Great Learning Capstone Project - AIML Online Batch 2020-21

Automatic Ticket Assignment- Final Report

Prepared By:
Kalpesh Mulye, Vishw Teja,
Bharat Pant, Prashant Parhad, Madhup Dwivedi

Supervised by:
Rohit Raj

1. Summary of problem statement, data and findings

Project Overview:

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.

In the support process, incoming incidents are analyzed and assessed by organization support teams to fulfill the request. In many organizations, better allocation and effective usage of the valuable support resources will directly result in substantial cost savings.

In this capstone project, using a powerful AI / ML technique we will build a classifier that can by analysing text in the incidents and classify incidents to right functional groups can help organizations to reduce the resolving time of the issue and can focus on more productive tasks.

Problem Statement:

In most of the IT organizations, the assignment of incidents to appropriate IT groups is still a manual process. 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.

2. Overview of the final process

Process:

Currently, the incidents are created by various stakeholders (Business Users, IT Users and Monitoring Tools) within IT Service Management Tool and are assigned to Service Desk teams (L1 / L2 teams). This team will review the incidents for right ticket categorization, priorities and then carry out initial diagnosis to see if they can resolve. Around ~54% of the incidents are resolved by L1 / L2 teams. Incase L1 / L2 is unable to resolve, they will then escalate / assign the tickets to Functional teams from Applications and Infrastructure (L3 teams). Some portions of incidents are directly assigned to L3 teams by monitoring tools or Callers / Requestors. L3 teams will carry

out detailed diagnosis and resolve the incidents. Around ~56% of incidents are resolved by Functional / L3 teams. Incase if vendor support is needed, they will reach out for their support towards incident closure.
L1 / L2 needs to spend time reviewing Standard Operating Procedures (SOPs) before Assigning to Functional teams (Minimum ~25-30% of incidents needs to be reviewed for SOPs before ticket assignment). 15 min is being spent for SOP review for each incident. Minimum of ~1 FTE effort needed only for incident assignment to L3 teams.
During the process of incident assignments by L1 / L2 teams to functional groups, there were multiple instances of incidents getting assigned to wrong functional groups. Around ~25% of Incidents are wrongly assigned to functional teams. Additional effort needed for Functional teams to re-assign to right functional groups. During this process, some of the incidents are in queue and not addressed timely resulting in poor customer service.

Solution

This capstone project intends to reduce the manual intervention of IT operations or Service desk teams by automating the ticket assignment process.

The goal here is to create a text classification based ML model that can automatically classify any new tickets by analysing ticket description to one of the relevant Assignment groups, which could be later integrated to any ITSM tool like Service Now based on the ticket description our model will output the probability of assigning it to one of the 74 Groups.

Assumptions

In the AS-IS process it's mentioned that around ~54% of the incidents are resolved by L1 / L2 teams and the rest will be resolved as L2. So the assumption is that GRP_0 and GRP_8 which contribute 54% of the tickets are related to L1/L2 teams and the rest of the tickets belongs to L3 teams

Since the dataset is very imbalanced, we will be considering a subset of groups for predictions. In 74 groups, 46% of tickets belong to group 1 and 16 groups just have more than 100 tickets, rest of the Assignment groups have very less ticket counts which might not add much value to the model prediction. If we conducted random sampling towards all the subcategories, then we would face a problem that we might miss all the tickets in some categories. Hence, we will be only considering the groups that have more than 100 tickets. Rest of the tickets would be ignored.

Approach

The solution is been implemented using below approach:
Approach 1 - Using a traditional machine learning algorithm we are classifying the tickets into one of the groups having more than 100 tickets.

Approach 2 - I t's mentioned that around ~54% of the incidents are resolved by L1 / L2 teams and the rest will be resolved as L3. So the assumption is that GRP_0 and GRP_8 which contribute 54% of the tickets are related to L1/L2 teams and the rest of the tickets belong to L3 teams. In this approach, firstly the ticket would be classified into one of L1/L2 or L3 classes and then it would be further classified into one of the given assignment groups belonging to L1/L2 or L3 teams respectively. In this approach, we have considered assignment groups having more than 50 tickets.

Data approach

Understanding the structure of data:

The data files used for this capstone project are available at below google drive location:
https://drive.google.com/file/d/1OZNJm81JXucV3HmZroMq6qCT2m7ez7IJ

❖ The data set contains 4 columns and all are string columns

Column	Description	Data type
Short description	Short description on the problem for which incident is being raised	8492 non-null object
Description	Detailed description of the problem	8499 non-null object
Caller	Email id of the User who raised the problem	8500 non-null object
Assignment Group	IT Support Group to which the Incident log is been assigned to	8500 non-null object

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

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

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

- There are totally 8500 row.

- There seems to be missing values in Short description and Description columns, which needs to be looked into and handled.
- There are **8 null/missing values** present in the Short description and **1 null/missing values** present in the description column.

```
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
```

- Caller columns mainly contain the details of the user who raised the incident and is of not much use in our analysis and can be dropped.

