch 9/25
20821/20821 [==============================] - 1550s 74ms/step - loss: 0.2195 - acc: 0.9296 - val_loss: 0.2583 - val_acc: 0.9173
Epoch 10/25
20821/20821 [==============================] - 1559s 75ms/step - loss: 0.2041 - acc: 0.9338 - val_loss: 0.2497 - val_acc: 0.9210
Epoch 11/25
20821/20821 [==============================] - 1552s 75ms/step - loss: 0.1987 - acc: 0.9346 - val_loss: 0.2677 - val_acc: 0.9205
Epoch 12/25
20821/20821 [==============================] - 1545s 74ms/step - loss: 0.1941 - acc: 0.9342 - val_loss: 0.2473 - val_acc: 0.9218
Epoch 13/25
20821/20821 [==============================] - 1544s 74ms/step - loss: 0.1785 - acc: 0.9397 - val_loss: 0.2432 - val_acc: 0.9241
Epoch 14/25
20821/20821 [==============================] - 1549s 74ms/step - loss: 0.1760 - acc: 0.9389 - val_loss: 0.2393 - val_acc: 0.9272
Epoch 15/25
20821/20821 [==============================] - 1544s 74ms/step - loss: 0.1844 - acc: 0.9390 - val_loss: 0.2373 - val_acc: 0.9255
Epoch 16/25
20821/20821 [==============================] - 1550s 74ms/step - loss: 0.1813 - acc: 0.9381 - val_loss: 0.2371 - val_acc: 0.9251
Epoch 17/25
20821/20821 [==============================] - 1562s 75ms/step - loss: 0.1714 - acc: 0.9395 - val_loss: 0.2321 - val_acc: 0.9249
Epoch 18/25
20821/20821 [==============================] - 1564s 75ms/step - loss: 0.1633 - acc: 0.9427 - val_loss: 0.2411 - val_acc: 0.9205
Epoch 19/25
20821/20821 [==============================] - 1574s 76ms/step - loss: 0.1580 - acc: 0.9434 - val_loss: 0.2390 - val_acc: 0.9288
Epoch 20/25
20821/20821 [==============================] - 1541s 74ms/step - loss: 0.1560 - acc: 0.9448 - val_loss: 0.2489 - val_acc: 0.9265

Epoch 00020: ReduceLROnPlateau reducing learning rate to 0.00020000000949949026.
Epoch 21/25
20821/20821 [==============================] - 1551s 75ms/step - loss: 0.1503 - acc: 0.9469 - val_loss: 0.2417 - val_acc: 0.9290
Epoch 22/25
20821/20821 [==============================] - 1544s 74ms/step - loss: 0.1482 - acc: 0.9458 - val_loss: 0.2434 - val_acc: 0.9303
```

```
#Evaluate test set and then print accuracy
test_bi = model_bi.evaluate(X_test, y_test)
print('Test accuracy: ', test_bi[1])
```

```
6941/6941 [==============================] - 83s 12ms/step
Test accuracy:  0.9302694201469421
```

```
[ ] #Evaluate train set and then print accuracy
train_bi = model_bi.evaluate(X_train, y_train)
print('Train accuracy: ', train_bi[1])
```

```
20821/20821 [==============================] - 246s 12ms/step
Train accuracy:  0.9488016963005066
```

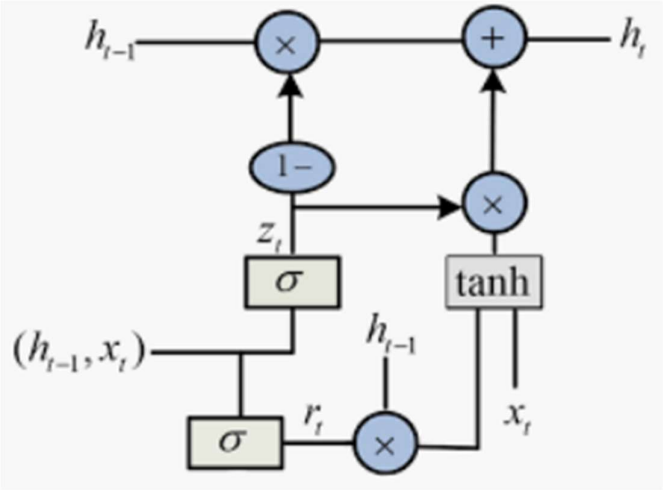
	precision	recall	f1-score	support
0	0.91	0.72	0.81	160
1	0.90	1.00	0.95	169
2	0.97	0.84	0.90	169
3	1.00	1.00	1.00	167
4	0.94	0.92	0.93	161
5	0.94	0.98	0.96	180
6	0.98	0.99	0.99	173
7	1.00	1.00	1.00	156
8	0.99	1.00	1.00	171
9	1.00	1.00	1.00	163
10	0.97	0.97	0.97	177
11	0.94	0.95	0.95	173
12	0.93	0.89	0.91	168
13	1.00	1.00	1.00	169
14	0.99	1.00	1.00	179
15	0.99	1.00	1.00	145
16	0.99	1.00	1.00	160
17	0.99	0.98	0.98	163
18	0.97	0.99	0.98	170
19	0.97	1.00	0.99	151
20	1.00	1.00	1.00	175
21	0.99	0.99	0.99	172
22	0.92	0.94	0.93	155
23	0.99	1.00	0.99	157
24	1.00	0.98	0.99	162
25	0.98	1.00	0.99	170
26	1.00	1.00	1.00	174
27	0.89	1.00	0.94	146
28	1.00	1.00	1.00	171
29	1.00	1.00	1.00	164
30	0.99	1.00	1.00	177
31	0.98	0.84	0.90	140
32	0.77	0.95	0.85	183
33	1.00	1.00	1.00	166
34	0.87	0.71	0.78	167
35	0.99	0.51	0.67	170
36	0.95	0.77	0.85	157
37	0.99	1.00	1.00	172
38	0.99	1.00	1.00	153
39	0.96	0.41	0.57	178
40	0.36	0.85	0.51	159
41	0.83	0.91	0.87	149
accuracy			0.93	6941
macro avg	0.95	0.93	0.93	6941
weighted avg	0.95	0.93	0.93	6941

GRU - BIDIRECTIONAL WITHOUT PRETRAINED EMBEDDINGS

Gated Recurrent Unit - To solve the vanishing gradient problem of a standard RNN, GRU uses, so-called, update gate and reset gate. Basically, these are two vectors which decide what information should be passed to the output. The special thing about them is that they can be trained to keep information from long ago, without washing it through time or remove information which is irrelevant to the prediction.

Why GRU over LSTM? The GRU controls the flow of information like the LSTM unit, but without having to use a memory unit. It just exposes the full hidden content without any control.

GRU is relatively new, and from my perspective, the performance is on par with LSTM, but computationally more efficient.



ADVANTAGES:

1. From my experience, GRUs train faster and perform better than LSTMs on less training data if you are doing language modelling (not sure about other tasks).
2. GRUs are simpler and thus easier to modify, for example adding new gates in case of additional input to the network. It's just less code in general.
3. LSTMs should in theory remember longer sequences than GRUs and outperform them in tasks requiring modelling long-distance relations.

