

AIML - CAPSTONE PROJECT

FINAL REPORT – AUTOMATIC TICKET ASSIGNMENT

ABSTRACT:

The purpose of this project is to classify an IT ticket that the customer raises into an appropriate group so that the issue can be resolved at the earliest. This can be achieved by building a NLP model that can automatically classify the ticket and place it in the appropriate assignment group. The given dataset had short description & long description about the IT issue the customers had and the corresponding assignment group. The data contained descriptions in many languages (English, German and Non-ASCII). Also, there were lot of Junk characters, null values, duplicates. All these were treated and the data was converted to English text using google translate web API. We performed EDA on the dataset and observed a class imbalance, with data skewed towards one particular Group. The Bigram and trigram analysis for different Assignment groups yielded several words which were common across groups. We then built different embeddings for the dataset and ran machine learning algorithms and deep learning algorithms. We also tuned the deep learning models with different hyperparameter values to improve the model performance. The state of the art models like BERT and ULMFit were also implemented. The performance of the algorithms were measured with respect to accuracy and F1 scores. Confusion matrix and classification reports were also obtained for all the models. All the models had an accuracy score between 50% - 65%. Since the data is skewed, the accuracy was only moderate. With more data collected for all the other groups, the model will perform well with improved accuracy.

Group: Ram Gopal, Krithika, Mexi Dsouza, Karthickeyan, Mithun

Supervised & Mentored by: Survesh Chauhan

PM: Remya Nair

Date: 17 August 2020

Table of contents:
Contents

1.	INTRODUCTION:	4
2.	PROBLEM STATEMENT:	4
3.	BUSINESS DOMAIN VALUE:	4
4.	PROJECT DESCRIPTION:	5
4.1	DATASET:	6
4.2	SUMMARY OF TASKS PERFORMED:	7
4.2.1	MILESTONE 1: EDA, Data Cleansing & Pre-Processing	7
	A. Exploring the given Data files & Understanding the structure of data:	7
	B. Missing points in data:	7
	C. Finding inconsistencies in the data:	7
	D. Visualizing different patterns:	9
	E. Visualizing different text features	9
	F. Dealing with missing values	10
	G. Text pre-processing	11
	H. Creating word vocabulary from the corpus of report text data	14
	I. Tokenization, Lemmatization and Stop words removal:	14
	J. Combining of non-frequent Assignment groups	16
	K. Bigrams & Trigrams	17
	L. Word clouds for Bigram, Trigram and Groups	18
4.2.2	MILESTONE 2: Model Building	20
	A. Building a model architecture which can classify	21
	B. Trying different model architectures by researching state of the art for similar tasks.	23
	C. Train the model	25
	D. Save the Weights:	26
4.2.3	MILESTONE 3: Test the Model, Fine-tuning and Repeat	27
4.2.4	WORK DONE FOR FINAL SUBMISSION	29
4.3	BENCH MARKING – Comparison of Approaches Taken	388
4.4	FINAL CODE	399
4.5	CHALLENGES FACED	399
4.6	CONCLUSION	399

1. INTRODUCTION:

In this capstone project, the goal is to build a classifier that can classify the tickets by analysing text mentioned in the IT incident records.

2. PROBLEM STATEMENT:

One of the key activities of any IT function is to “Keep the lights on” to ensure there is no impact to the Business operations. IT leverages the Incident Management process to achieve the above Objective. An incident is something that is an 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 the 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.

3. BUSINESS DOMAIN VALUE:

In the support process, incoming incidents are analysed and assessed by the 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 the 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 on 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.

4. PROJECT DESCRIPTION:

The Agenda is to build a model which classifies the Incident tickets based on data given, the next step would be to do a series of activities which are bisected into milestones and listed below. The output generated in each stage of Milestone would be utilised as input for the next. We have listed down the completed milestones.

Milestone 1: Pre-Processing, Data Visualization and EDA

- Exploring the given Data files & understanding the structure
- Missing points in data
- Finding inconsistencies in the data
- Visualizing different patterns
- Visualizing different text features
- Dealing with missing values
- Text pre-processing
- Creating word vocabulary from the corpus of report text data
- Creating tokens as required

Milestone 2: Model Building

- Building a model architecture which can classify.
- Trying different model architectures by researching state of the art for similar tasks.
- Train the model

- 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

- Test the model and report as per evaluation metrics
- Try different models
- Try different evaluation metrics
- 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
- Report evaluation metrics for these models along with your observation on how changing different hyper parameters leads to change in the final evaluation metric.

PROJECT OBJECTIVE:

1. Learn how to use different classification models.
2. Use transfer learning to use pre-built models.
3. Learn to set the optimizers, loss functions, epochs, learning rate, batch size, checkpointing, early stopping etc.
4. Read different research papers of a given domain to obtain the knowledge of advanced models for the given problem.

4.1 DATASET:

We received the data for the above stated problem in excel format “input_data.xlsx”. Each record contains information about an IT incident. There are 4 columns in the given data as seen below:

Sample of records are shown below:

Short description	Description	Caller	Assignment group
login issue	-verified user details.(employee# & manager name)-checked the u	spxjnwir pjlcqds	GRP_0
outlook	received from: hmjdrvpb.komuaywn@gmail.comhello team,my n	hmjdrvpb komuaywn	GRP_0
cant log in to vpn	received from: eylqgodm.ybqkwiam@gmail.comhii cannot log on	eylqgodm ybqkwiam	GRP_0
unable to access hr_tool page	unable to access hr_tool page	xbkucsvz gcpydteq	GRP_0
skype error	skype error	owlgajme qhcozdfx	GRP_0
unable to log in to engineering too	unable to log in to engineering tool and skype	eflahbxn ltdgrvkz	GRP_0

- Short Description : Contains the title of the reported issue.
- Description : Contains the detailed explanation of the reported issue.
- Caller : User name who logged/called for the incident
- Assignment Group : Different category of groups in which the raised issues are assigned.

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

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

4.2 SUMMARY OF TASKS PERFORMED:

4.2.1 MILESTONE 1: EDA, Data Cleansing & Pre-Processing

A. Exploring the given Data files & Understanding the structure of data:

The data is composed and given in the form of an Excel file, "input_data.xlsx ". We decided to import the excel file content and store as a [Data-Frame](#) for further exploratory activities. The dimension of the imported data-frame from the excel is 8500 rows spread across 4 columns.

▼ Reading and Exploring Data

```
[ ] data = pd.read_excel(Excel_data_file)
data.head()
```

	Short description	Description	Caller	Assignment group
0	login issue	-verified user details.(employee# & manager na...	spjxnwir pjloqds	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: eylogodm.ybqkwiam@gmail...	eylogodm 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.shape
```

```
(8500, 4)
```

```
[ ] data.columns
```

```
Index(['Short description', 'Description', 'Caller', 'Assignment group'], dtype='object')
```

B. Missing points in data:

There were 9 Null records placed in two columns which needs to be removed:

```
[ ] data.isna().sum()
```

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

C. Finding inconsistencies in the data:

In the 8500 rows of information, we had inconsistent data type which are listed below:

1. Duplicate cell values

```
[ ] #Find duplicate records in the given data
duplicate_data = data[data.duplicated()]
print(len(duplicate_data))
```

83

2. Non-ASCII values.

8471	plant_101 value added services - one day pick route project.	request to phase in additional vas customers. file attached with c\xnqzhtwu hivumtfz	GRP_18
8472	please review your recent ticketing_tool tickets and let me know	from: mikhghytr wafiglhdhrop sent: thursday, august 04, 2016 8:51 azxhejvq fyemlavd	GRP_16
8473	ç"µè„'â%€æœ%â%€ä„â%æ%ä	to â"b"9%Œæ—©ä„šç"µè„'â%€æœ%â%€ä„â%æ%ä	GRP_30
8474	ticket update	from: rakthesh ramdntythanesh sent: friday, august 05, 2016 5:2 eqzibjhw ymebpoih	GRP_0

3. Special Characters.

8318	network outage: sao pollaurido-mercedes benz plant network is	what type of outage: __x__ network __circuit__ pov dkmcfreg anwmfvlg	GRP_8
8319	designation change required in outlook.	----- original message -----from: bvlcarfe aztlkeifowndararaje bvlcarfe aztlkeif	GRP_2
8320	HostName_1015: average (4 samples) disk free on c:\ is now 15%	HostName_1015: average (4 samples) disk free on c:\ is now 15% rkupshb gsmzfojw	GRP_12
8321	HostName_111 : average disk free on e:\ is now 10%, out of tota	noticed the alarm in reporting_tool .which is below the warning t\uxgrdfc kqxdjeov	GRP_8
8322	abended job in job_scheduler: bk_hana_SID_62_erp_dly_dp	received from: monitoring_tool@company.com abended job in jc ZkBoxib QsElzdZO	GRP_8
8323	reset passwords for xuqvaobxuy ntquocx using password_mana change to new password: alabama4\$	klrghwc ykjrblvs	GRP_17

4. Random combination of alphabets which are not readable.

8494	vip2: windows password reset for tifpdchb pedxruyf	vip2: windows password reset for tifpdchb pedxruyf	oybwdsqx oxyhwrzf	GRP_0
8495	machine nÃ£o estÃ¡j funcionando	i am unable to access the machine utilities to finish the drawers a ufawcgob aowhxjky		GRP_62
8496	an mehreren pc's lassen sich verschiedene prgramdntyme nicht Ä	an mehreren pc's lassen sich verschiedene prgramdntyme nicht Ä	kqvbrspl jyzoklfx	GRP_49

5. Non-English words (The foreign language we had found the most is German).

4359	re; erp	received from: btyvqhvw.xbyolhsw@gmail.comgood daytrust you zvenmiap kocjpnl	GRP_0
	an terminal 12 bei iso-u kÄ¶Innen im EU_tool keine zeichnungen	mehr aufgerufen werden!! diese funktion wird wegen der	
4360	mehr aufgerufen werden!!	auftragsbereitstellung benÄ¶tigt!!!!	tiefszyh sfujdlgv
4361	bobj error	issue while refreshing the global spend analytics v2 report in bobj	ginjmaxk zumkvfeb
			GRP_9

6. Static words which are not relevant to the Ticket.

Static Disclaimer at the end of emails:

4399	vip2: printer driver update	vip2: printer driver update	fabijhsd ocsnugeh	GRP_0
		please unlock user ghvireicj immediately in erp and extend the account to 31 dec 16. it is immediately needed ä€" tomorrow is our py run!!!!		
		von: tszvorba wrtdpnx [mailto:reichard@ppstrixner.de] gesendet: mittwoch, 14. september 2016 16:57 an: nkjtoxwv wqtlzvzu cc: lixwgnto krutnylz betreff: in erp gesperrt - bitte dringend wieder frei schalten wichtigkeit: hoch		
		hallo herr hohgajnn,		
4400	erp user blocked, please unlock immediately & extend the accou	ich wollte mich eben ein weiteres mal anmelden und habe auch sicher die korrekten anmeldedaten eingegeben. leider bin ich nun gesperrt.	lixwgnto krutnylz	GRP_0

Static eMail attributes:

4399	vip2: printer driver update	vip2: printer driver update	fabijhsd ocsnugeh	GRP_0
		received from: lixwgnto.krutnylz@gmail.com		
		please unlock user ghvireicj immediately in erp and extend the account to 31 dec 16. it is immediately needed ä€" tomorrow is our py run!!!!		
		von: tszvorba wrtdpnx [mailto:reichard@ppstrixner.de] gesendet: mittwoch, 14. september 2016 16:57 an: nkjtoxwv wqtlzvzu cc: lixwgnto krutnylz betreff: in erp gesperrt - bitte dringend wieder frei schalten wichtigkeit: hoch		
		hallo herr hohgajnn,		
4400	erp user blocked, please unlock immediately & extend the accou	ich wollte mich eben ein weiteres mal anmelden und habe auch sicher die korrekten anmeldedaten eingegeben.	lixwgnto krutnylz	GRP_0

7. The information provided in the Caller is not readable and usable, but before deciding to drop it from pre-processing we analysed and got some insights. There are only 2950 callers in total for 8500 records which means that each caller may have raised one or more than one ticket.