- "Short Description" and "Description" can be concatenated as a single column, so that we won't miss any necessary info about the ticket.

```
In [8]: # Combining all the Rows
tickets_df['Cleaned'] = ''
for i in range(tickets_df.shape[0]):
    if tickets_df['Description'][i] == tickets_df['Short description'][i]:
        tickets_df.loc[i,'Cleaned'] = tickets_df['Description'][i]
    else:
        tickets_df.loc[i,'Cleaned'] = tickets_df['Short description'][i] +' '+ tickets_df['Description'][i]
```

- Assignment group is our predictor / target column with multiple classes. This is a **Multiclass Classification problem**

```
df_incidents['Assignment group'].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', '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'],
      dtype=object)
```

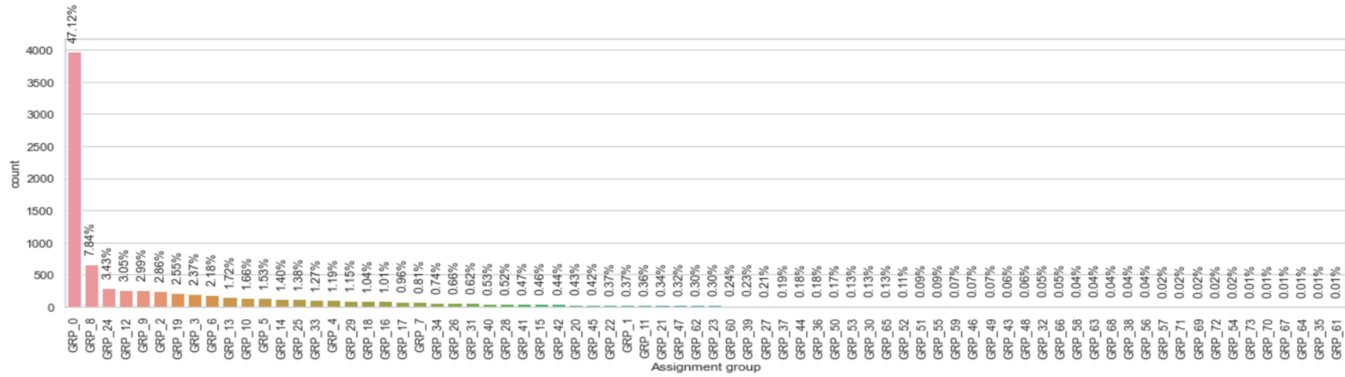
3. EDA

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

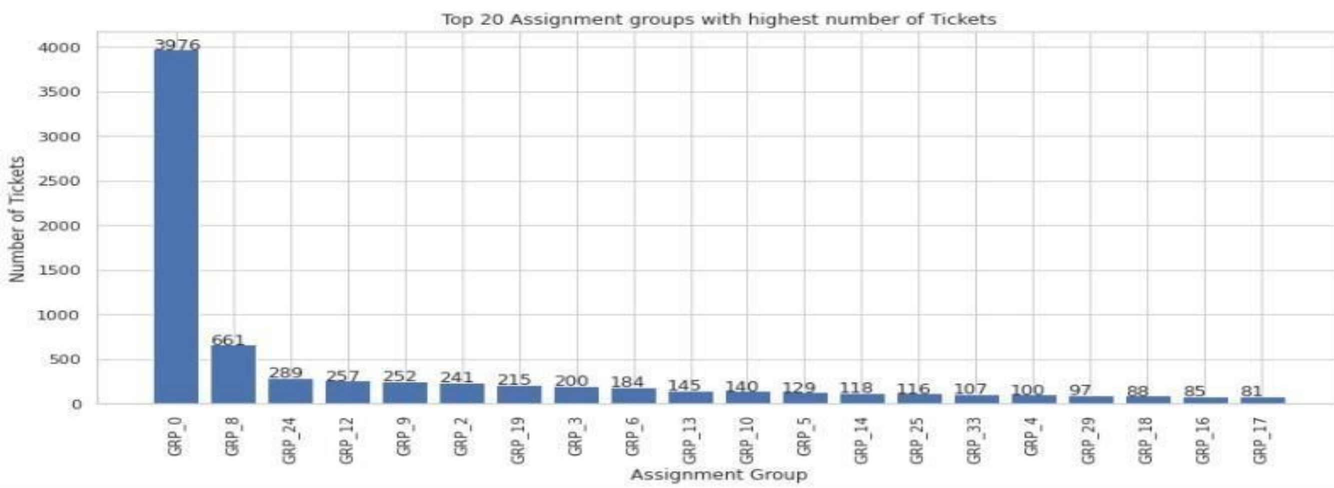
- Maximize Insight Into A Data Set;
- Uncover Underlying Structure;
- Extract Important Variables;
- Detect Outliers And Anomalies;
- Test Underlying Assumptions;
- Determine Optimal Factor Setting

Visually representing the content of a text document is one of the most important tasks in the field of text mining.