```
[ ]  #USING GRU MODEL WITH RMSPROP AS OPTIMIZER
     model5 = Sequential()
     model5.add(Embedding(max_features, 200,input_length=maxlen))
     model5.add(Bidirectional(GRU(128, return_sequences = True)))
     model5.add(GlobalMaxPool1D())
     model5.add(Dropout(0.2))
     model5.add(Dense(42, activation="softmax"))
     model5.compile(loss='categorical_crossentropy', optimizer='rmsprop', metrics=['accuracy'])
     model5.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 769, 200)          2600000
_________________________________________________________________
bidirectional_1 (Bidirection (None, 769, 256)          252672
_________________________________________________________________
global_max_pooling1d_2 (Glob (None, 256)               0
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 42)                10794
=================================================================
Total params: 2,863,466
Trainable params: 2,863,466
Non-trainable params: 0
_________________________________________________________________
```

```
Train on 16656 samples, validate on 4165 samples
Epoch 1/15
16656/16656 [==============================] - 819s 49ms/step - loss: 2.7854 - accuracy: 0.3421 - val_loss: 1.7152 - val_accuracy: 0.6139
Epoch 2/15
16656/16656 [==============================] - 815s 49ms/step - loss: 1.1605 - accuracy: 0.7147 - val_loss: 0.8000 - val_accuracy: 0.7899
Epoch 3/15
16656/16656 [==============================] - 813s 49ms/step - loss: 0.5626 - accuracy: 0.8482 - val_loss: 0.5061 - val_accuracy: 0.8519
Epoch 4/15
16656/16656 [==============================] - 832s 50ms/step - loss: 0.3721 - accuracy: 0.8922 - val_loss: 0.4037 - val_accuracy: 0.8852
Epoch 5/15
16656/16656 [==============================] - 817s 49ms/step - loss: 0.2852 - accuracy: 0.9134 - val_loss: 0.3720 - val_accuracy: 0.8929
Epoch 6/15
16656/16656 [==============================] - 821s 49ms/step - loss: 0.2416 - accuracy: 0.9243 - val_loss: 0.3587 - val_accuracy: 0.8994
Epoch 7/15
16656/16656 [==============================] - 814s 49ms/step - loss: 0.2140 - accuracy: 0.9322 - val_loss: 0.3401 - val_accuracy: 0.9001
Epoch 8/15
16656/16656 [==============================] - 821s 49ms/step - loss: 0.2003 - accuracy: 0.9341 - val_loss: 0.3512 - val_accuracy: 0.9054
Epoch 9/15
16656/16656 [==============================] - 816s 49ms/step - loss: 0.1857 - accuracy: 0.9376 - val_loss: 0.3251 - val_accuracy: 0.9008
Epoch 10/15
16656/16656 [==============================] - 816s 49ms/step - loss: 0.1765 - accuracy: 0.9401 - val_loss: 0.3458 - val_accuracy: 0.9131
Epoch 11/15
16656/16656 [==============================] - 817s 49ms/step - loss: 0.1728 - accuracy: 0.9392 - val_loss: 0.3464 - val_accuracy: 0.9080
Epoch 12/15
16656/16656 [==============================] - 817s 49ms/step - loss: 0.1692 - accuracy: 0.9425 - val_loss: 0.3419 - val_accuracy: 0.9102
Epoch 13/15
16656/16656 [==============================] - 813s 49ms/step - loss: 0.1639 - accuracy: 0.9419 - val_loss: 0.3557 - val_accuracy: 0.9107
Epoch 14/15
16656/16656 [==============================] - 812s 49ms/step - loss: 0.1607 - accuracy: 0.9451 - val_loss: 0.3492 - val_accuracy: 0.9100
Epoch 15/15
16656/16656 [==============================] - 814s 49ms/step - loss: 0.1606 - accuracy: 0.9455 - val_loss: 0.3498 - val_accuracy: 0.9126
```

```
[ ]  #Evaluate test set and then print accuracy
     results5 = model5.evaluate(X_test, y_test)
     print('Test accuracy: ', results5[1])
```

```
6941/6941 [==============================] - 70s 10ms/step
Test accuracy:  0.9194640517234802
```

```
[ ]  #Evaluate train set and then print accuracy
     results5 = model5.evaluate(X_train, y_train)
     print('Train accuracy: ', results5[1])
```

```
20821/20821 [==============================] - 210s 10ms/step
Train accuracy:  0.9389078617095947
```

	precision	recall	f1-score	support
0	0.80	0.66	0.72	160
1	0.92	1.00	0.96	169
2	0.98	0.87	0.92	169
3	1.00	1.00	1.00	167
4	0.94	0.83	0.88	161
5	0.99	0.95	0.97	180
6	0.99	0.95	0.97	173
7	1.00	1.00	1.00	156
8	0.99	1.00	0.99	171
9	1.00	0.98	0.99	163
10	0.98	0.97	0.97	177
11	0.88	0.84	0.86	173
12	0.89	0.89	0.89	168
13	0.99	1.00	0.99	169
14	0.99	1.00	1.00	179
15	1.00	1.00	1.00	145
16	1.00	1.00	1.00	160
17	0.98	0.99	0.98	163
18	0.97	1.00	0.99	170
19	0.97	1.00	0.98	151
20	1.00	1.00	1.00	175
21	0.98	0.97	0.97	172
22	0.93	0.91	0.92	155
23	0.99	1.00	0.99	157
24	0.99	0.99	0.99	162
25	0.99	1.00	0.99	170
26	0.99	1.00	1.00	174
27	0.91	0.99	0.95	146
28	1.00	1.00	1.00	171
29	0.99	1.00	0.99	164
30	0.99	1.00	1.00	177
31	0.97	0.80	0.88	140
32	0.92	0.79	0.85	183
33	1.00	1.00	1.00	166
34	0.90	0.66	0.76	167
35	0.97	0.53	0.68	170
36	0.96	0.74	0.83	157
37	0.99	1.00	0.99	172
38	1.00	1.00	1.00	153
39	0.88	0.47	0.62	178
40	0.33	0.99	0.50	159
41	0.76	0.89	0.82	149
accuracy			0.92	6941
macro avg	0.95	0.92	0.92	6941
weighted avg	0.95	0.92	0.92	6941

GRU WITH PRE-TRAINED GLOVE EMBEDDINGS.

```
#model building

model6 = Sequential()  #Sequential model
model6.add(Embedding(max_features, 200, input_length=maxlen, weights=[embedding_matrix], trainable=True)) #I
model6.add(Dropout(0.5)) #use dropout for regularization embedding_matrix
model6.add(GRU(100,return_sequences=True, dropout=0.4))  #Hidden layer with dropout Here return_sequences=tr
model6.add(GlobalMaxPool1D()) #This is to get values from all states and not missing important info
model6.add(Dense(42, activation="sigmoid")) #Final dense layer
#Note:GlobalMaxPool1D requires output from all lSTMs therfore return_sequences = True, this returns output f
#ALso GlobalMaxPool1D takes max of all output in very special way this step is done so that every words weth
```