```
data['Caller'].nunique()
2948

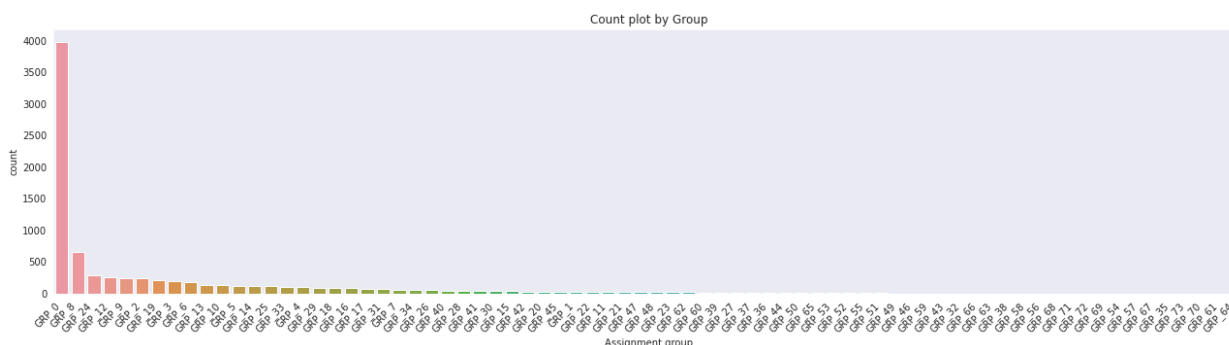
[ ] data['Caller'].value_counts()
bpcwthsn kzqsbmtp      810
Zk8ogxib QsE3zdzo      151
fumkcsji sarwtlhy      134
rbozivdq gmlhrtvp       87
rkupnshb gsmzfojw       71
...
vebizknu qlknijvb        1
ruhbyzpv vksnjti         1
eahkpnbm uptiveok         1
eqcudbks zbjeqrui         1
psbulrtd jxkvzmnf         1
Name: Caller, Length: 2948, dtype: int64
```

From the above data, it is clearly visible that the caller data is also highly skewed with the max being 810 but even the 75% data is at 3 records per caller. There were totally 2950 unique callers of which 1460 callers have created only 1 ticket. The remaining 1460 callers have created more than 2 tickets. ***bpcwthsn kzqsbmtp*** is the caller who raised maximum tickets. The topmost word used in short description is **password reset**

D. Visualizing different patterns:

Each ticket is assigned to a different category of groups and that information is available under the Assignment group column. When we applied the data to check the distribution of tickets across the groups, we had found the dataset is highly imbalanced and skewed with GRP_0.

Around 46% of the dataset is represented by just one class GRP_0. There are many classes that have just 1 datapoint which leads to bias in prediction. The other groups with high records are GRP_8, GRP_9, GRP_12 and GRP_24.



E. Visualizing different text features

As we got data which contains multiple columns with multiple rows of information, we used [Word Cloud](#) technique to understand the frequency and significant textual data points of words in the columns on raw data.



F. Dealing with missing values

We had 9 records with Null values in the given data-frame, which we removed:

```
data.isna().sum()
```

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

Observations:

The data has 8500 records/articles and 4 columns. There are null records in the data as seen above. Remove the 9 records that had null values.

```
[ ] #Dropping the null values
data.dropna(inplace=True)
```

```
[ ] data.shape
```

```
(8491, 4)
```

Removed the 9 records with null values since it does not add any value in our model building and prediction

G. Text pre-processing

Since the given data is not fit to be processed directly, it requires Cleansing followed by pre-processing for which we derived a number of functions to have a junk-free, clean and valuable data for asserting the classification model.

For ease of work we combined the “Short Description” & “Description” columns together to make sure all the data relevant to the ticket are captured properly and keeping these two columns separately doesn’t make any sense sine both contains repeated strings in most of the records.

```
[ ] #Having callers unique list separately to remove the caller names from description later.
callers = data['caller'].unique()

[ ] ## merging the Short Description and Description Columns
new_data= pd.DataFrame({"Description": data["Short description"] + " " + data["Description"], "AssignmentGroup": data["Assignment group"]}, columns=["Description","AssignmentGroup"])

[ ] new_data.head()
```

	Description	AssignmentGroup
0	login issue -verified user details.(employee# ...	GRP_0
1	outlook \r\n\r\nreceived from: hmjdrvpb.komuay...	GRP_0
2	cant log in to vpn \r\n\r\nreceived from: eyiq...	GRP_0
3	unable to access hr_tool page unable to access...	GRP_0
4	skype error skype error	GRP_0

```
[ ] new_data.shape

(8408, 2)
```

As part of Pre-Processing we defined a number of functions as listed below:

1. Removal of Duplicate values:

```
[ ] #Find duplicate records in the given data
duplicate_data = data[data.duplicated()]
print(len(duplicate_data))

83

[ ] duplicate_data.head()
```

	Short description	Description	Caller	Assignment group
51	call for ecwtrjnj jpecxuty	call for ecwtrjnj jpecxuty	oickhmvx pqqobjnd	GRP_0
229	call for ecwtrjnj jpecxuty	call for ecwtrjnj jpecxuty	oickhmvx pqqobjnd	GRP_0
493	ticket update on inplant_872730	ticket update on inplant_872730	fumkcsji samtlthy	GRP_0
512	blank call //gso	blank call //gso	rbozldq gmlhrtvp	GRP_0
667	job bkbackup_tool_powder_prod_full failed in j...	received from: monitoring_tool@company.com\r\n...	bpctwhsn kzqsbmtip	GRP_8

```
[ ] #Removing those duplicates
data = data.drop_duplicates()
data.shape

(8408, 4)
```

There were 83 records with duplicate values which we removed from the data.

2. Non-English & Non-ASCII character handling via Google API

As part of data preprocessing, we found that given data had a lot of Non-English text and non-ASCII characters, especially lots of data in German language. So, we decided to translate

all the data to English language and to achieve that, we used the google translator API to translate the given data for those records that are not in English language.

```
[ ] df_before_translation = data.copy() ##Taking backup before translation
df_before_translation.tail()
```

	Short description	Description	Caller	Assignment group
8495	emails not coming in from zz mail	\r\n\r\nreceived from: avglimrts.vhqmtlua@gmail...	avglimrts.vhqmtlua	GRP_29
8496	telephony_software issue	telephony_software issue	rbozivdq.gmlhrtvp	GRP_0
8497	vip2: windows password reset for tifpdchb pedx...	vip2: windows password reset for tifpdchb pedx...	oybwdsqg oxyhwrftz	GRP_0
8498	machine nÃ&Eo estÃ&Ij funcionando	i am unable to access the machine utilities to...	ufawcgob.aowhxjky	GRP_62
8499	an mehreren pc's lassen sich verschiedene prgr...	an mehreren pc's lassen sich verschiedene prgr...	kqvbrspl.jyzokifx	GRP_49

```

#Translate to english if the given sentence is not in english.
def Translate_to_English(x):
    translator = Translator()
    if translator.detect(x).lang != 'en':
        #print("Source: ", x)
        translatedText = translator.translate(x).text
        #print("Translated text in English: ", translatedText)
    else:
        translatedText = x
    return translatedText

[ ] #Translate the description column and short description column
for i in data.index:
    data['Description'][i] = Translate_to_English(str(data['Description'][i]))
    data['Short description'][i] = Translate_to_English(str(data['Short description'][i]))

```

After translation, we can notice that the description and Short description column are translated to English language.

```
[ ] data.tail()
```

	Short description	Description	Caller	Assignment group
8495	emails not coming in from zz mail	\r\n\r\nreceived from: avglimrts.vhqmtlua@gmail...	avglimrts.vhqmtlua	GRP_29
8496	telephony_software issue	telephony_software issue	rbozivdq.gmlhrtvp	GRP_0
8497	vip2: windows password reset for tifpdchb pedx...	vip2: windows password reset for tifpdchb pedx...	oybwdsqg oxyhwrftz	GRP_0
8498	machine is not working	i am unable to access the machine utilities to...	ufawcgob.aowhxjky	GRP_62
8499	Different prgramdntypes cannot be opened on se...	Different prgramdntypes cannot be opened on se...	kqvbrspl.jyzokifx	GRP_49

3. Disclaimer removal:

The below **Remove_Disclaimer()** function would remove the disclaimer messages given as part of the email sent by callers which is not required for our classification. The reference Disclaimer messages were taken from the dataset after Translating to English.

```

[ ] #this function is to remove the disclaimer messages given as part of the email sent by callers which is not needed
def Remove_Disclaimer(text):
    text = str(text)
    strDisclaimerMsg1 = r'this communication is intended solely for the use of the addressee and may contain information that is sensitive, confidential or excluded from disclosure in
    strDisclaimerMsg2 = r'select the following link to view the disclaimer in an alternate language.'
    #to remove the pattern '[ # + company / posts> ['
    strDisclaimerMsg3 = r'\[.*?]'
    strDisclaimerMsg4 = r'this message is intended for the exclusive use of the person to whom it is addressed and may contain privileged, confidential information that is exempt from
    #text=text.lower()
    text = re.sub(strDisclaimerMsg1, ' ', str(text))
    text = re.sub(strDisclaimerMsg2, ' ', str(text))
    text = re.sub(strDisclaimerMsg3, ' ', str(text))
    text = re.sub(strDisclaimerMsg4, ' ', str(text))

    text = text.strip()
    return text

```

We defined a function **clean_data()** which we use to replace the known email attributes which will be present generically over the email messages, Numbers, New line characters, remove only Hashtag while keeping Hashtag text, HTML Special entities like & , Hyperlinks, Removing characters beyond readable format by Unicode and Unreadable format and extra spaces.