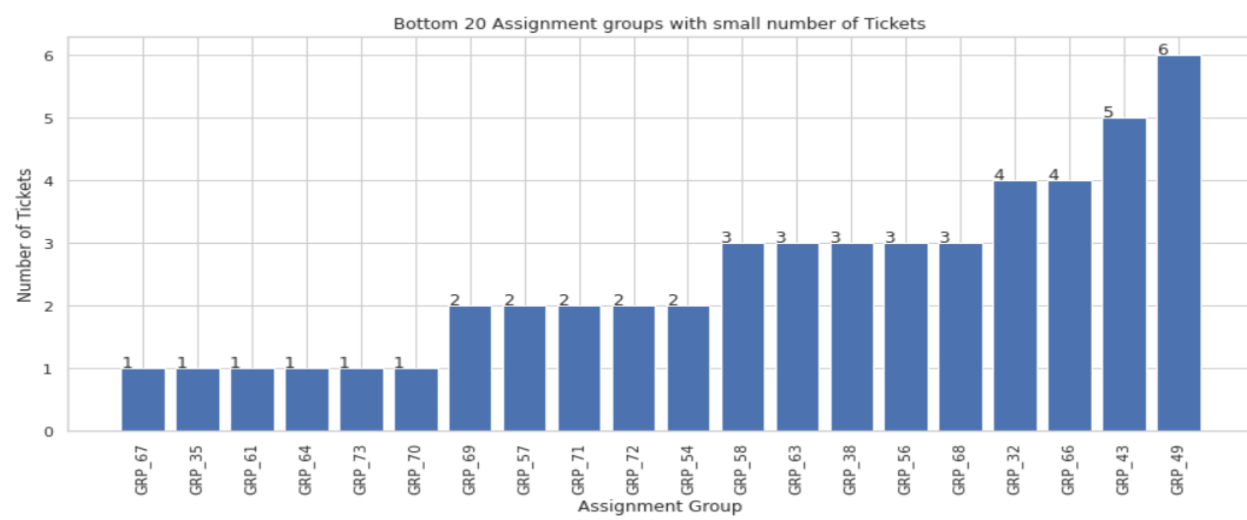
Class Percentage Distribution:



Top 20 Assignment Groups Distribution:

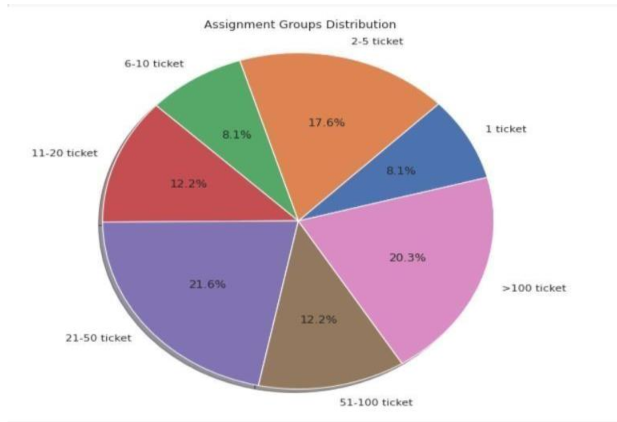


Bottom 20 Assignment Groups Distribution:



Ticket Distribution in Groups:

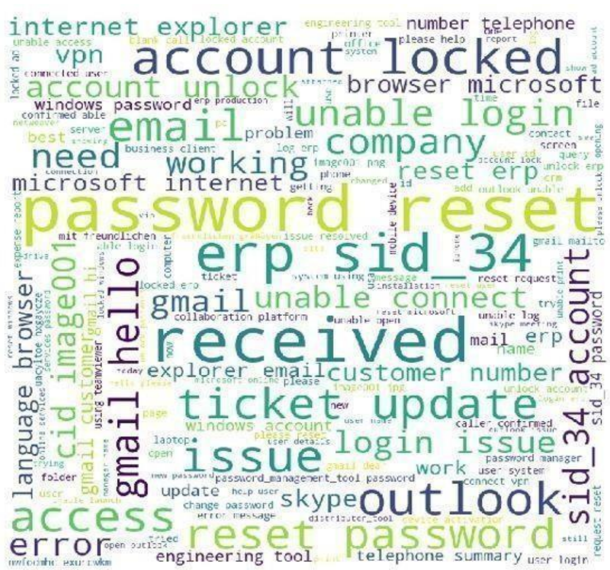
	Description	No of Categories	Ticket Count
0	1 ticket	NaN	6.0
1	2-5 ticket	NaN	13.0
2	6-10 ticket	NaN	6.0
3	11-20 ticket	NaN	9.0
4	21-50 ticket	NaN	16.0
5	51-100 ticket	NaN	9.0
6	>100 ticket	NaN	15.0



- We see that there are 6 Assignment Group’s for which just have 1 ticket in the dataset
- 15 Assignment groups have more than 100 tickets.
- Only20% of the Assignmentgroups have greater than 100 tickets.

WordCloud

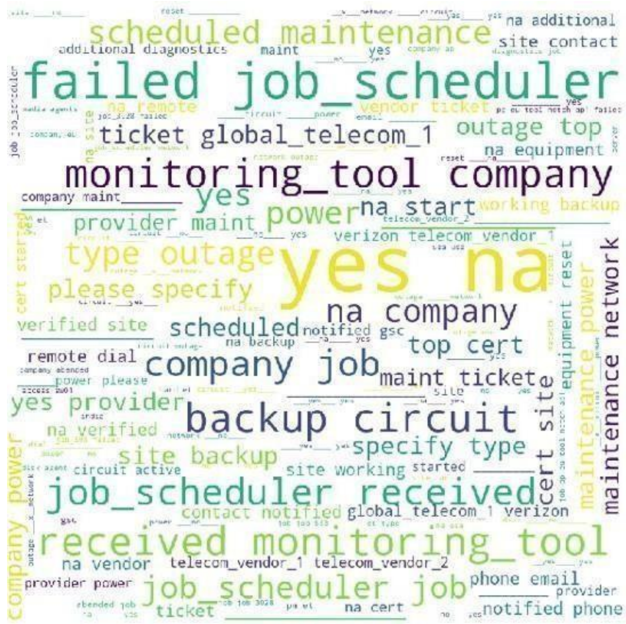
<u>Word Cloud for tickets with Assignment group 'GRP_0'</u>



- Analysis of GRP_0 which is the most frequent group to assign a ticket to, reveals that this group deals with mostly the maintenance problems such as password reset, account lock, login issue, ticket update etc.