```
Train on 20821 samples, validate on 6941 samples
Epoch 1/25
20821/20821 [==============================] - 608s 29ms/step - loss: 3.4013 - acc: 0.1534 - val_loss: 2.8763 - val_acc: 0.2989
Epoch 2/25
20821/20821 [==============================] - 619s 30ms/step - loss: 2.4728 - acc: 0.3650 - val_loss: 1.9701 - val_acc: 0.4936
Epoch 3/25
20821/20821 [==============================] - 622s 30ms/step - loss: 1.7991 - acc: 0.5182 - val_loss: 1.4057 - val_acc: 0.6335
Epoch 4/25
20821/20821 [==============================] - 629s 30ms/step - loss: 1.3324 - acc: 0.6579 - val_loss: 1.0044 - val_acc: 0.7521
Epoch 5/25
20821/20821 [==============================] - 638s 31ms/step - loss: 0.9836 - acc: 0.7500 - val_loss: 0.7691 - val_acc: 0.8035
Epoch 6/25
20821/20821 [==============================] - 634s 30ms/step - loss: 0.7515 - acc: 0.8041 - val_loss: 0.5933 - val_acc: 0.8519
Epoch 7/25
20821/20821 [==============================] - 639s 31ms/step - loss: 0.5963 - acc: 0.8473 - val_loss: 0.4860 - val_acc: 0.8744
Epoch 8/25
20821/20821 [==============================] - 631s 30ms/step - loss: 0.4913 - acc: 0.8708 - val_loss: 0.4181 - val_acc: 0.8927
Epoch 9/25
20821/20821 [==============================] - 634s 30ms/step - loss: 0.4171 - acc: 0.8865 - val_loss: 0.3600 - val_acc: 0.9048
Epoch 10/25
20821/20821 [==============================] - 632s 30ms/step - loss: 0.3567 - acc: 0.9030 - val_loss: 0.3252 - val_acc: 0.9101
Epoch 11/25
20821/20821 [==============================] - 638s 31ms/step - loss: 0.3184 - acc: 0.9104 - val_loss: 0.3006 - val_acc: 0.9186
Epoch 12/25
20821/20821 [==============================] - 633s 30ms/step - loss: 0.2836 - acc: 0.9184 - val_loss: 0.2779 - val_acc: 0.9213
Epoch 13/25
20821/20821 [==============================] - 633s 30ms/step - loss: 0.2619 - acc: 0.9233 - val_loss: 0.2627 - val_acc: 0.9261
Epoch 14/25
20821/20821 [==============================] - 633s 30ms/step - loss: 0.2415 - acc: 0.9275 - val_loss: 0.2622 - val_acc: 0.9235
Epoch 15/25
20821/20821 [==============================] - 633s 30ms/step - loss: 0.2219 - acc: 0.9310 - val_loss: 0.2556 - val_acc: 0.9258
Epoch 16/25
20821/20821 [==============================] - 631s 30ms/step - loss: 0.2105 - acc: 0.9351 - val_loss: 0.2480 - val_acc: 0.9290
Epoch 17/25
20821/20821 [==============================] - 628s 30ms/step - loss: 0.2040 - acc: 0.9353 - val_loss: 0.2460 - val_acc: 0.9283
Epoch 18/25
20821/20821 [==============================] - 627s 30ms/step - loss: 0.1970 - acc: 0.9384 - val_loss: 0.2429 - val_acc: 0.9301
Epoch 19/25
20821/20821 [==============================] - 627s 30ms/step - loss: 0.1877 - acc: 0.9396 - val_loss: 0.2397 - val_acc: 0.9320
Epoch 20/25
20821/20821 [==============================] - 630s 30ms/step - loss: 0.1842 - acc: 0.9398 - val_loss: 0.2385 - val_acc: 0.9314
Epoch 21/25
20821/20821 [==============================] - 627s 30ms/step - loss: 0.1802 - acc: 0.9404 - val_loss: 0.2339 - val_acc: 0.9317
Epoch 22/25
20821/20821 [==============================] - 629s 30ms/step - loss: 0.1770 - acc: 0.9407 - val_loss: 0.2358 - val_acc: 0.9320
Epoch 23/25
20821/20821 [==============================] - 627s 30ms/step - loss: 0.1743 - acc: 0.9414 - val_loss: 0.2425 - val_acc: 0.9306
Epoch 24/25
20821/20821 [==============================] - 628s 30ms/step - loss: 0.1695 - acc: 0.9424 - val_loss: 0.2379 - val_acc: 0.9324

Epoch 00024: ReduceLROnPlateau reducing learning rate to 0.00020000000949949026.
Epoch 25/25
20821/20821 [==============================] - 628s 30ms/step - loss: 0.1625 - acc: 0.9436 - val_loss: 0.2343 - val_acc: 0.9336
```

```
[ ]  #Evaluate test set and then print accuracy
     test6 = model6.evaluate(X_test, y_test)
     print('Test accuracy: ', test6[1])
```

```
6941/6941 [==============================] - 41s 6ms/step
Test accuracy:  0.9335830807685852
```

```
[ ]  #Evaluate train set and then print accuracy
     train6 = model6.evaluate(X_train, y_train)
     print('Train accuracy: ', train6[1])
```