```
def clean_data(text):
    text=text.lower()
    text = ' '.join([w for w in text.split() if not is_valid_date(w)])

    text = Remove_Disclaimer(text)
    text = re.sub(r"received from:", ' ', text)
    text = re.sub(r"from:", ' ', text)
    text = re.sub(r"to:", ' ', text)
    text = re.sub(r"subject:", ' ', text)
    text = re.sub(r"sent:", ' ', text)
    text = re.sub(r"ic:", ' ', text)
    text = re.sub(r"cc:", ' ', text)
    text = re.sub(r"bcc:", ' ', text)
    #Remove email
    text = re.sub(r'\s*@\s*\s?', ' ', text)
    # Remove numbers
    text = re.sub(r'\d+', ' ', text)
    # Remove new line characters
    text = re.sub(r'\n', ' ', text)
    # Remove hashtag while keeping hashtag text
    text = re.sub(r'#', ' ', text)
    #&
    text = re.sub(r'&?', 'and', text)
    # Remove HTML special entities (e.g. & )
    text = re.sub(r'&w*;', ' ', text)
    # Remove hyperlinks
    text = re.sub(r'https?:\V.*\Vw*', ' ', text)
    # Remove characters beyond Readable format by Unicode:
    text= ' '.join(c for c in text if c <= '\uFFFF')
    text = text.strip()
    # Remove unreadable characters (also extra spaces)
    text = ' '.join(re.sub(r"^\u0030-\u0039\u0041-\u005a\u0061-\u007a", " ", text).split())
    for name in callers:
        namelist = [part for part in name.split()]
        for namepart in namelist:
            text = text.replace(namepart, '')

    text = re.sub(r"\s+[a-zA-Z]\s+", ' ', text)
    text = re.sub(' +', ' ', text)
    text = text.strip()
    return text
```

4. We had some special cases, where certain type of texts might sound as junk but the text cannot be simply cleansed out of the data-frame but needs improved handling to retain as proper strings for better prediction as they are part of IT Incidents raised.

Cases like FTP Location and IP will always accompany with location reference in format of numbers and special characters, that information are properly prefixed with the reference words to identify and ensured not cleaned. We had the below function **preprocess_replace()** for that purpose.

```
#Replace known formats with proper strings for better prediction
def preprocess_replace(text1):
    text1=text1.replace(to_replace="[Hh][Oo][Ss][Tt][Nn][Aa][Mm][Ee]_[0-9]*", value='hostname ', regex = True)
    text1=text1.replace(to_replace="ftp.*", value='ftp location ', regex=True)
    text1=text1.replace(to_replace="[a-z0-9]*.xlsx", value='excel ', regex=True)
    text1=text1.replace(to_replace="outside:[0-9./]*", value='outside ipaddress ', regex=True)
    text1=text1.replace(to_replace="inside:[0-9./]*", value='inside ipaddress ', regex=True)
    text1=text1.replace(to_replace="\\*hostname_[0-9]*", value='hostname ', regex=True)
    text1=text1.replace(to_replace="lmsl[0-9]*", value='lmsl ', regex=True)
    text1=text1.replace(to_replace="SID_[0-9][0-9]", value='sid ', regex = True)
    text1=text1.replace(to_replace="[Tt]icket_no[0-9]*", value='ticket_no ', regex=True)
    text1=text1.replace(to_replace="[jJ]ob_[0-9_a-z]*", value='job_id ', regex=True)
    text1=text1.replace(to_replace="[0-9]+.[0-9]+.[0-9]+.[0-9]+/[0-9]*", value='ipaddress ', regex=True)
    return text1
```

H. Creating word vocabulary from the corpus of report text data

We had created a Corpus which contains the data after cleansing:

```
[ ] # Create Dictionary
id2word = corpora.Dictionary(final_data1['BagOfWords'])

# Create Corpus from post clean data
texts = final_data1['BagOfWords']

# Term Document Frequency and Bag of words
corpus = [id2word.doc2bow(text) for text in texts]

[ ] # View as ID
print(corpus[:1])

[[ (0, 1), (1, 1), (2, 1), (3, 1), (4, 2), (5, 1), (6, 1), (7, 1), (8, 1), (9, 1), (10, 1), (11, 1), (12, 1), (13, 1), (14, 1), (15, 1), (16, 1)]]

[ ] # View as word
print([[(id2word[id], freq) for id, freq in cp] for cp in corpus[:1]])

[[('able', 1), ('ad', 1), ('advised', 1), ('caller', 1), ('check', 2), ('confirmed', 1), ('detail', 1), ('employee', 1), ('issue', 1), ('login', 1),
```

I. Tokenization, Lemmatization and Stop words removal:

As part of NLP data processing, it is important to do Tokenize, Lemmatize and to remove Stop Words. We have used the WordNetLemmatizer class for lemmatization, done the POS tagging for all the words and also tokenization using wordNet.

```
▶ lemmatizer = WordNetLemmatizer()

# function to convert nltk tag to wordnet tag
def nltk_tag_to_wordnet_tag(nltk_tag):
    if nltk_tag.startswith('J'):
        return wordnet.ADJ
    elif nltk_tag.startswith('V'):
        return wordnet.VERB
    elif nltk_tag.startswith('N'):
        return wordnet.NOUN
    elif nltk_tag.startswith('R'):
        return wordnet.ADV
    else:
        return None

# Function to convert array into string
def listToString(s):
    str1 = " "
    return (str1.join(s))
```

```
def lemmatize_sentence(sentence):
    #tokenize the sentence and find the POS tag for each token
    nltk_tokenized = nltk.word_tokenize(sentence)
    ordered_tokens = set()
    result = []
    #remove duplicate words in sentence
    for word in nltk_tokenized:
        if word not in ordered_tokens:
            ordered_tokens.add(word)
            result.append(word)
    new_desc = listToString(result)

    nltk_tagged = nltk.pos_tag(ordered_tokens)
    #tuple of (token, wordnet_tag)
    wordnet_tagged = map(lambda x: (x[0], nltk_tag_to_wordnet_tag(x[1])), nltk_tagged)
    lemmatized_sentence = []
    for word, tag in wordnet_tagged:
        if tag is None:
            #if there is no available tag, append the token as is
            lemmatized_sentence.append(word)
        else:
            #else use the tag to lemmatize the token
            lemmatized_sentence.append(lemmatizer.lemmatize(word, tag))
    return " ".join(lemmatized_sentence), new_desc
```

We have performed the clean-up again and also removed some of the common words used that does not add any context to the classification like **hello, good morning, hi team**, etc just to ensure no junk values are getting passed after this stage:

```
temp = []
temp1 = []
for sentence in new_data["Description"]:
    sentence = sentence.lower()
    cleanr = re.compile('<.*?>')
    sentence = re.sub(cleanr, ' ', sentence) #Removing HTML tags
    sentence = re.sub(r'\S+@\S+', 'EmailId', sentence)
    sentence = re.sub(r'\'', '', sentence, re.I|re.A)
    sentence = re.sub(r'[0-9]', '', sentence, re.I|re.A)
    sentence = re.sub(r'^a-zA-Z0-9\s', ' ', sentence)
    sentence = sentence.lower()
    sentence = re.sub(r'com ', ' ', sentence, re.I|re.A)
    sentence = re.sub(r'hello team', ' ', sentence, re.I|re.A)
    sentence = re.sub(r'hello ', ' ', sentence, re.I|re.A)
    sentence = re.sub(r'hi team', ' ', sentence, re.I|re.A)
    sentence = re.sub(r'hi ', ' ', sentence, re.I|re.A)
    sentence = re.sub(r'hello team', ' ', sentence, re.I|re.A)
    sentence = re.sub(r'best', ' ', sentence, re.I|re.A)
    sentence = re.sub(r'kind', ' ', sentence, re.I|re.A)
    sentence = re.sub(r'hello helpdesk', ' ', sentence, re.I|re.A)
    sentence = re.sub(r'good morning ', ' ', sentence, re.I|re.A)
    sentence = re.sub(r'good afternoon ', ' ', sentence, re.I|re.A)
    sentence = re.sub(r'good evening ', ' ', sentence, re.I|re.A)
    l_sentence, new_desc = lemmatize_sentence(sentence)
```

```
[ ] #Add the corrected description and bag of words in the data frame
new_data['BagOfWords'] = temp
new_data['NewDescription'] = temp1
```

```
[ ] new_data.head()
```

	Description	AssignmentGroup	BagOfWords	NewDescription
0	login issue verified user details employee and...	GRP_0	[detail, ad, caller, able, manager, issue, res...	login issue verified user details employee man...
1	outlook hello team my meetings skype meetings ...	GRP_0	[appear, advise, correct, outlook, calendar, s...	outlook meetings skype etc appearing calendar ...
2	cant log in to vpn hi cannot log on to vpn best	GRP_0	[log, vpn, cant]	cant log vpn
3	unable to access hr tool page unable to access...	GRP_0	[tool, access, unable, page, hr]	unable access hr tool page
4	skype error skype error	GRP_0	[error, skype]	skype error

We have also calculated and added the length of every description record and number of words in every Description record and added that to the dataframe to understand the word distribution

```
[ ] final_data1 = final_data1[['Description', 'BagOfWords', 'length', 'num_words', 'AssignmentGroup']]
final_data1.head()
```

	Description	BagOfWords	length	num_words	AssignmentGroup
0	login issue verified user details employee man...	[detail, ad, caller, able, manager, issue, res...	126	18	GRP_0
1	outlook meetings skype etc appearing calendar ...	[appear, advise, correct, outlook, calendar, s...	76	10	GRP_0
2	cant log vpn	[log, vpn, cant]	12	3	GRP_0
3	unable access hr tool page	[tool, access, unable, page, hr]	26	5	GRP_0
4	skype error	[error, skype]	11	2	GRP_0

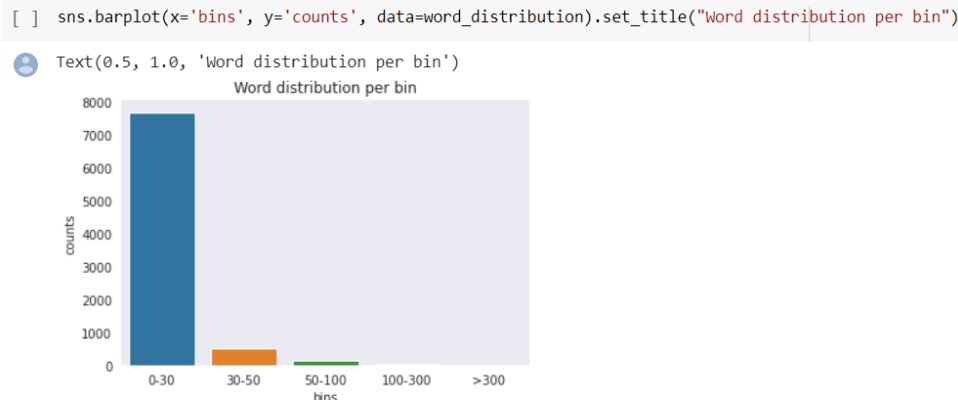
```
[ ] final_data1.shape
```

(8377, 5)