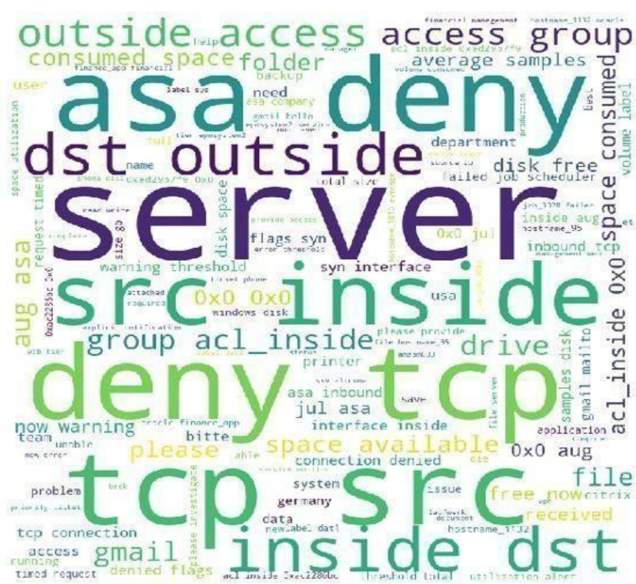
- The maximum of the tickets from GRP_0 can be reduced by self- correcting itself by putting automation scripts/mechanisms to help resolve these common maintenance issues. This will help in lowering the inflow of service tickets thereby saving the person/hour efforts spent and increasing the business revenue

Word Cloud for tickets with Assignment group 'GRP_8'



- GRP_8 seems to have tickets related to the outage, job failures, monitoring and maintenance.
- Anything regarding the electrical failure is been assigned to the GRP_8

Word Cloud for tickets with Assignment group 'GRP_12'



- GRP_12 contains tickets related to systems like disk space issues, network issues like tie out, Citrix issue, connectivity timeout, etc.
- It's indicative from the n-gram analysis and the word cloud is that the entire dataset speaks more about issues around

Word Cloud for tickets with Assignment group 'GRP_24'

- GRP_2 - Tickets are mainly in German (Deustch) these tickets need to be translated to English before passing them to our model.

Missing Values

	Short description	Description	Assignment group
2604	NaN	\r\n\r\nreceived from: ohdrnswl.rezuibdt@gmail...	GRP_34
3383	NaN	\r\n-connected to the user system using teamvi...	GRP_0
3906	NaN	-user unable tologin to vpn.\r\n-connected to...	GRP_0
3910	NaN	-user unable tologin to vpn.\r\n-connected to...	GRP_0
3915	NaN	-user unable tologin to vpn.\r\n-connected to...	GRP_0
3921	NaN	-user unable tologin to vpn.\r\n-connected to...	GRP_0
3924	NaN	name:wvqgbdhm fwchqjor\nlanguage:\nbrowser:mic...	GRP_0
4341	NaN	\r\n\r\nreceived from: eqmuniov.ehxkcbgj@gmail...	GRP_0

	Short description	Description	Assignment group
4395	i am locked out of skype	NaN	GRP_0

IMPUTATION

- We have various ways of treating the NULL/Missing values in the dataset such as
  - Replacing them with the empty string
  - Replacing them with some default values
  - Duplicating the Short description and Description values wherever one of them is Null
  - Dropping the records with null/missing values completely. We're not choosing to drop any record as we don't want to lose any information. And as we're going to concatenate the Short description and Description columns for each record while feeding them into NLP, we neither want to pollute the data by introducing any default values nor bias it by duplicating the description columns.

- Hence our NULL/Missing value treatment replaces the NaN cells with just an empty string

```
tickets_df['Description'].fillna(' ',inplace=True)
tickets_df['Short description'].fillna(' ',inplace=True)
```

Data Cleaning

Before we start with any text analytics we need to pre-process the data to get it all in a consistent format. We need to clean, tokenize and convert our data into a matrix. Some of the basic text pre-processing techniques include:

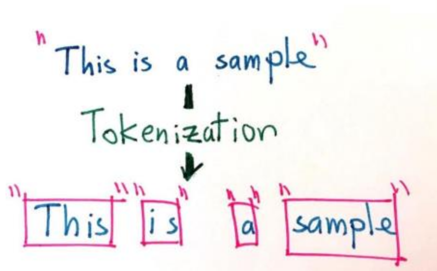
- Translation: A small number of tickets were written in German. Hence, weused the Google translate python API to convert German to English to generate the input data for the next steps. However, the google translator API can only process a limited number of texts daily, so we translated the text in batches and saved the file for further processing.
- Make text all **lowercase** so that the algorithm does not treat the samewords in different cases as different
- **Removing Noise** i.e everything that isn’t in a standard number or letter i.e Punctuation, Numerical values
- Removing extract spaces
- Removed punctuations
- Removed words containing numbers

```
In [6]: from dateutil import parser
def is_date(str_):
    try:
        parser.parse(str_)
        return True
    except:
        return False

def Formatting(text):
    text = str(text)
    text=text.lower()
    # Removing date from the text
    text = ' '.join([w for w in text.split() if not is_date(w)])
    # Remove numbers
    text = re.sub(r'\d+','' ,text)
    #Remove email
    text = re.sub(r'\S*@\S*\s?', '', 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. &amp;)
    text = re.sub(r'\&\w*;', '', text)
    # Remove hyperlinks
    text = re.sub(r'https?:\/\/.*\/\w*', '', text)
    # Removing addressings
    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 characters beyond Readable formart by Unicode:
    text= ''.join(charac for charac in text if charac <= '\uFFFF')
    text = text.strip()
    # Remove unreadable characters  (also extra spaces)
    text = ' '.join(re.sub("[^\u0030-\u0039\u0041-\u005a\u0061-\u007a]", " ", text).split())
    for caller in tickets_df['Caller'].unique():
        caller_name = [a for a in caller.split()]
        for name in caller_name:
            text = text.replace(name,'')

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

- **Tokenization**: Tokenization is just the term used to describe the process of converting the normal text strings into a list of tokens i.e words that we want. A sentence tokenizer can be used to find the list of sentences and a Word tokenizer can be used to find the list of words in strings. Tokenization breaks the raw text into words, sentences called tokens. These tokens help in understanding the context or developing the model for the NLP.

  - Raw text: I ate a burger, and it was good.

    Tokenized text: [’ I’, ’ate’, ’a’, ’burger’, ‘,’, ‘and’, ’it’, ’was’, ’good’, ‘.’]



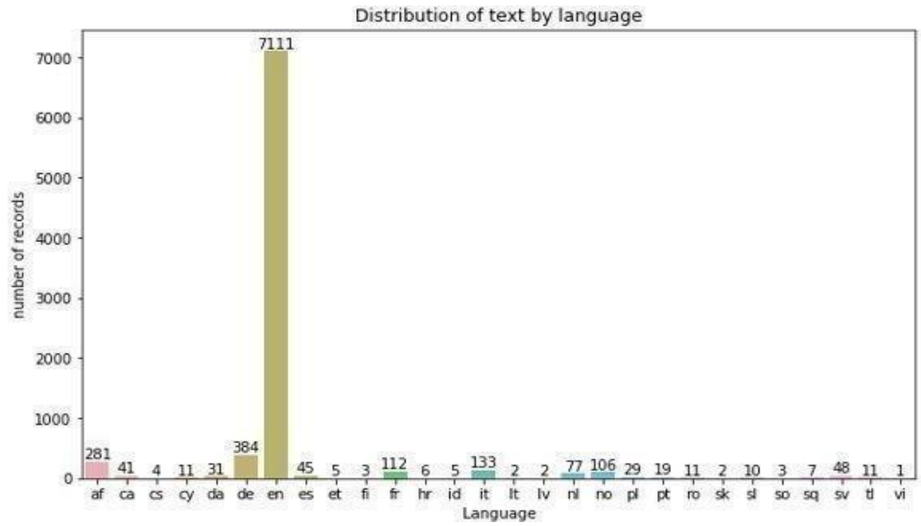
- **Stop words Removal**: Sometimes, some extremely common words which would appear to be of little value in helping select documents matching a user need are excluded from the vocabulary entirely. These words are called stop words.

  - **Stop Words (Removal) NLP**

    Stop words are the most common words in any natural language. To analyze text data and build NLP models, these stop words might not add much value to the meaning of the document. Consider this text string – “There is a pen on the table”. Now, the words “is”, “a”, “on”, and “the” add no meaning to the statement while parsing it. Whereas words like “there”, “book”, and “table” are the keywords and tell us what the statement is all about.

- **Text Processing with Unicode:** Encode the string, to make it easier to be passed to language detection API.

Distribution of text by language



Language Translator

```
In [14]: from google_trans_new import google_translator
         import time
         import langdetect

         def translate(text):
             translator = google_translator(url_suffix="ind",timeout=5)
             translatedText = translator.translate(text)
             time.sleep(0.5)
             return translatedText.lower()
```

Language Detector

```
In [15]: def detect_language_gog(text):
             translator = google_translator()
             temp = translator.detect(text)
             time.sleep(0.5)
             return(temp)

         def detect_language(text):
             return(langdetect.detect(text))
```

- We can see that most of the tickets are in English, followed by tickets in the German language. We need to translate these into English.

- We will be using the google translate package to translate, however, there is a limitation on the number of requests that google translate API can accept per day. So we translated those in batches and saved the translated file to the disk.

- **Label Encoding**
  - In label encoding in Python, we replace the categorical value with a numeric value between **0 and the number of classes minus 1.**
  - To understand label encoding with an example, let us take COVID-19 cases in Indiaacross states. If we observe the below data frame, the State column contains a categorical value that is not very machine-friendly and the rest of the columns contain a numerical value. Let us perform Label encoding for State Column.

State	Confimed	Deaths	Recovered
Maharashtra	284281	11194	158140
Tamil Nadu	156369	2236	107416
Delhi	118645	3545	97693
Karnataka	51422	1032	19729
Gujarat	45481	2089	32103
Uttar Pradesh	43441	1046	26675

  - From the below image, we tokenized the complete vocabulary of the dataset.

Here we are simply lable encoding the target categories