```
20821/20821 [==============================] - 122s 6ms/step
Train accuracy:  0.9480812549591064
```

	precision	recall	f1-score	support
0	0.88	0.71	0.79	160
1	0.96	1.00	0.98	169
2	1.00	0.85	0.92	169
3	1.00	1.00	1.00	167
4	0.92	0.89	0.91	161
5	0.98	0.94	0.96	180
6	0.99	0.98	0.99	173
7	0.99	1.00	1.00	156
8	0.98	1.00	0.99	171
9	0.99	1.00	1.00	163
10	0.97	0.95	0.96	177
11	0.96	0.92	0.94	173
12	0.95	0.92	0.93	168
13	1.00	1.00	1.00	169
14	0.98	1.00	0.99	179
15	0.99	1.00	1.00	145
16	0.99	1.00	1.00	160
17	1.00	0.99	1.00	163
18	0.98	1.00	0.99	170
19	0.98	1.00	0.99	151
20	0.99	1.00	0.99	175
21	0.99	0.98	0.99	172
22	0.94	0.97	0.95	155
23	0.98	1.00	0.99	157
24	0.99	0.98	0.99	162
25	0.99	0.98	0.99	170
26	0.96	1.00	0.98	174
27	0.93	1.00	0.96	146
28	0.99	1.00	0.99	171
29	0.99	1.00	1.00	164
30	1.00	1.00	1.00	177
31	0.98	0.84	0.90	140
32	0.77	0.96	0.86	183
33	1.00	1.00	1.00	166
34	0.90	0.78	0.84	167
35	0.96	0.52	0.68	170
36	0.97	0.80	0.87	157
37	0.99	1.00	0.99	172
38	0.96	1.00	0.98	153
39	0.94	0.47	0.63	178
40	0.36	0.87	0.51	159
41	0.91	0.91	0.91	149
accuracy			0.93	6941
macro avg	0.95	0.93	0.94	6941
weighted avg	0.95	0.93	0.94	6941

FINAL COMPARISON

MODEL	TRAIN ACCURACY	TEST ACCURACY
NN	0.938235	0.923066
LSTM	0.943326	0.926235
LSTM with pre weights	0.945344	0.930125
Bi-LSTM with weights	0.937227	0.915718
Bi-LSTM iter 2	0.9488	0.9302
Bi-GRU	0.93890	0.91946
GRU with pre weights	0.948081	0.933583

Three models are performing really well

- **LSTM WITH PRE-TRAINED GLOVE EMBEDDINGS**
- **BI-LSTM ITERATION 2**
- **GRU WITH PRE-TRAINED GLOVE EMBEDDINGS**

- The accuracy of each variants of LSTM model is as follows in the table. This clearly indicates how LSTM, in the family of RNN is efficient in dealing with textual data.
- We have been able to bump the performance to 93% on test and 94% on train.
- Making the dataset balanced, helped the model to be trained more accurately.
- Adding different layers and using pre-trained GloVe embeddings resulted in finding the model with more accuracy without overfitting, which is evident from the train vs test accuracy curve.

5.COMPARISON TO BENCHMARK

MILESTONE 1-BENCHMARK

Performance of ML models with HyperParameters:

	Model	Test acc	Train acc
0	kNN_hyperTuned	65.941176	94.411765
1	SVM with Linear kernel	68.411765	93.808824
2	SVM with RBF kernel	61.529412	83.455882
3	LogisticRegression_hyperTuned	66.529412	88.382353

Performance of ML models without HyperParameters:

	Model	Test acc	Train acc
0	Naive bayes	53.058824	56.838235
1	kNN	64.411765	72.779412
2	SVM()	61.529412	83.455882
3	decision tree classifier	59.235294	95.514706
4	RandomForestClassifier	63.705882	95.514706
5	LogisticRegression	62.000000	69.397059

Performance of ML models On Downsampled trained data:

	Model	Test acc	Train acc
0	LogisticRegression()	0.219133	0.245137
1	KNN()	0.856649	0.911724
2	Random Forest	0.918744	0.950531

We notice that hyper parameter tuning helped to increase the accuracy marginally. Created a balanced dataset has given very good accuracies with slight overfitting.

Deep learning model on raw data(pre-processed)

MODEL	TRAIN ACCURACY	TEST ACCURACY
NN	0.9276	0.6272
LSTM	0.6240	0.5651
LSTM with pretrained weights	0.6240	0.6023
BI-LSTM with pretrained weights	0.7105	0.5868

DEEP LEARNING MODEL ON UPSAMPLED DATA

Method	train acc	test acc
Simple NN	0.02381	0.447647
LSTM	0.02381	0.005882
LSTM with pretrained weights	0.43251	0.4735

Very poor accuracies.

SUMMARY

- Out of all the models we tried the accuracy of each model is as follows as given in the table above.
- Used TF_IDF features to build machine learning models which helped.
- Neural networks need to be fine-tuned to increase accuracies.

MILESTONE 2- BENCHMARK

Deep learning model on down sampled data

MODEL	TRAIN ACCURACY	TEST ACCURACY
NN	0.938235	0.923066
LSTM	0.943326	0.926235
LSTM with pre weights	0.945344	0.930125
Bi-LSTM with weights	0.937227	0.915718
Bi-LSTM iter 2	0.9488	0.9302
Bi-GRU	0.93890	0.91946
GRU with pre weights	0.948081	0.933583

SUMMARY

- Balancing the dataset by down sampling increased the accuracy to a greater extent without overfitting the model.
- Using transfer learning and playing around with different layers of the models helped to increase the accuracy

We improved the benchmark from 62% to 93 %.

So our objective here is to build AI based classifier model to assign the tickets to right functional groups by assigning given description with an accuracy of at least 90%.

From the prediction results we see that LSTM, BI-LSTM and GRU on resampled data (down sampling)is able to achieve an accuracy of about 93% which is above our benchmark.

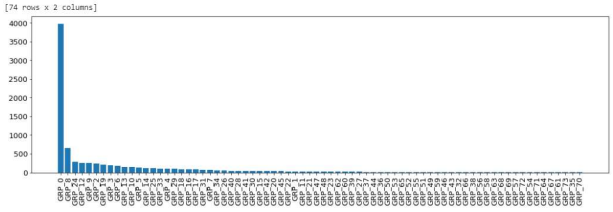
6. VISUALISATIONS.

ASSIGNMENT GROUPS

Visualization of ticket Distribution

Assignment group	Assignment group	cumpercentage
GRP_0	3976	46.776471
GRP_8	661	54.552941
GRP_24	289	57.952941
GRP_12	257	60.976471
GRP_9	252	63.941176
...	...	...
GRP_67	1	99.952941
GRP_61	1	99.964706
GRP_73	1	99.976471
GRP_35	1	99.988235
GRP_70	1	100.000000

[74 rows x 2 columns]



From the above statistics we can observe that the dataset is not normally distributed.

46% of the tickets are assigned to **GROUP_0**.

Hence we can expect better prediction for Group_0 incident tickets when compared with other groups in the dataset*

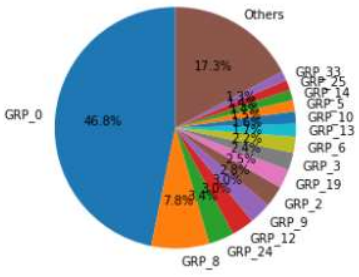
Pie chart visualization for group counts.

- Here we are creating a pie chart for groups have more than 100 tickets.

Pie Chart Visualization for group counts

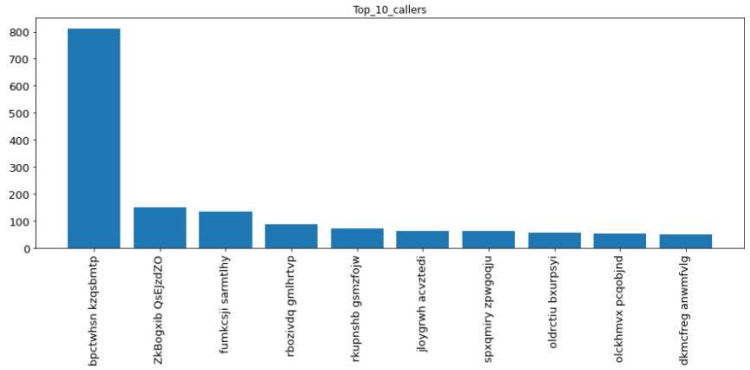
```
[ ]  #selecting groups having more count more than 100
     Pie_chart_data=group_freq["Assignment group"][group_freq["Assignment group"] >100]
     Pie_chart_data['Others']=(8500 - Pie_chart_data.values.sum(axis=0))


     plt.figure(figsize=(20,12))
     labels=list(Pie_chart_data.index)
     sizes=Pie_chart_data.values/8500
     fig1, ax1 = plt.subplots()
     ax1.pie(sizes,labels=labels, shadow=False, startangle=90,autopct='%1.1f%%')
     ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
     plt.show()
```