```
[ ] final_data1.describe().transpose()
```

	count	mean	std	min	25%	50%	75%	max
length	8377.0	90.981019	140.858365	3.0	26.0	50.0	109.0	3263.0
num_words	8377.0	13.064582	18.884464	1.0	4.0	8.0	16.0	429.0

Word Distribution



There are no records that have greater than 500 words, Almost 98% of the records have words in the range of 1-30 which means most of the customers have given only short descriptions in their issue.

J. Combining of non-frequent Assignment groups

Since ticket distribution is not the same across the groups, we decided to take out the Groups which have less data and to combine them as one single group called *LeastDataGroup*. We had seen in the EDA that there are 25 groups that had less than 10 samples only and this will not help in predictions. Hence combining them into one group will help to categorize/classify the tickets correctly. We will see how the two data differ in the performance while modelling the data.

1. Data without grouping,
2. Data with grouping


```
[ ] AssignmentGrp = Final_Data_Grouped.groupby(['AssignmentGroup'])
LeastDataGroup=[]
for grp in Final_Data_Grouped['AssignmentGroup'].unique():
    if(AssignmentGrp.get_group(grp).shape[0]<10):
        LeastDataGroup.append(grp)
print('Number of groups that has less than 10 samples: ', (len(LeastDataGroup)))
Final_Data_Grouped['AssignmentGroup']=Final_Data_Grouped['AssignmentGroup'].apply(lambda x : 'least_data_grp' if x in LeastDataGroup else x)
```

```
Number of groups that has less than 10 samples: 25
```

```
[ ] Final_Data_Grouped['AssignmentGroup'].nunique()
```

```
50
```

```
[ ] Final_Data_Grouped['AssignmentGroup'].unique()
```

```
array(['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', 'least_data_grp', 'GRP_36', 'GRP_37', 'GRP_39',
      'GRP_40', 'GRP_41', 'GRP_42', 'GRP_44', 'GRP_45', 'GRP_47',
      'GRP_50', 'GRP_53', 'GRP_48', 'GRP_60', 'GRP_62', 'GRP_65'],
      dtype=object)
```

As you can see, the uniqueness count is reduced from 74 to 50 groups now for Assignment Group. The regrouping is done because with very less samples (each group having 1 or 2 records) might not help us in proper classification and that can impact our accuracy.

K. Bigrams & Trigrams

A **bigram** makes a prediction for a word based on the one before, and a **trigram** makes a prediction for the word based on the two words before that. We have built the Bigram and Trigram model for the preprocessed data using gensim models and see how the words are co-occurring together in the given data. The intention here is to utilise the advantage of Bigram and Trigram to understand whether the Ticket context is valid and to clean up word occurrences which are not correct.

```
[ ] # define the bigram and trigram models (higher threshold is used to have fewer phrases)
bigram = gensim.models.Phrases(bag_of_words, min_count=5, threshold=100)
trigram = gensim.models.Phrases(bigram[bag_of_words], threshold=100)

# Faster way to get a sentence clubbed as a trigram/bigram
bigram_model = gensim.models.phrases.Phraser(bigram)
trigram_model = gensim.models.phrases.Phraser(trigram)

[ ] #Build bigram and trigram model for given data
def Build_Bigrams(texts):
    return [bigram_model[doc] for doc in texts]

def Build_Trigrams(texts):
    return [trigram_model[doc] for doc in texts]

[ ] # Form Bigrams
data_words_bigrams = Build_Bigrams(bag_of_words)
print(data_words_bigrams)

#Form trigrams
data_words_trigrams = Build_Trigrams(data_words_bigrams)
print(data_words_trigrams)

[['login', 'issue', 'verified', 'user', 'details_employee', 'manager', 'name', 'checked_ad', 're
[['login', 'issue', 'verified', 'user', 'details_employee_manager', 'name', 'checked_ad', 'reset
```


GRP 0 is mostly account related issues like skype account, vpn account, access issues where HR or manager help is required to proceed further.

GRP 8 is mostly related to server issues, scheduler job issues that is not working or inactive and needs a restart.

GRP 24 is mostly related to computer issues like screen issues, drive issues, computer slowness, Bluetooth issues etc which needs support to be fixed.

Challenges Faced & Approach Taken to handle the issues arrived during Interim Submission:

1. Handling of Skewed Data:

As we know that the data-set we got is heavily skewed based on the tickets assigned to the groups, Hence we had different approaches among us.

Approach 1 – Dropping the groups which has less tickets assigned into it and to model for the data or Combining the groups which have least tickets assigned to form a group and consider that one among the group, hence we are not dropping even one value and not taking the risk of losing even a single value.

Approach 2 – Include every group information for modelling and not dropping or combining any of them.

Finally, to address an individual person's concern, we neither dropped nor used the group information as is, we decided to combine the least value group and treat it as one group.

2. Handling of Junk Values:

The data-set had to be cleansed and handled for numerous type of junk value and at the same time we have some essential values which might look like junk value, so most of the project duration had gone for the finalising the approach to derive the functions we had to cleanse the data and to retain essential information for classification.

4.2.2 MILESTONE 2: Model Building

We have used some of the traditional Machine learning models with TFIDF Vectorization and also Deep Learning models to see how the results are achieved for both kinds of data approach that we are using.

I. Grouped Data

II. Ungrouped Data

Split Train, Test & Validation Data for Ungrouped Data:

```
# Split into Test, train, validation for Ungrouped data

train_ratio = 0.60
validation_ratio = 0.20
test_ratio = 0.20

final_data1.head()
X = final_data1['Description']
y = final_data1['AssignmentGroup']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=1 - train_ratio)

X_val, X_test, y_val, y_test = train_test_split(X_test, y_test, test_size=test_ratio/(test_ratio + validation_ratio))
```

Split Train, Test & Validation Data for Grouped Data:

```
# Split into Test, train, validation for Ungrouped data

train_ratio = 0.60
validation_ratio = 0.20
test_ratio = 0.20

Final_Data_Grouped.head()
X_Grped = Final_Data_Grouped['Description']
Y_Grped = Final_Data_Grouped['AssignmentGroup']
X_train_Grped, X_test_Grped, y_train_Grped, y_test_Grped = train_test_split(X_Grped, Y_Grped, test_size=1 - train_ratio)

X_val_Grped, X_test_Grped, y_val_Grped, y_test_Grped = train_test_split(X_test_Grped, y_test_Grped, test_size=test_ratio/(test_ratio + validation_ratio))
```

A. Building a model architecture which can classify

We have used the following traditional Machine learning model classifiers with TF/IDF:

1. Logistic regression Model
2. Support Vector Classifier
3. Decision Tree Classifier
4. Random Forest Classifier
5. AdaBoost Classifier

We have performed the model building and evaluation for both grouped and ungrouped approaches that we followed. We have used the Pipeline to define the model and then we have done the model fit and prediction. A snapshot of the models used are shown below:

1. Logistic Regression Model:

```
lr_grped = Pipeline([('vect', CountVectorizer()),
                    ('tfidf', TfidfTransformer()),
                    ('clf', LogisticRegression()),
                    ])

lr_grped.fit(X_train_Grped, y_train_Grped)
y_pred_grped = lr_grped.predict(X_test_Grped)

LR_Grp_Accuracy_Score = accuracy_score(y_test_Grped, y_pred_grped)
print('accuracy %s' % LR_Grp_Accuracy_Score)
LR_Grp_F1_Score = f1_score(y_test_Grped, y_pred_grped, average='weighted')
print('Testing F1 score: {}'.format(LR_Grp_F1_Score))
print(classification_report(y_test_Grped, y_pred_grped))
```

2. Support Vector Classifier Model:

```
svc = Pipeline([('vect', CountVectorizer()),
                ('tfidf', TfidfTransformer()),
                ('clf', SVC()),
                ])
svc.fit(X_train_Grped, y_train_Grped)

y_pred_grped= svc.predict(X_test_Grped)
SVC_Grped_Accuracy_Score = accuracy_score(y_test_Grped, y_pred_grped)
print('accuracy %s' % SVC_Grped_Accuracy_Score)
SVC_Grped_F1_Score = f1_score(y_test_Grped, y_pred_grped, average='weighted')
print('Testing F1 score: {}'.format(SVC_Grped_F1_Score))
print(classification_report(y_test_Grped, y_pred_grped))
```

3. Decision tree Classifier Model:

```
#Model 3: Decision Tree Classifier
nb = Pipeline([('vect', CountVectorizer()),
                ('tfidf', TfidfTransformer()),
                ('clf', DecisionTreeClassifier()),
                ])
nb.fit(X_train_Grped, y_train_Grped)

y_pred_grped = nb.predict(X_test_Grped)
DT_Grped_Accuracy_Score = accuracy_score(y_test_Grped, y_pred_grped)
print('accuracy %s' % DT_Grped_Accuracy_Score)
DT_Grped_F1_Score = f1_score(y_test_Grped, y_pred_grped, average='weighted')
print('Testing F1 score: {}'.format(DT_Grped_F1_Score))
print(classification_report(y_test_Grped, y_pred_grped))
```

4. Random Forest Classifier Model:

```
#Model 4: Random Forest Classifier
nb = Pipeline([('vect', CountVectorizer()),
                ('tfidf', TfidfTransformer()),
                ('clf', RandomForestClassifier()),
                ])
nb.fit(X_train_Grped, y_train_Grped)

y_pred_grped = nb.predict(X_test_Grped)
RF_Grped_Accuracy_Score = accuracy_score(y_test_Grped, y_pred_grped)
print('accuracy %s' % RF_Grped_Accuracy_Score )
RF_Grped_F1_Score = f1_score(y_test_Grped, y_pred_grped, average='weighted')
print('Testing F1 score: {}'.format(RF_Grped_F1_Score))
print(classification_report(y_test_Grped, y_pred_grped))
```

5. Adaboost Classifier Model:

```
#Model 5: Adaboost Classifier

nb = Pipeline([('vect', CountVectorizer()),
                ('tfidf', TfidfTransformer()),
                ('clf', AdaBoostClassifier()),
                ])
nb.fit(X_train_Grped, y_train_Grped)

y_pred_grped = nb.predict(X_test_Grped)
ABC_Grped_Accuracy_Score = accuracy_score(y_test_Grped, y_pred_grped)
print('accuracy %s' % ABC_Grped_Accuracy_Score )
ABC_Grped_F1_Score = f1_score(y_test_Grped, y_pred_grped, average='weighted')
print('Testing F1 score: {}'.format(ABC_Grped_Accuracy_Score))
print(classification_report(y_test_Grped, y_pred_grped))
```


B. Trying different model architectures in Deep Learning.

We have also tried a couple of Deep learning/Neural network (state of the art) models by defining the model architecture, loss function, optimizers and metric. The models that we have used for the interim submission are:

1. Simple Sequential NLP
2. Simple LSTM with Word2Vec
3. Simple LSTM with GLOVE

1. Simple Sequential NLP Model:

In this model, we have used the word corpus that was created earlier for getting the vocab_size and also to create the embeddings. We have used **Convolution** Neural Network model with **pooling, dropouts, flattening** and finally we have used **softmax** activation function since this is a multiclass classification problem. This model has totally 1,156,783 trainable parameters.