```
In [26]: X = tickets_df["Cleaned"]
         y = tickets_df["Assignment group"]
```

Importing train_test_split from sklearn

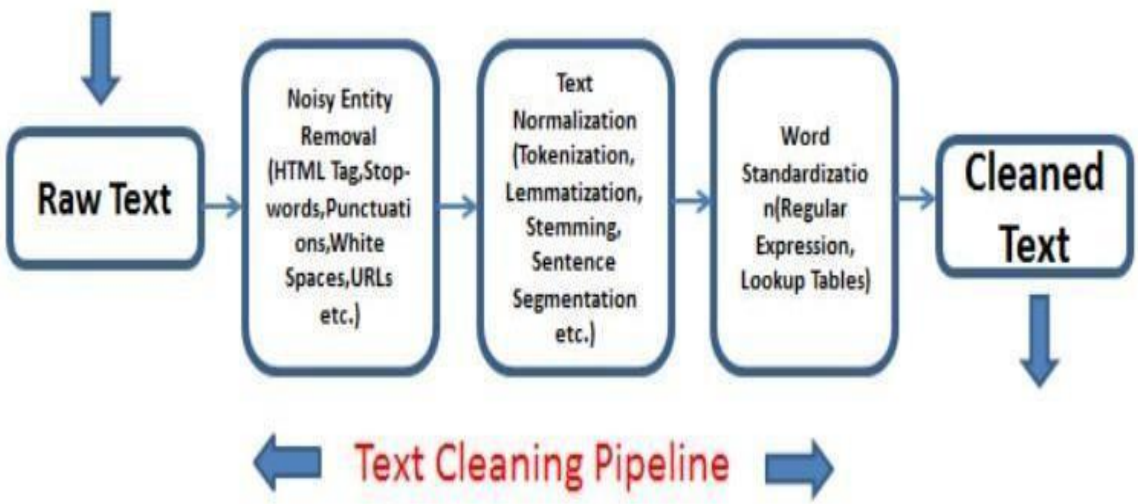
```
In [28]: from tensorflow.keras.preprocessing.text import Tokenizer
         tk = Tokenizer(num_words=max_features)
         tk.fit_on_texts(X.values)
         tk.texts_to_sequences(X)
         X = tk.texts_to_sequences(X.values)

         import pickle

         # saving
         with open('tokenizer.pickle', 'wb') as handle:
             pickle.dump(tk, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

- **Lemmatization with NLTK**
  - Lemmatization is the process of grouping together the different inflected forms of a word so they can be analyzed as a single item. Lemmatization is similar to stemming but it brings context to the words. So it links words with similar meanings to one word.
  - Advantage of lemmatization is that it is more accurate. So if you're dealing with an NLP application such as a chatbot or a virtual assistant where understanding the meaning of the dialogue is crucial, lemmatization would be useful. But this accuracy comes at a cost.
  - Because lemmatization involves deriving the meaning of a word from something like a dictionary, it's very time-consuming. So most lemmatization algorithms are slower compared to their stemming counterparts.

Lemmatize Words ¶

In [23]:
```
import spacy
nlp = spacy.load("en_core_web_sm", disable=['parser', 'ner'])

def lemmatize(text):
    doc = nlp(text)
    allowed_postags=['NOUN', 'ADJ', 'VERB', 'ADV']
    temp = []
    for word in doc:
         if word.pos_ in allowed_postags:
            temp.append(word.lemma_)
    return(' '.join(temp))
```

In [24]:
```
# Applying Lametization
for i in tickets_df.index:
    tickets_df.loc[i,'Cleaned'] = lemmatize(tickets_df.loc[i,'Cleaned'])
```



2. Model evaluation

BERT MODEL:

BERT, or **B**idirectional **E**ncoder **R**epresentations from **T**ransformers, is a new method of pre-training language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks.

- BERT makes use of Transformer, an attention mechanism that learns contextual relations between words (or sub-words) in a text. In its vanilla form, Transformer includes two separate mechanisms — an encoder that reads the text input and a decoder that produces a prediction for the task. Since BERT’s goal is to generate a language model, only the encoder mechanism is necessary. The detailed workings of Transformer are described in a paper by Google.

- As opposed to directional models, which read the text input sequentially (left-to-right or right-to-left), the Transformer encoder reads the entire sequence of words at once. Therefore it is considered bidirectional, though it would be more accurate to say that it’s non-directional. This characteristic allows the model to learn the context of a word based on all of its surroundings (left and right of the word).