```
<Figure size 1440x864 with 0 Axes>
```

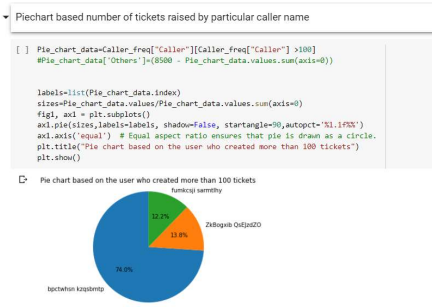


CALLERS

Visualizing top 10 callers

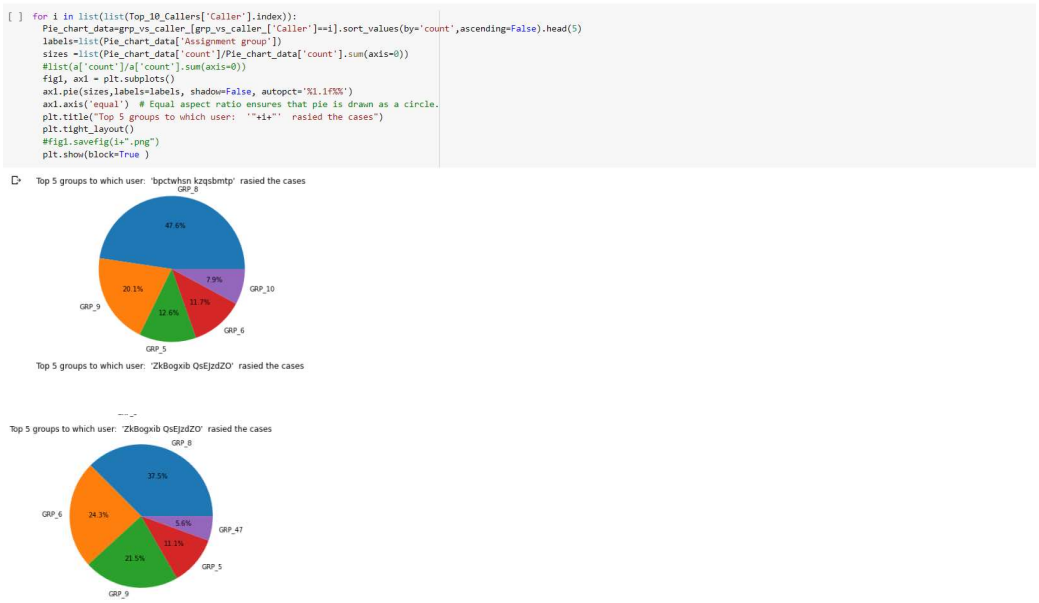


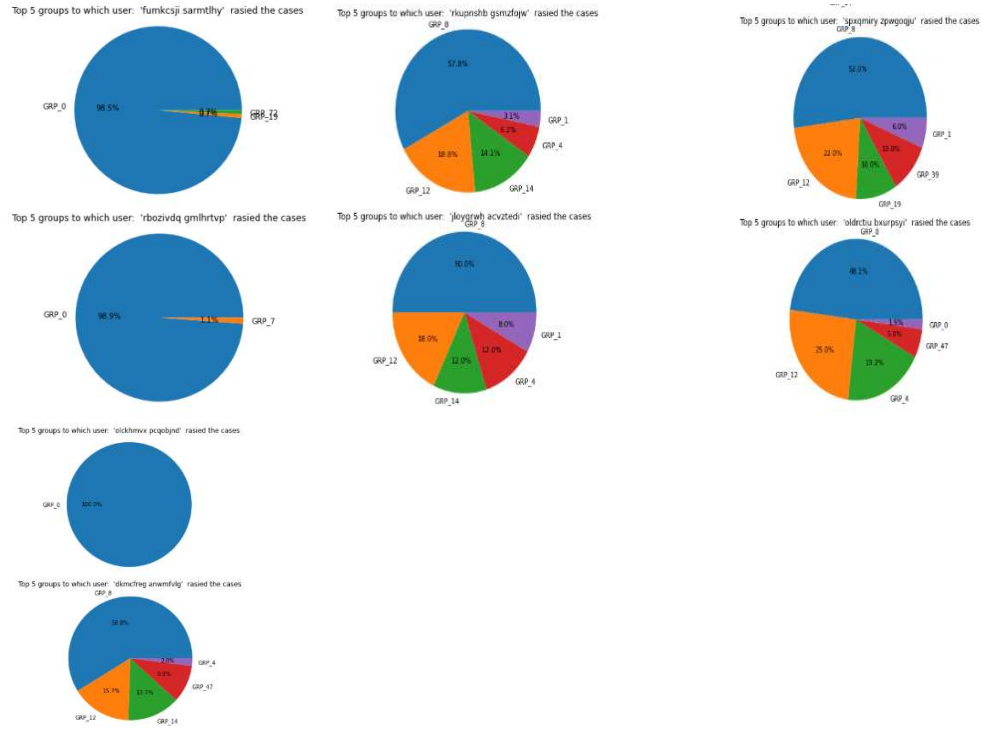
- For users who have raised more than 100 tickets.



We see only 3 users who have raised more than 100 tickets.

- Let us now visualize the top 5 groups to which our top 10 users have raised tickets to.





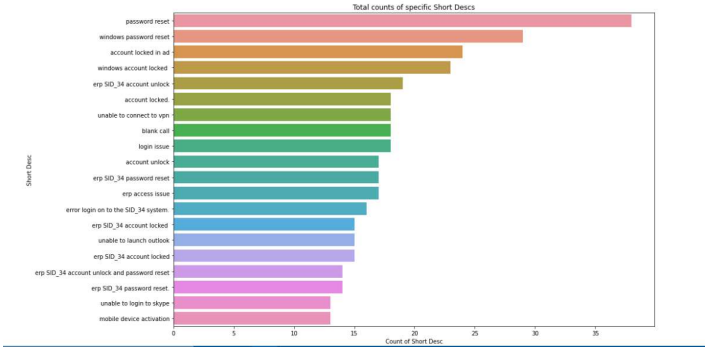
'om the above statistics we can observe that Caller bpotwhsn kzqsbmtp has raised majority of the tickets and out of 2950 unique callers.

SHORT DESCRIPTION

Visuals on short Description.

```
[ ] #Visualizing the distrinution of the column'Short Description'
    import seaborn as sns
    fig, (ax1) = plt.subplots(1,1, figsize=(15,10))

    x = df['Short description'].value_counts()[:20]
    sns.barplot(x, x.index, ax=ax1)
    plt.title('Total counts of specific Short Descs')
    plt.gca().set_xlabel('Count of Short Desc')
    plt.gca().set_ylabel('Short Desc')