```
[ ] # Define the Keras model
model = Sequential()
model.add(Embedding(vocab_size, embedding_size, input_length=maxlen))
model.add(Dropout(0.50))
model.add(Conv1D(filters=32, kernel_size=2, padding='same', activation='relu'))
model.add(Dropout(0.50))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dropout(0.50))
model.add(Dense(50, activation='softmax'))
model.compile(optimizer=Adam(lr = 0.01), loss='sparse_categorical_crossentropy', metrics=['acc'])
```

```
[ ] model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 100, 100)	1070300
dropout (Dropout)	(None, 100, 100)	0
conv1d (Conv1D)	(None, 100, 32)	6432
dropout_1 (Dropout)	(None, 100, 32)	0
max_pooling1d (MaxPooling1D)	(None, 50, 32)	0
flatten (Flatten)	(None, 1600)	0
dropout_2 (Dropout)	(None, 1600)	0
dense (Dense)	(None, 50)	80050
Total params: 1,156,782		
Trainable params: 1,156,782		
Non-trainable params: 0		

2. Simple LSTM Model with Word2Vec Embedding:

In this model, we have first created the Word2Vec embedding file using the word corpus generated using the final data and then used them to create the embedding matrix. This matrix was saved and later used as weights in the model definition. We have used a basic **LSTM model** with **flattening** and **dropouts** and also used a couple of activation functions like **relu** and **softmax**.

```
lstm_model = Sequential()
#Embedding layer
lstm_model.add(Embedding(vocab_size, embedding_size, weights=[embedding_matrix]))
lstm_model.add(LSTM(units=128))
lstm_model.add(Flatten())
lstm_model.add(Dropout(0.50))
lstm_model.add(Dense(50, activation='relu'))
lstm_model.add(Flatten())
lstm_model.add(Dropout(0.50))
lstm_model.add(Dense(50, activation='softmax'))
lstm_model.compile(optimizer=Adam(lr = 0.01), loss='sparse_categorical_crossentropy', metrics=['acc'])
```

```
lstm_model.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
=====		
embedding_1 (Embedding)	(None, None, 100)	1070300
=====		
lstm (LSTM)	(None, 128)	117248
=====		
flatten_1 (Flatten)	(None, 128)	0
=====		
dropout_3 (Dropout)	(None, 128)	0
=====		
dense_1 (Dense)	(None, 50)	6450
=====		
flatten_2 (Flatten)	(None, 50)	0
=====		
dropout_4 (Dropout)	(None, 50)	0
=====		
dense_2 (Dense)	(None, 50)	2550
=====		
Total params: 1,196,548		
Trainable params: 1,196,548		
Non-trainable params: 0		
=====		

3. Simple LSTM Model with Glove Embedding:

In this model, we have used Glove embedding instead of Word2Vec. We have used the glove file with 100 dimensions for this embedding. There are 1,196,548 trainable params for this model.

```
[ ] lstm_model_glove = Sequential()
#Embedding layer
lstm_model_glove.add(Embedding(vocab_size, embedding_size, weights=[embedding_matrix_glove]))
lstm_model_glove.add(LSTM(units=128))
lstm_model_glove.add(Flatten())
lstm_model_glove.add(Dropout(0.50))
lstm_model_glove.add(Dense(50, activation='relu'))
lstm_model_glove.add(Flatten())
lstm_model_glove.add(Dropout(0.50))
lstm_model_glove.add(Dense(50, activation='softmax'))
lstm_model_glove.compile(optimizer=Adam(lr = 0.01), loss='sparse_categorical_crossentropy', metrics=['acc'])
```



```
[ ] lstm_model_glove.summary()
```

Model: "sequential_3"

Layer (type)	Output Shape	Param #
embedding_3 (Embedding)	(None, None, 100)	1070300
lstm_2 (LSTM)	(None, 128)	117248
flatten_5 (Flatten)	(None, 128)	0
dropout_7 (Dropout)	(None, 128)	0
dense_5 (Dense)	(None, 50)	6450
flatten_6 (Flatten)	(None, 50)	0
dropout_8 (Dropout)	(None, 50)	0
dense_6 (Dense)	(None, 50)	2550
Total params: 1,196,548		
Trainable params: 1,196,548		
Non-trainable params: 0		

C. Train the model

All the models have been trained using the training data, validated with validation data and tested using Test data. The data is split into train, validation and test data in the ratio of 0.6:0.2:0.2. A sample of model training code for ML and Deep learning models are shown below:

Logistic Regression:

```
#fit the model with training data
lr.fit(X_train, y_train)
#predict the model for test data
y_pred = lr.predict(X_test)

#calculate scores and print classification report
LR_Accuracy_Score = accuracy_score(y_test,y_pred)
print('accuracy %s' % LR_Accuracy_Score)
LR_F1_Score = f1_score(y_test,y_pred, average='weighted')
print('Testing F1 score: {}'.format(LR_F1_Score))
print(classification_report(y_test, y_pred))
```

LSTM with Glove Embedding

```
Batch_size = 100
Epochs = 5

lstm_glove_Model_history = lstm_model_glove.fit(X_train, y_train, batch_size = Batch_size, validation_data = (x_val,y_val), epochs = Epochs)
```

Epoch 1/5
51/51 [=====] - 19s 378ms/step - loss: 2.5082 - acc: 0.4833 - val_loss: 2.0088 - val_acc: 0.5633
Epoch 2/5
51/51 [=====] - 19s 381ms/step - loss: 2.0187 - acc: 0.5378 - val_loss: 1.8553 - val_acc: 0.5824
Epoch 3/5
51/51 [=====] - 19s 379ms/step - loss: 1.7458 - acc: 0.5687 - val_loss: 1.8309 - val_acc: 0.5890
Epoch 4/5
51/51 [=====] - 19s 379ms/step - loss: 1.5524 - acc: 0.5894 - val_loss: 1.8096 - val_acc: 0.5938
Epoch 5/5
51/51 [=====] - 19s 379ms/step - loss: 1.3996 - acc: 0.6213 - val_loss: 1.9519 - val_acc: 0.5998

D. Save the Weights:

To deal with large training time, we have saved the embedding weights so that we can use them when training the model for the second time without starting from scratch. For both Word2Vec & Glove models of LSTM, the embeddings have been defined separately and re-used (weight matrix) where considerable time has been saved.

Word2Vec Embedding:

```
[ ] # load the whole embedding into memory
embeddings1 = dict()
f = open(project_path+'word2vec_vector.txt')


for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings1[word] = coefs
f.close()
print('Loaded %s word vectors.' % len(embeddings1))
```

 Loaded 10703 word vectors.

```
[ ] embedding_matrix = np.zeros((vocab_size, 100))

for word, i in tokenizer.word_index.items():
    embedding_vector = embeddings1.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector

len(embeddings1.values())
```

 10703

Glove Embedding:

```
[ ] EMBEDDING_FILE = project_path + 'glove.6B.100d.txt'

embeddings = {}
for o in open(EMBEDDING_FILE):
    word = o.split(" ")[0]
    # print(word)
    embd = o.split(" ")[1:]
    embd = np.asarray(embd, dtype='float32')
    # print(embd)
    embeddings[word] = embd
```

▼ Create Weight Matrix for GLOVE

```
[ ] embedding_matrix_glove = np.zeros((vocab_size, 100))

for word, i in tokenizer.word_index.items():
    embedding_vector = embeddings.get(word)
    if embedding_vector is not None:
        embedding_matrix_glove[i] = embedding_vector

len(embeddings.values())
```

 400000

4.2.3 MILESTONE 3: Test the Model, Fine-tuning and Repeat

A. Test the model and report as per evaluation metrics

All the 8 models have been trained using the training data and scores are calculated for the model based on the test data. Also, we have predicted the target values using the `model.predict()` function and calculated the accuracy scores by comparing the original versus predicted values.

The following metrics are used for traditional ML models:

- 1) Accuracy Score
- 2) F1 Score
- 3) Classification report (that includes Precision, recall and F1 scores)

A snapshot of the model testing and evaluation metrics is given below:

```
accuracy 0.6176821983273596
Testing F1 score: 0.5263098347000964
```

	precision	recall	f1-score	support
GRP_0	0.61	0.98	0.75	779
GRP_1	0.00	0.00	0.00	9
GRP_10	1.00	0.14	0.25	28
GRP_11	0.00	0.00	0.00	8
GRP_12	0.51	0.52	0.51	52
GRP_13	0.92	0.39	0.55	28
GRP_14	0.75	0.13	0.22	23
GRP_15	0.00	0.00	0.00	10
GRP_16	0.00	0.00	0.00	17
GRP_17	1.00	0.60	0.75	15
GRP_18	0.33	0.06	0.10	18
GRP_19	0.50	0.13	0.21	45

For Deep learning models, the following evaluation metrics are used:

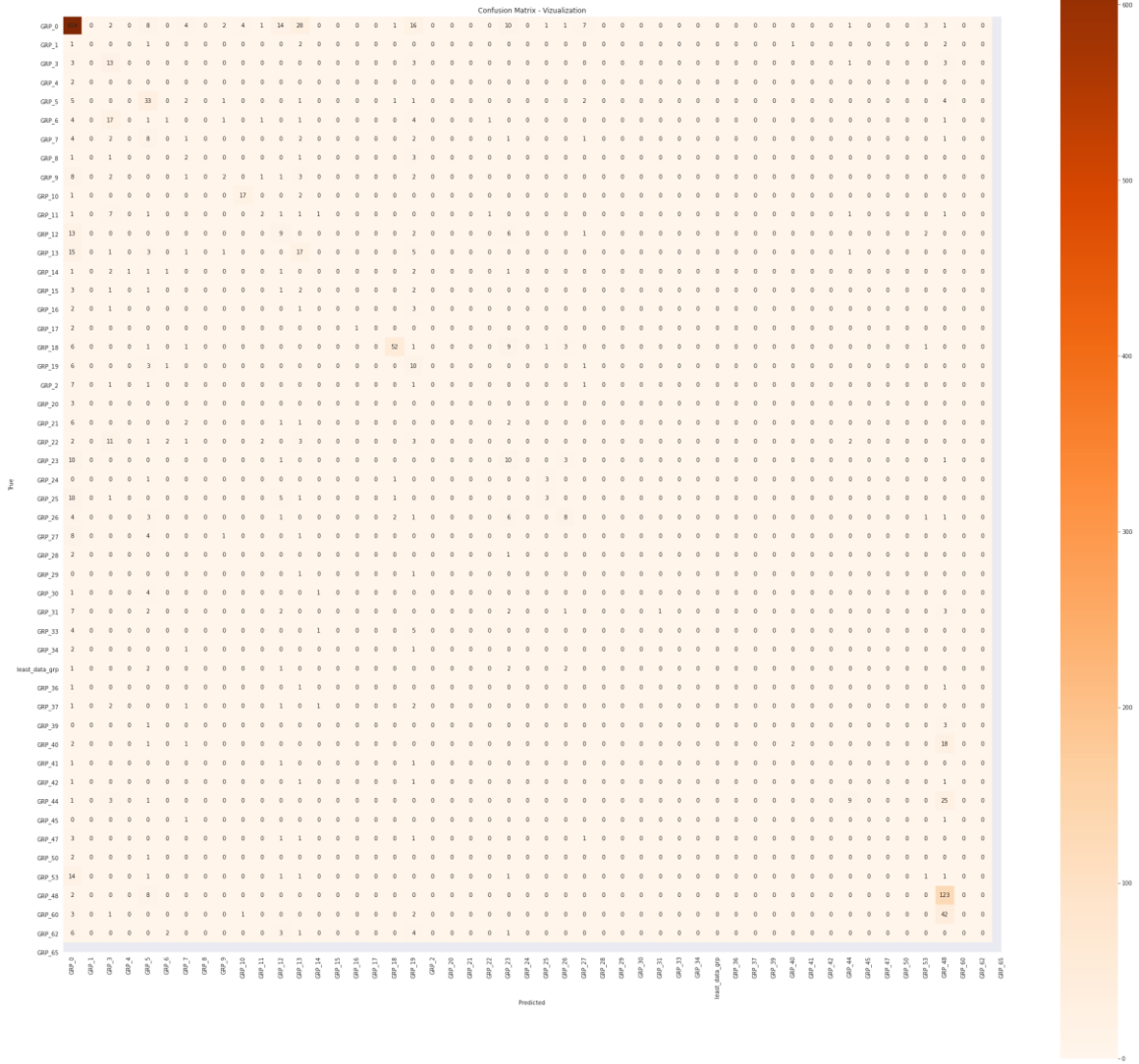
1. Confusion Matrix
2. F1 Score
3. Accuracy Score
4. Classification Report
5. Graph plot for Training Accuracy Vs Validation Accuracy
6. Graph plot for Training Loss Vs Validation Loss

A sample snapshot of the metrics used are shown below:

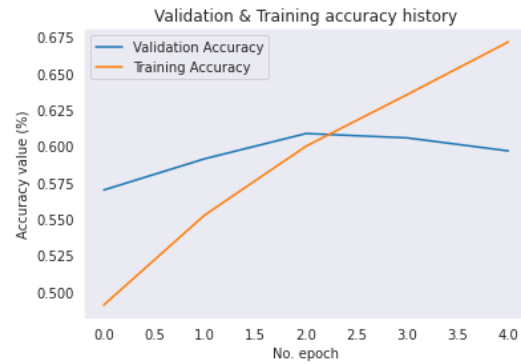
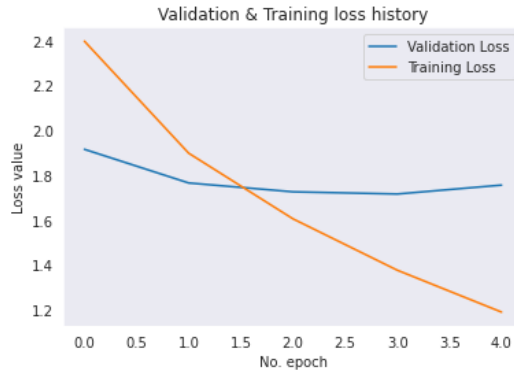
```
[ ] # Test the model after training
    test_results = lstm_model_glove.evaluate(x_test, y_test, verbose=False)
    print(f'Test results - Loss: {test_results[0]} - Accuracy: {100*test_results[1]}%')
```

➞ Test results - Loss: 1.8448033332824707 - Accuracy: 57.825565338134766%

Confusion Matrix:



	precision	recall	f1-score	support
0	0.78	0.86	0.82	758
1	0.00	0.00	0.00	7
2	0.19	0.57	0.29	23
3	0.00	0.00	0.00	2
4	0.36	0.66	0.46	50
5	0.14	0.03	0.05	32
6	0.05	0.05	0.05	22



B. Try different models

As discussed above, different models (traditional ML and deep learning) have been tried and all the models have been compared and presented in a table to identify the best performing model.

	Accuracy	F1 Score
LR Model	0.641577	0.550900
LR_Grped	0.617682	0.526310
SVC Model	0.642772	0.553294
SVC Grped	0.621266	0.527660
Decision Tree	0.603943	0.590088
Decision Tree Grped	0.571685	0.553171
Random Forest	0.658901	0.587384
Random Forest Grped	0.635006	0.564301
AdaBoost	0.520311	0.387019
AdaBoost Grped	0.500597	0.369240
Sequential NLP	0.579450	0.492030
LSTM with Word2Vec	0.535245	0.475101
LSTM with Glove	0.589606	0.526212

4.2.4 WORK DONE FOR FINAL SUBMISSION

After the Interim submission, we worked on the following to find the better accuracy and F1 score. The brief description of the same is provided in the following points.

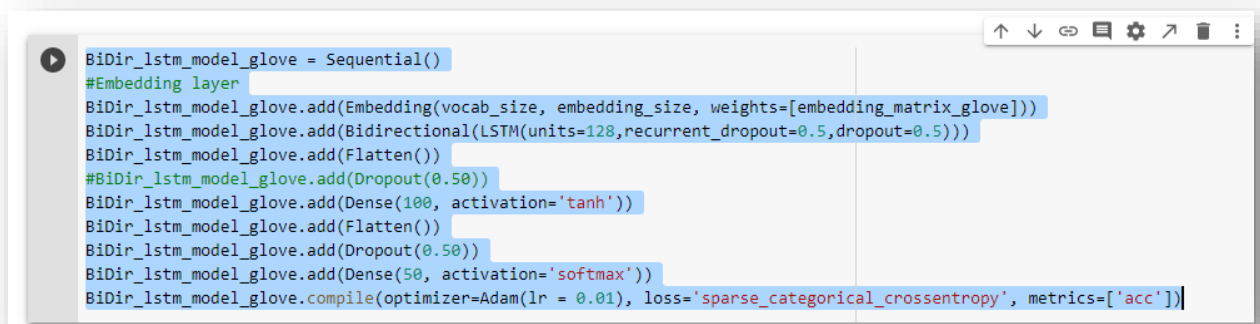
1. Bidirectional LSTM
2. Hyperparameters tuning
3. GRU model
4. RNN Model
5. State of the art Models (Bert, ULMFit)
6. Visual aids for Model Comparison

- **Bi-Directional LSTM model**

Bidirectional LSTM (Long Short-Term Memory) models are an extension of traditional LSTMs that can improve model performance on sequence classification problems. In problems where all time steps of the input sequence are available, Bidirectional LSTMs train two instead of one LSTMs on the input sequence. The first on the input sequence as-is and the second on a reversed copy of the input sequence. This can provide additional context to the network and result in faster and even fuller learning on the problem.

Following is the summary of the tasks performed

- Model Built for Bidirectional LSTM model with Glove as the embedding.
- Have used Dropouts and Flattening layers as well.
- Used Batch Size 100 and Epochs 5
- Activation functions Tanh and Softmax were used instead of relu that was used previously.
- The loss function used is Categorical Cross entropy with learning rate of 0.01 and accuracy as the metric



```

BiDir_lstm_model_glove = Sequential()
#Embedding layer
BiDir_lstm_model_glove.add(Embedding(vocab_size, embedding_size, weights=[embedding_matrix_glove]))
BiDir_lstm_model_glove.add(Bidirectional(LSTM(units=128, recurrent_dropout=0.5, dropout=0.5)))
BiDir_lstm_model_glove.add(Flatten())
#BiDir_lstm_model_glove.add(Dropout(0.50))
BiDir_lstm_model_glove.add(Dense(100, activation='tanh'))
BiDir_lstm_model_glove.add(Flatten())
BiDir_lstm_model_glove.add(Dropout(0.50))
BiDir_lstm_model_glove.add(Dense(50, activation='softmax'))
BiDir_lstm_model_glove.compile(optimizer=Adam(lr = 0.01), loss='sparse_categorical_crossentropy', metrics=['acc'])
  
```

Observations:

Bidirectional LSTM with GloVe is comparatively better than Simple LSTM, Word2Vec and Sequential NLP with respect to F1 score and accuracy and also the best performance model in Deep learning model.

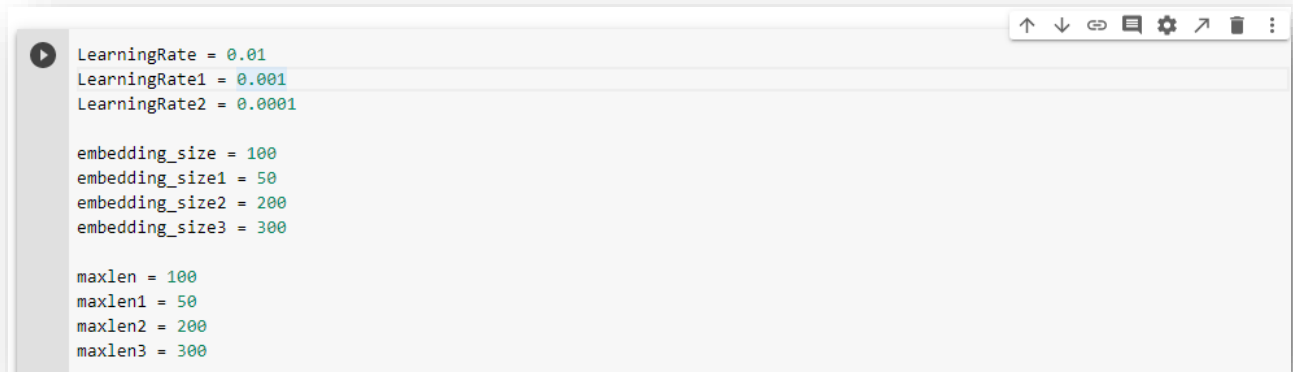
- **Hyper Parameter Tuning for Bidirectional LSTM**

Hyper parameter Tuning were performed to check if the performance of the model improves. Following are some of the hyper parameters tried for the performance tuning.

- Learning Rate
- Embedding size
- Maxlen
- LSTM units

- e. Multiple LSTM layers
- f. Return Sequence
- g. Epochs
- h. Using Callbacks
- i. Batch Size

Some of the learning rates, embedding size and length used are shown below.