<u>BERT pre-processing:</u>

- **Position Embeddings**: BERT learns and uses positional embeddings to express the position of words in a sentence. These are added to overcome the limitation of Transformer which, unlike an RNN, is not able to capture “sequence” or “order” information
- **Segment Embeddings**: BERT can also take sentence pairs as inputs for tasks (Question-Answering). That’s why it learns a unique embedding for the first and the second sentences to help the model distinguish between them. In the above example, all the tokens marked as EA belong to sentence A (and similarly for EB)
- **Token Embeddings**: These are the embeddings learned for the specific token from the WordPiece token vocabulary
  Some special tokens added by BERT are: [SEP] , [CLS]
    - We set the max_seq_length to 30 tokens.
    - It makes a fixed length of sequence for every sample.

```
class Data:
  DATA_COLUMN = "Cleaned"
  LABEL_COLUMN = "Assignment RE-group"

  def __init__(self, train, test, tokenizer: FullTokenizer, classes, max_seq_len=192):
    self.tokenizer = tokenizer
    self.max_seq_len = 0
    self.classes = classes

    train, test = map(lambda df: df.reindex(df[Data.DATA_COLUMN].str.len().sort_values().index), [train, test])

    ((self.train_x, self.train_y), (self.test_x, self.test_y)) = map(self._prepare, [train, test])

    print("max seq_len", self.max_seq_len)
    self.max_seq_len = min(self.max_seq_len, max_seq_len)
    self.train_x, self.test_x = map(self._pad, [self.train_x, self.test_x])

  def _prepare(self, df):
    x, y = [], []

    for _, row in tqdm(df.iterrows()):
      text, label = row[Data.DATA_COLUMN], row[Data.LABEL_COLUMN]
      tokens = self.tokenizer.tokenize(text)
      tokens = ["[CLS]"] + tokens + ["[SEP]"]
      token_ids = self.tokenizer.convert_tokens_to_ids(tokens)
      self.max_seq_len = max(self.max_seq_len, len(token_ids))
      x.append(token_ids)
      y.append(self.classes.index(label))

    return np.array(x), np.array(y)

  def _pad(self, ids):
    x = []
    for input_ids in ids:
      input_ids = input_ids[:min(len(input_ids), self.max_seq_len - 2)]
      input_ids = input_ids + [0] * (self.max_seq_len - len(input_ids))
      x.append(np.array(input_ids))
    return np.array(x)
```

```
[168] tokenizer = FullTokenizer(vocab_file=os.path.join(bert_ckpt_dir, "vocab.txt"))
```

Model & Accuracy:

Architecture:

```
def create_model(max_seq_len, bert_ckpt_file):

  with tf.io.gfile.GFile(bert_config_file, "r") as reader:
      bc = StockBertConfig.from_json_string(reader.read())
      bert_params = map_stock_config_to_params(bc)
      bert_params.adapter_size = None
      bert = BertModelLayer.from_params(bert_params, name="bert")


  input_ids = keras.layers.Input(shape=(max_seq_len, ), dtype='int32', name="input_ids")
  bert_output = bert(input_ids)

  print("bert shape", bert_output.shape)

  cls_out = keras.layers.Lambda(lambda seq: seq[:, 0, :])(bert_output)
  cls_out = keras.layers.Dropout(0.5)(cls_out)
  logits = keras.layers.Dense(units=128, activation="relu")(cls_out)
  logits = keras.layers.Dropout(0.5)(logits)
  logits = keras.layers.Dense(units=len(classes), activation="softmax")(logits)

  model = keras.Model(inputs=input_ids, outputs=logits)
  model.build(input_shape=(None, max_seq_len))


  load_stock_weights(bert, bert_ckpt_file)

  return model
```

```
[173]
classes = df1['Assignment RE-group'].unique().tolist()

data = Data(train, test, tokenizer, classes, max_seq_len=30)
```

```
6224it [00:03, 1928.16it/s]
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:30: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-tuple
1556it [00:00, 1932.66it/s]
max seq_len 1346
```

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
input_ids (InputLayer)      [(None, 30)]              0

bert (BertModelLayer)       (None, 30, 768)           108890112

lambda (Lambda)             (None, 768)               0

dropout (Dropout)           (None, 768)               0

dense (Dense)               (None, 128)               98432

dropout_1 (Dropout)         (None, 128)               0

dense_1 (Dense)             (None, 8)                 1032

=================================================================
Total params: 108,989,576
Trainable params: 108,989,576
Non-trainable params: 0
_________________________________________________________________
```

```
[ ]
model.compile(
  optimizer=keras.optimizers.Adam(1e-5),
  loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
  metrics=[keras.metrics.SparseCategoricalAccuracy(name="acc")]
)
```

Performance:

```

log_dir = "log/intent_detection/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%s")
tensorboard_callback = keras.callbacks.TensorBoard(log_dir=log_dir)

history = model.fit(
  x=data.train_x,
  y=data.train_y,
  validation_data=(data.test_x,data.test_y),
  batch_size=16,
  shuffle=True,
  epochs=5,
  callbacks=[tensorboard_callback]
)
```