    plt.show()
```

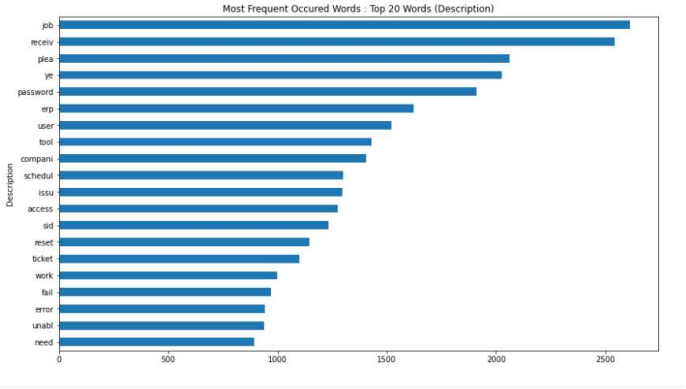
```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning: pandas.util.testing is deprecated. Use the functions in the public
  import pandas.util.testing as tm
```



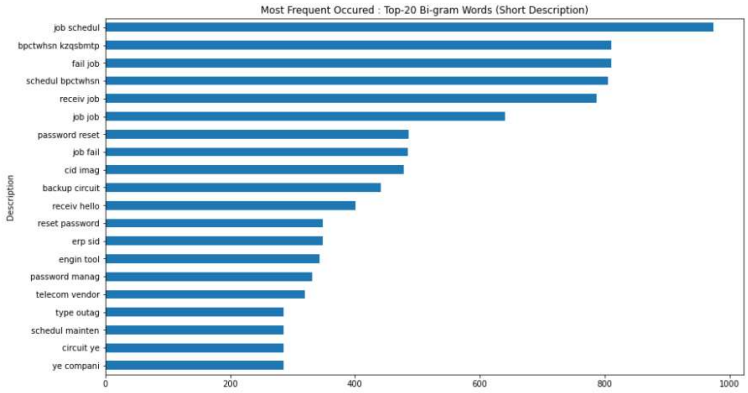
DESCRIPTION

Visualizing top 20 most frequently occurred unigrams and bi-grams.

Uni-gram words.



Bi-gram words



VISUALIZING UNI-GRAMS OF TOP 4 GROUPS USING WORD CLOUD

- Using Word Cloud library for visualization.

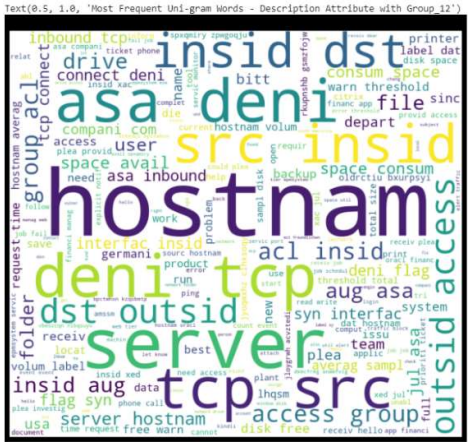
```
! pip install Wordcloud
from wordcloud import WordCloud
```

GROUP 0

GROUP 8

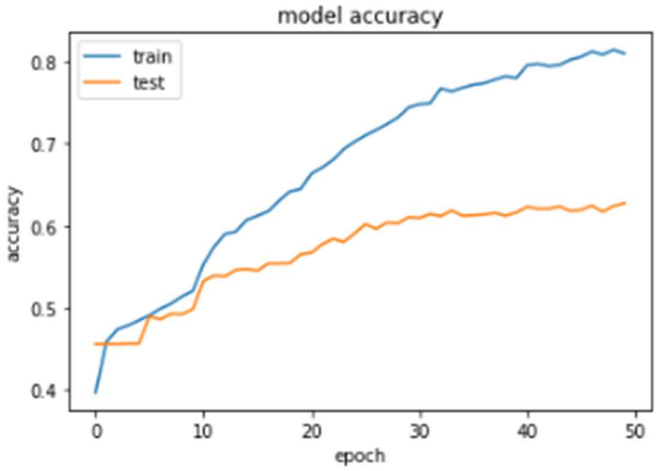
GROUP 24

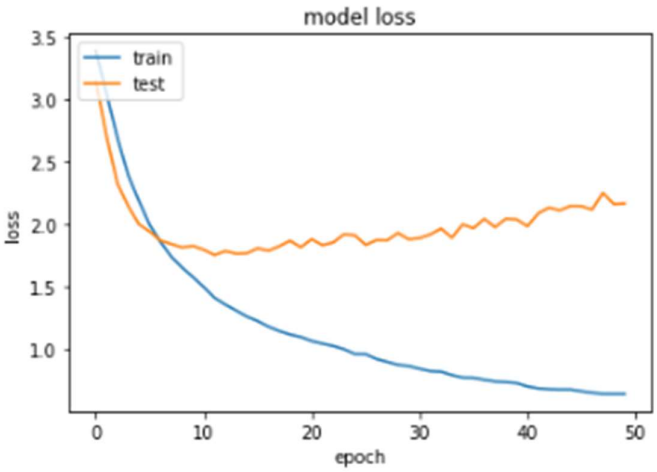
GROUP 12



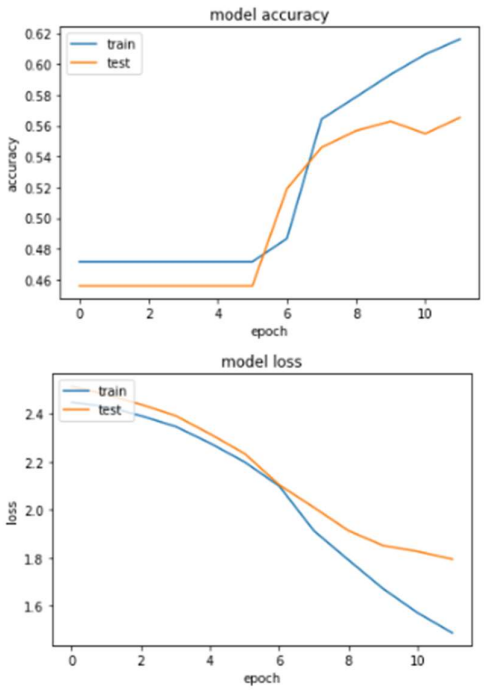
DEEP LEARNING MODELS -RAW-PRE-PROCESSED.

- **SIMPLE NEURAL NETWROK**

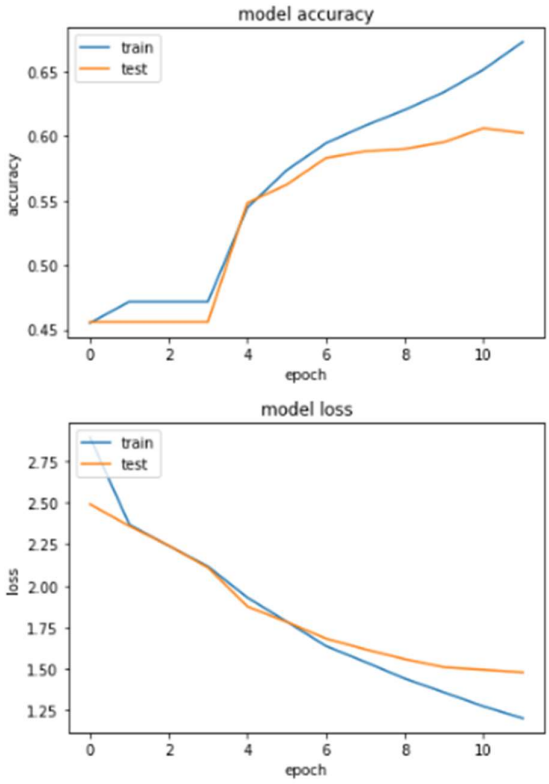




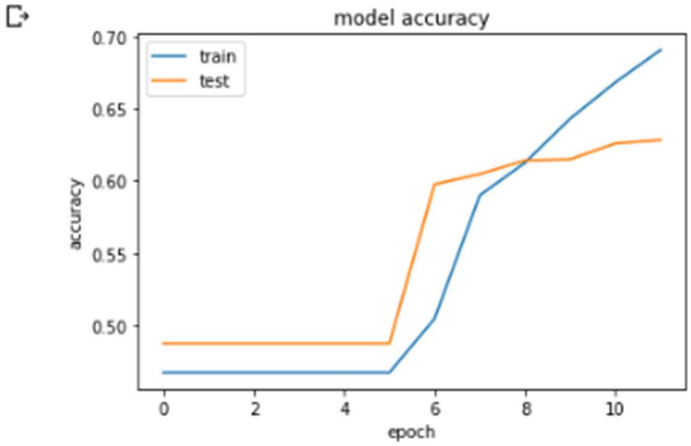
- **LSTM**

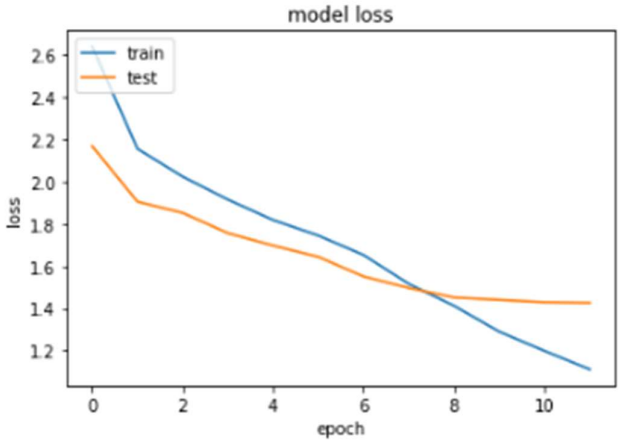


- **LSTM WITH PRE-TRAINED EMBEDDINGS**



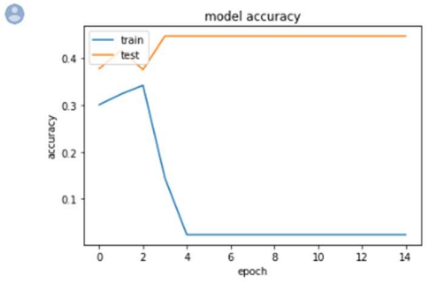