```
LearningRate = 0.01
LearningRate1 = 0.001
LearningRate2 = 0.0001

embedding_size = 100
embedding_size1 = 50
embedding_size2 = 200
embedding_size3 = 300

maxlen = 100
maxlen1 = 50
maxlen2 = 200
maxlen3 = 300
```

There were 7 models tried with different hyperparameter values. They are:

1. Bidirectional LSTM Model with GLOVE Embedding - 50 Dimensions
2. Bidirectional LSTM Model with GLOVE Embedding - 200 Dimensions
3. Bidirectional LSTM Model with GLOVE Embedding - 300 Dimensions
4. Bidirectional LSTM Model with different learning rate and callbacks
5. Bidirectional LSTM Model with different LSTM Units
6. Bidirectional LSTM Model with multiple LSTM Layers
7. Base LSTM Model with Return sequences and trainable parameters set to True

Results:

BiDirectional LSTM with Glove	0.605735	0.578127
BiDirectional LSTM with 50 Dimensions	0.587216	0.523063
Bidirectional LSTM with 200 Dimensions	0.567503	0.470190
BiDirectional LSTM with 300 Dimensions	0.587814	0.490047
BiDirectional LSTM with LR 0.0001	0.516129	0.360143
BiDirectional LSTM with 256 units	0.541816	0.398733
Multiple LSTM	0.529869	0.457035
LSTM with Return Sequences	0.583035	0.516597

Observations:

- LSTM with return sequences have performed well compared to the other LSTM models and also the second best performing model after the base Bidirectional LSTM model with Glove Embedding.
- Even though we tried tuning the model with multiple hyper parameter values, the model performance did not improve drastically from the base model. This shows the skewness/imbalance in the data.
- The given data is highly imbalanced with having more records in GRP_0 and this affects the model performance and accuracy.

• RNN Model

Recurrent neural networks (RNN) are the state of the art algorithm for sequential data. It is the first algorithm that remembers its input, due to an internal memory, which makes it perfectly suited for machine learning problems that involve sequential data. In this model, we used Bidirectional LSTM, Convolution 1D, MaxPooling 1D, Dropout and Dense layers with Activation functions tanh and softmax. We ran the model with batch size of 100 and 5 Epochs.

```
[ ] RNN_model = Sequential()
    #Embedding layer
    RNN_model.add(Embedding(vocab_size, embedding_size, input_length=maxlen, weights=[embedding_matrix_glove], trainable=True))
    RNN_model.add(Conv1D(100,10,activation='tanh'))
    RNN_model.add(MaxPooling1D(pool_size=2))
    RNN_model.add(Dropout(0.3))
    RNN_model.add(Conv1D(100,10,activation='tanh'))
    RNN_model.add(MaxPooling1D(pool_size=2))
    RNN_model.add(Bidirectional(LSTM(units=128)))
    RNN_model.add(Dropout(0.3))
    RNN_model.add(Dense(100, activation='tanh'))
    RNN_model.add(Dense(50, activation='softmax'))
    RNN_model.compile(optimizer=Adam(lr = 0.01), loss='sparse_categorical_crossentropy', metrics=['acc'])
```


On executing, RNN Model has the lowest accuracy score and F1 Score of all the deep learning models.

RNN Model	0.505376	0.384452
------------------	----------	----------

- **GRU Model**

This GRU model (Gated Recurrent Unit) is a different variation of LSTM that has update gate and reset gate. These gates decide what should be passed to the output. This keeps the relevant information and passes it to the next steps. In this GRU model,

- We have used the Glove embedding with trainable parameter set to Tune
- We have used GRU with 128 units followed by Tanh and Softmax activation functions.

```
[ ] GRU_model = Sequential()
    #Embedding layer
    GRU_model.add(Embedding(vocab_size, embedding_size, input_length=maxlen, weights=[embedding_matrix_glove], trainable=True))
    GRU_model.add(GRU(units=128))
    GRU_model.add(Dropout(0.3))
    GRU_model.add(Dense(100, activation='tanh'))
    GRU_model.add(Dense(50, activation='softmax'))
    GRU_model.compile(optimizer=Adam(lr = 0.01), loss='sparse_categorical_crossentropy', metrics=['acc'])
```

GRU Model has performed better than all the LSTM models with accuracy at 62% and F1 score at 60%. This is the best model so far with respect to accuracy and F1 score when compared to all the models.

GRU Model	0.622461	0.601732
------------------	----------	----------

- **State of the Art Models**

In this section we explored the following 2 Models

- ULMFit (**U**niversal **L**anguage **M**odel **F**ine **T**uning)
- BERT (**B**idirectional **E**ncoder **R**epresentations from **T**ransformers)

ULMFit:

Universal Language Model Fine-Tuning (ULMFiT) is a transfer learning technique which can help in various NLP tasks. ULMFiT involves 3 major stages:

1. LM pre-training
2. LM fine-tuning
3. Classifier fine-tuning

The method is universal:

1. It works across tasks varying in document size, number, and label type.
2. It uses a single architecture and training process.
3. It requires no custom feature engineering or pre-processing.
4. It does not require additional in-domain documents or labels.

- **Language Modeling Pretraining:**

```
[ ] # Language model data : We use test_df as validation for language model
data_lm = TextLMDataBunch.from_df(path = "", train_df= train_df , valid_df = valid_df, test_df=test_df, text_cols='Description', label_cols='label')
data_lm.show_batch()
```

idx	text
0	co team view instead getting authorized message recreate condition nsu attach result ticketing tool ticket provide xxbos setup computer repair information xxbos ipaddress xxbos support roboworker xxunk xxbos wifi issue getting connect xxbos account locked xxbos vogelfontein south africa sa company eu zaf access sw since pm et interface gigabitethernet uplink core com xxbos release device please quarantine microsoft location ipaddress mobile temporarily blocked synchronizing using exchange activesync administrator usas
1	firewall blocking legitimate server application also compromised reac ng malicious propagating worm code escalating incident via medium priority ticket email per default event handling procedures would like us handle incidents differently future see options questions concerns please let know either corresponding delegating back soc calling full escalation windows login failure explicit notification phone automatically resolve directly portal events available reporting purposes xxbos user new personal device activation without approval form
2	xxbos firmware upgrade phones xxunk called issue supposed mentioned still rebooting loop xxbos atp getting committed sid please note screen shot error msg given xxunk b xxbos please provide permission files department networking xxbos urgent access disconnected could please help aft location xxbos job hr payroll na failed i d ipaddress xxbos network outage south amerirtca rrc company sa dmvpn router since backup circuit type power please specify top cert

```
#train the model with model learner and LSTM algorithm
learn = language_model_learner(data_lm, AWD_LSTM, drop_mult=0.3)
#fit the trained model
learn.fit_one_cycle(1, 1e-2)
```

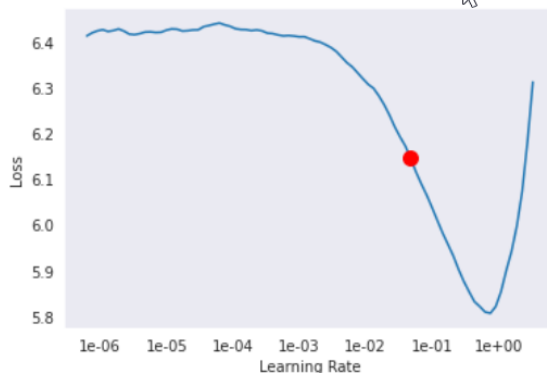
Downloading <https://s3.amazonaws.com/fast-ai-modelzoo/wt103-fwd.tgz>

epoch	train_loss	valid_loss	accuracy	time
0	7.302647	6.244890	0.117894	00:02

- **Fine Tuning:**

```
#plot the findings
learn.recorder.plot(suggestion=True)
min_lr = learn.recorder.min_grad_lr
print(min_lr)
```

Min numerical gradient: 4.79E-02
Min loss divided by 10: 7.59E-02
0.0478630092322638



```
learn.fit_one_cycle(2, min_lr)
```

epoch	train_loss	valid_loss	accuracy	time
0	5.572680	4.792077	0.282031	00:02
1	4.870867	4.577581	0.313876	00:02

- **Classifier Model**

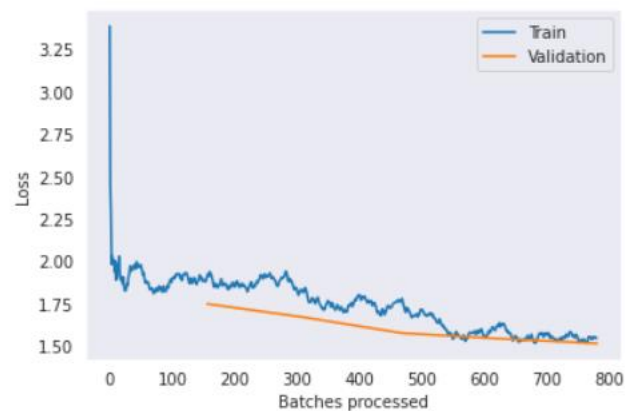
```
print("Creating Classifier Object")
claslearn = text_classifier_learner(data_clas, AWD_LSTM, drop_mult=0.5)
claslearn.load_encoder('fine_tuned_enc')
```

When all layers were frozen, the accuracy improved from 32% to 55%

epoch	train_loss	valid_loss	accuracy	time
0	2.611714	2.083801	0.530466	00:04
1	2.172193	1.828868	0.556750	00:04

When last 2 layers were unfreezed, the accuracy improved from 55% to 61%

epoch	train_loss	valid_loss	accuracy	time
0	1.896863	1.748095	0.571087	00:04
1	1.819044	1.668725	0.577658	00:05
2	1.770027	1.576215	0.590800	00:05
3	1.623517	1.543369	0.614098	00:04
4	1.544952	1.515405	0.618280	00:04



When all the layers were unfreezed, the accuracy improved from 61% to 63%

epoch	train_loss	valid_loss	accuracy	time
0	1.420484	2.061326	0.606332	00:08
1	1.445919	1.476614	0.623656	00:08
2	1.317027	1.458153	0.635603	00:08
3	1.266327	2.314915	0.626643	00:08
4	1.163098	1.429094	0.639188	00:08

ULMFit Model	0.646953	0.596049
---------------------	----------	----------

Observation:

1. The model performed well with 63% accuracy and 60% F1 Score and this is the highest performing model of all.

BERT:

BERT stands for Bidirectional Encoder Representations from Transformers. It is designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context. As a result, the pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of NLP tasks.

In this model, we have defined a class for Bert layer, that takes the training, validation and test dataframes as input, creates masks, segments and ids, initialize a bert module and creates the sequence output based on the output representation.

We have also defined a class for tokenization that tokenizes and pads the input and then builds the model. We also trained the weights and then trained the model. We have used Keras libraries and Tensorflow 1.x for Bert model alone, since BERT module is not fully compatible yet with tensorflow 2.x.