- **BI-DIRECTIONAL LSTM WITH PRE-TRAINED WEIGHTS.**

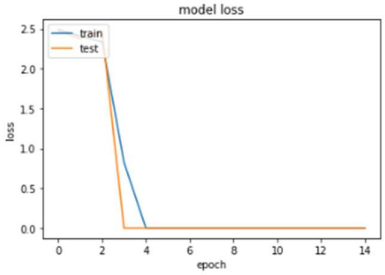




DEEP LEARNING MODELS -UPSAMPLED

- **SIMPLE NEURAL NETWORK**

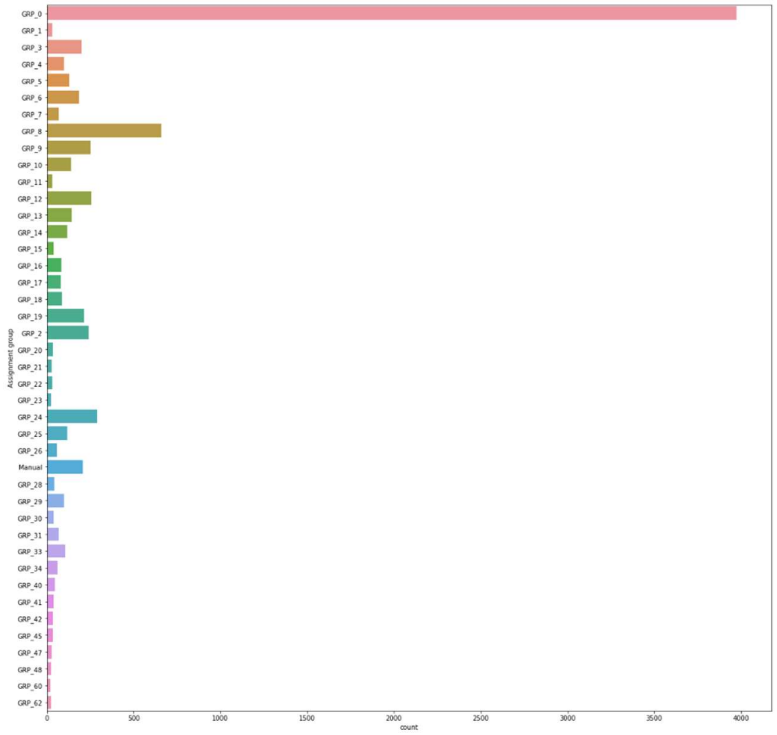




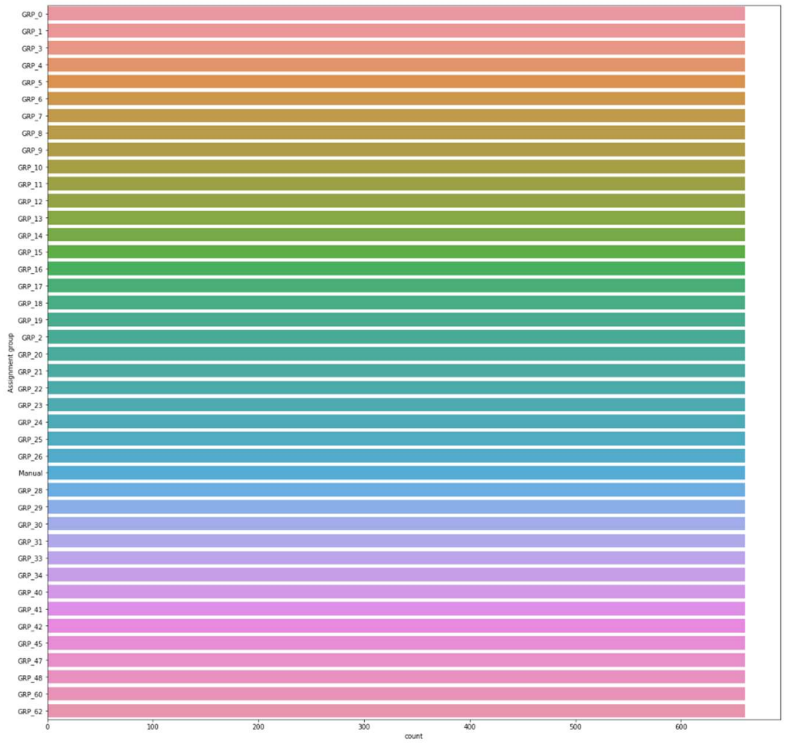
This is a very unusual pattern ,after 4 epochs the train and test accuracy remains constant.

DEEP LEARNING MODELS -DOWN SAMPLED.

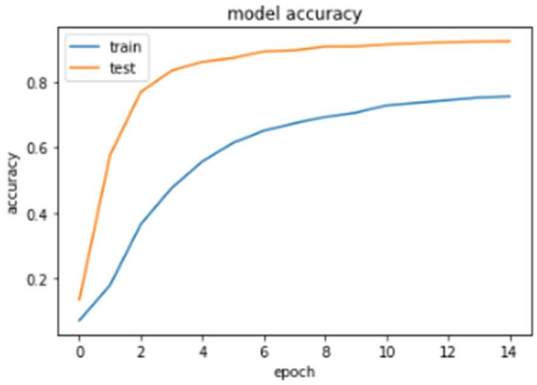
Before down sampling

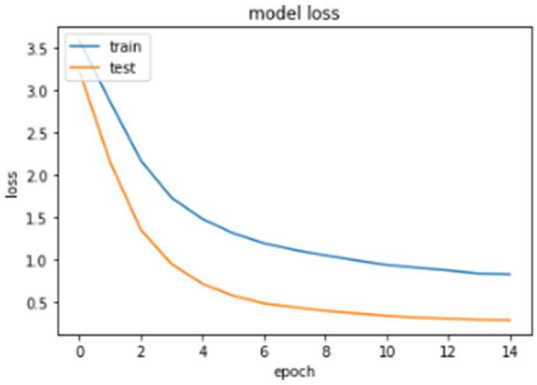


After down sampling

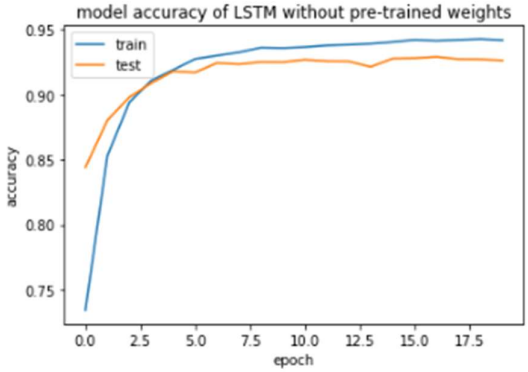


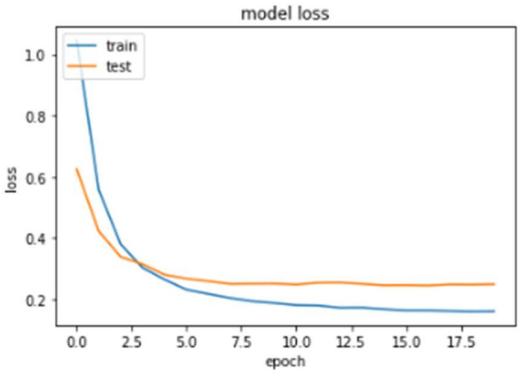
- **SIMPLE NEURAL NETWORK**



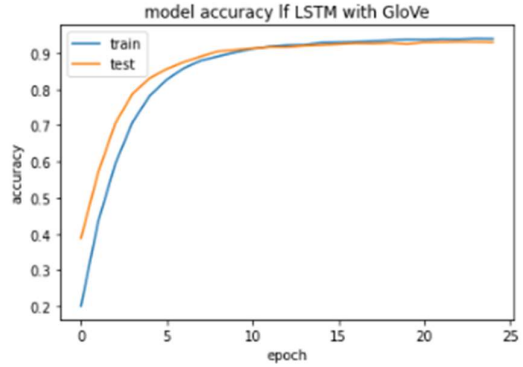


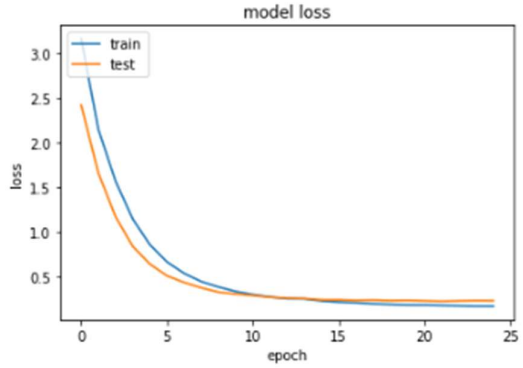
- **LSTM WITHOUT PRE-TRAINED WEIGHTS**



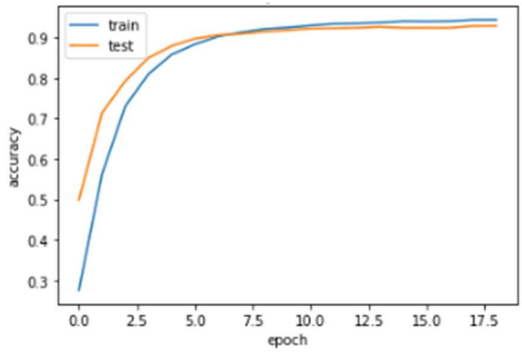


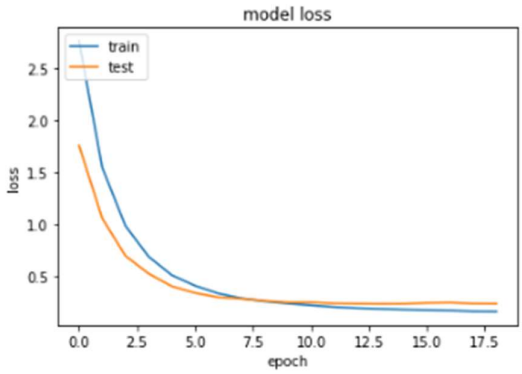
- **LSTM WITH PRE-TRAINED WEIGHTS**



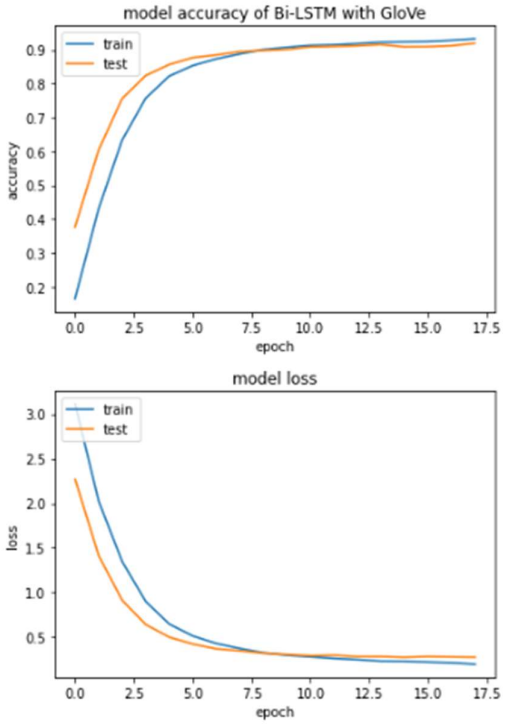


- **LSTM WITH PRE-TRAINED WEIGHTS-ITERATION 2**

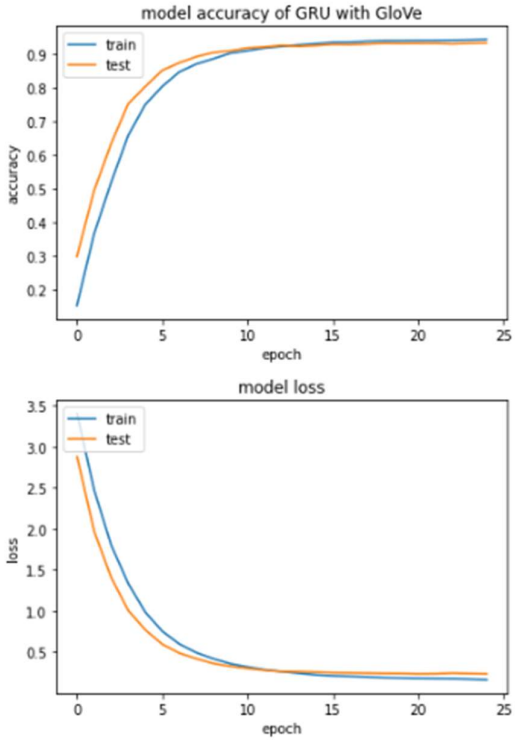




- **BI-DIRECTIONAL LSTM WITH PRE-TRAINED WEIGHTS**



- **GRU WITH PRE-TRAINED WEIGHTS**



7. IMPLICATIONS

Although this model can classify the IT tickets with an accuracy of 93% , to achieve better accuracy in the real world it would be great if the business can collect additional data around 300 records for each group.

8. LIMITATIONS

While checking the distribution of 8500 incidents across 74 Assignment groups, it was noted that there were few groups has tickets 20 or less tickets, we had to club the groups with less than 20 tickets into a new group called **Manual** and reduced the total number of Assignment groups from 74 to 42. Manual assignment will be required for the tickets belonging to these group.

9. CLOSING REFLECTIONS

We found the data was present in different languages and in various formats such as emails, chat etc. bringing in a lot of variability in the data to be analysed. The business can improve the process of raising tickets via a common unified IT ticket service portal which reduces the above-mentioned variability.
By doing this, the model can perform better which can help business to identify the problem area for relevant clusters of topics.