```

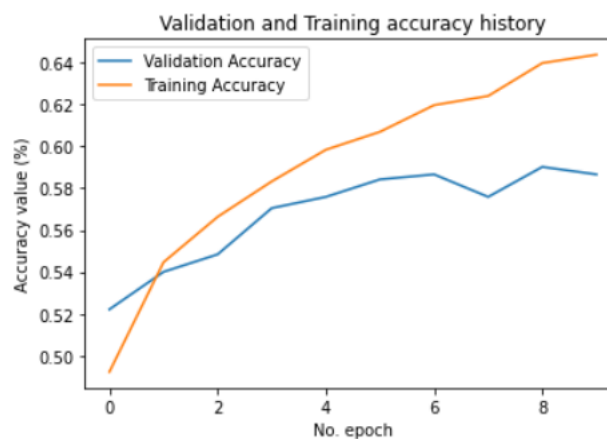
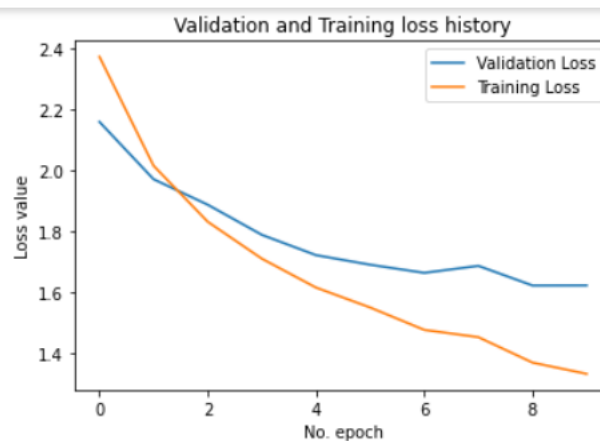
# function to build the bert model using class BertLayer
def build_model(max_seq_length, tf_hub, n_classes, n_fine_tune):
    in_id = keras.layers.Input(shape=(max_seq_length,), name="input_ids")
    in_mask = keras.layers.Input(shape=(max_seq_length,), name="input_masks")
    in_segment = keras.layers.Input(shape=(max_seq_length,), name="segment_ids")
    bert_inputs = [in_id, in_mask, in_segment]

    bert_output = BertLayer(n_fine_tune_layers=n_fine_tune, tf_hub = tf_hub, output_representation = 'mean_pooling', trainable = True)(bert_inputs)
    drop = keras.layers.Dropout(0.3)(bert_output)
    dense = keras.layers.Dense(128, activation='tanh')(drop)
    drop = keras.layers.Dropout(0.3)(dense)
    dense = keras.layers.Dense(64, activation='tanh')(drop)
    pred = keras.layers.Dense(n_classes, activation='softmax')(dense)

    model = keras.models.Model(inputs=bert_inputs, outputs=pred)
    Adam = keras.optimizers.Adam(lr = 0.001)
    model.compile(loss='sparse_categorical_crossentropy', optimizer=Adam, metrics=['sparse_categorical_accuracy'])
    model.summary()

    return model

```



BERT Model 0.597372 0.528474

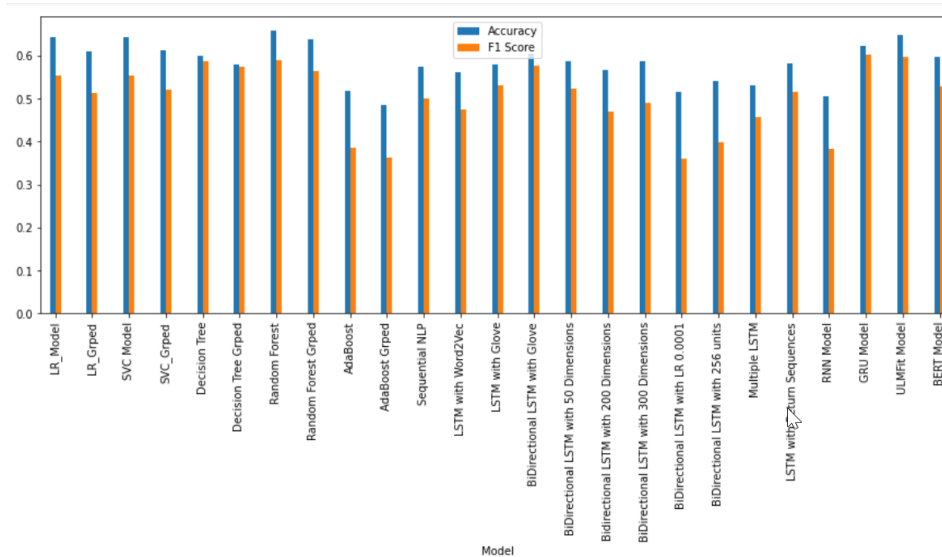
Observations:

BERT model has an accuracy of 59% and F1 Score of 52%. But this is not the best performing model compared to other models seen above.

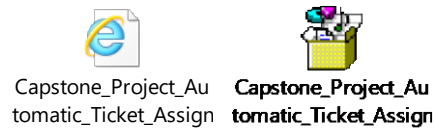
4.3 BENCH MARKING – Comparison of Approaches Taken

The parameters considered for comparison and bench marking are Accuracy and F1 Score.

	Model	Accuracy	F1 Score
0	LR_Model	0.642772	0.553273
1	LR_Grped	0.611111	0.514081
2	SVC Model	0.643369	0.553122
3	SVC_Grped	0.613501	0.519972
4	Decision Tree	0.599761	0.586772
5	Decision Tree Grped	0.580645	0.573504
6	Random Forest	0.659498	0.589122
7	Random Forest Grped	0.636798	0.564951
8	AdaBoost	0.517324	0.386926
9	AdaBoost Grped	0.486260	0.361604
10	Sequential NLP	0.574074	0.500027
11	LSTM with Word2Vec	0.560932	0.476163
12	LSTM with Glove	0.578256	0.531280
13	BiDirectional LSTM with Glove	0.605735	0.578127
14	BiDirectional LSTM with 50 Dimensions	0.587216	0.523063
15	BiDirectional LSTM with 200 Dimensions	0.567503	0.470190
16	BiDirectional LSTM with 300 Dimensions	0.587814	0.490047
17	BiDirectional LSTM with LR 0.0001	0.516129	0.360143
18	BiDirectional LSTM with 256 units	0.541816	0.398733
19	Multiple LSTM	0.529869	0.457035
20	LSTM with Return Sequences	0.583035	0.516597
21	RNN Model	0.505376	0.384452
22	GRU Model	0.622461	0.601732
23	ULMFIt Model	0.646953	0.596049
24	BERT Model	0.597372	0.528474



4.4 Final Code



4.5 Challenges

1. Handling of Skewed Data:

The given data-set is heavily skewed based on the tickets assigned to the groups. Hence we had different approaches among us.

Approach 1 – Dropping the groups which has less tickets assigned into it and to model for the data or Combining the groups which have least tickets assigned to form a group and consider that one among the group, hence we are not dropping even one value and not taking the risk of losing even a single value.

Approach 2 – Include every group information for modelling and not dropping or combining any of them.

Finally, to address an individual person's concern, we neither dropped nor used the group information as is, we decided to combine the least value group and treat it as one group.

2. Handling of Junk Values:

The data-set had to be cleansed and handled for numerous type of junk value and at the same time we have some essential values which might look like junk value, so most of the project duration had gone for the finalising the approach to derive the functions we had to cleanse the data and to retain essential information for classification

3. Handling of other languages:

The given dataset had the ticket description in both English and German languages. So we had to convert the ticket descriptions in German to English before proceeding. For this we used the google translator web-service.

4. Tensorflow compatibility for BERT model

The BERT model we build used tensorflow 1.x version where as the default version of tensorflow in google colab is 2.3 and all the other deep learning models including LSTM were built using Tensorflow version 2.x. So there were challenges in integrating all the models together and we had to switch the tensorflow version to 1.x while running the BERT model.

4.6 Conclusion

- The given problem involved classifying the ticket assignments based on the description and short description columns.
- The data pre-processing was done by removing all the junk characters, translating to english text, removing stop words, tokenization, and Lemmatization.

- Then **bi-grams, tri-grams model and word clouds** were built to understand the mapping between language and the ticket groups.
- Different Machine learning Models such as Logistic Regression, SVC, Decision Tree, Random Forest and Ada Boost Classifier with TFIDF vectorization were built and executed.
- **Random Forest and Decision Tree Models** performed well compared to other Machine learning models.
- The metrics mainly used for model comparison and evaluation were **Accuracy Score, F1 Score. Classification Report and confusion matrix** were also generated for all the models.
- Different deep learning algorithms such as Sequential NLP, Simple LSTM, Bidirectional LSTM, RNN and GRU models were also built.
- The Embeddings used were **Word2Vec and GLOVE**. Since GLOVE embeddings performed better compared to Word2Vec, GLOVE was predominantly used in most of the models.
- **Hyper parameter tuning** was also done for different parameters such as maxlen, embedding size, epochs, batch size, learning rate, etc
- **GRU Model** is the best model in deep learning algorithms followed by **Bidirectional LSTM with Glove**.
- Also, state of the art models such as **ULMFit and BERT** were also built and executed.
- All these algorithms were run on google colabs and the code was developed on **Tensorflow Keras** libraries.
- The test results of all these algorithms with their accuracy and F1 scores have been added to a separate dataframe and provided in the notebook. A **comparison of all the Models** were also done.
- The accuracy and F1 score of all the models are less than 65%. This is because the given data is highly skewed with GRP_0 data. **Accuracy, F1 Score < 65%**
- The Top 3 Performing models were:
 1. **ULMFit**
 2. **GRU**
 3. **BiDirectional LSTM with Glove**

4.7 Future work

1. Data Sampling:

More data about other assignment groups need to be collected and the data needs to be sampled inorder for the algorithms to improve upon the accuracy scores.

2. Two Model Approach for Grp 0 and rest of the groups:

Another approach to tackle the skewness of data is by first running a binary classifier and identify whether the ticket belongs to GRP_0 or not. Once done, another multi-class classifier can be developed to run the model to classify the ticket in rest of the assignment groups.

3. ML Pipeline to automate the Model Buiding:

Further a ML pipeline can also be developed using apache airflow to automate the model building process so that the models can be continuously evaluated for future data.

REFERENCES

Pandas Data Frame is two-dimensional size-mutable, potentially heterogeneous tabular data structure with labelled axes (rows and columns). A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns. Pandas DataFrame consists of three principal components, the **data**, **rows**, and **columns**.

Word Cloud is a data visualization technique used for representing text data in which the size of each word indicates its frequency or importance. Significant textual data points can be highlighted using a word cloud. Word clouds are widely used for analyzing data from social network websites.

<https://stackabuse.com/text-classification-with-bert-tokenizer-and-tf-2-0-in-python/>

<https://www.kaggle.com/c/tweet-sentiment-extraction/discussion/143281>

<https://www.kaggle.com/mlwhiz/ulmfit>

<https://www.kaggle.com/igetii/bert-keras>