```
Epoch 1/5
284/284 [==============================] - 109s 383ms/step - loss: 0.4404 - acc: 0.8505 - val_loss: 0.6105 - val_acc: 0.8025
Epoch 2/5
284/284 [==============================] - 107s 378ms/step - loss: 0.3716 - acc: 0.8714 - val_loss: 0.5803 - val_acc: 0.8175
Epoch 3/5
284/284 [==============================] - 108s 381ms/step - loss: 0.3197 - acc: 0.8842 - val_loss: 0.6802 - val_acc: 0.8254
Epoch 4/5
284/284 [==============================] - 108s 381ms/step - loss: 0.2572 - acc: 0.9069 - val_loss: 0.8192 - val_acc: 0.8069
Epoch 5/5
284/284 [==============================] - 108s 379ms/step - loss: 0.2278 - acc: 0.9124 - val_loss: 0.9301 - val_acc: 0.8395
```

Model	Categories	Train_accuracy	Test_accuracy
Bert	TOP 5 Classes	95.47	91.03
	TOP 8 Classes	90.62	83.95
	ALL 74 Classes	79.25	64.4

GenSim word2vec Embeddings with Bi-Directional LSTM:

<u>GenSim word2vec preprocessing:</u>

Word2Vec is a widely used algorithm based on neural networks, commonly referred to as “deep learning” (though word2vec itself is rather shallow). Using large amounts of unannotated plain text, word2vec learns relationships between words automatically. The output are vectors, one vector per word, with remarkable linear relationships that allow us to do things like:

- vec(“king”) - vec(“man”) + vec(“woman”) =~ vec(“queen”)
- vec(“Montreal Canadiens”) – vec(“Montreal”) + vec(“Toronto”) =~ vec(“Toronto Maple Leafs”).

1) we have created a phrase_model that creates bigrams for relevant words using "npmi" scoring from the cleaned description.

2)we have converted every phrase(uni-grams and bi-grams included) into a 50 dimension embedding.

3)The word2vec model used is the skip gram model with 10 negative sampling. And created a bidirectional model with that.

```
[ ]  1 from gensim.models import Word2Vec

[ ]  1 sentences = df['phrases'].tolist()

[ ]  1 vec_model = Word2Vec(sentences,min_count=1,size=50,sg=1,negative=10,iter=100)

[ ]  1 print(vec_model['outlook'])

[-0.06557938  1.3137809  -0.08147745  0.25948685 -0.6277236   0.3044606
  0.04053769 -0.36101955  0.02466223  0.69405454  1.3979331  -0.16846064
 -0.8171429  -0.02235341 -0.56769943  0.44449264 -0.97246295 -0.07660929
  0.14324284 -0.59499925 -0.30560502  0.44988126 -0.3167281   0.36009607
  0.12176763 -0.30439937  1.1364713   0.810118    0.428784   -0.21388547
  0.80163926  0.52291846 -0.44619545 -0.17165008 -0.11869454 -0.42450887
 -0.44253516  0.14792119  0.6815057   0.17404675  0.67598367  0.0649448
 -0.27485707 -0.60135895  0.25863114  0.47501072  0.16159262 -0.3043874
 -0.14369848  0.21277744]
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: DeprecationWarning: Call to deprecated `__getitem__` (Method will be removed in 4.0.0, use self.wv.__getitem__() i
  """Entry point for launching an IPython kernel.
```

Model & Accuracy:

Architecture:

jupyter GENSIM word2vec embeddings Last Checkpoint: 5 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

Run Code

In [171]:
```
sgd = optimizers.SGD(learning_rate=0.001,momentum=0.9)
adam = optimizers.Adam()

model.compile(loss = 'categorical_crossentropy', optimizer = adam, metrics= ['accuracy'])
```

In [109]:
```
# from tensorflow.keras.callbacks import ReduceLROnPlateau,EarlyStopping

# early_stop = EarlyStopping(monitor='val_loss',patience=3)
# reduc = ReduceLROnPlateau(monitor='val_loss',patience=4,factor=0.5)
```

In [172]:
```
model.fit(x_train,y_train,epochs=30,batch_size=32,validation_data=(x_test,y_test))
```

```
Epoch 1/30
142/142 [==============================] - 7s 24ms/step - loss: 1.1639 - accuracy: 0.7230 - val_loss: 0.8116 - val_accurac
y: 0.7751
Epoch 2/30
142/142 [==============================] - 3s 18ms/step - loss: 0.7713 - accuracy: 0.7676 - val_loss: 0.7464 - val_accurac
y: 0.7795
Epoch 3/30
142/142 [==============================] - 3s 18ms/step - loss: 0.7098 - accuracy: 0.7816 - val_loss: 0.7156 - val_accurac
y: 0.7840
Epoch 4/30
142/142 [==============================] - 3s 18ms/step - loss: 0.6752 - accuracy: 0.7870 - val_loss: 0.7150 - val_accurac
y: 0.7610
Epoch 5/30
142/142 [==============================] - 3s 18ms/step - loss: 0.6577 - accuracy: 0.7900 - val_loss: 0.6990 - val_accurac
y: 0.7831
Epoch 6/30
142/142 [==============================] - 3s 18ms/step - loss: 0.6554 - accuracy: 0.7915 - val_loss: 0.7071 - val_accurac
y: 0.7813
```

jupyter GENSIM word2vec embeddings Last Checkpoint: 5 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

Run Code

```
142/142 [==============================] - 3s 18ms/step - loss: 0.5447 - accuracy: 0.8212 - val_loss: 0.6538 - val_accurac
y: 0.8016
Epoch 21/30
142/142 [==============================] - 3s 18ms/step - loss: 0.5498 - accuracy: 0.8192 - val_loss: 0.6860 - val_accurac
y: 0.7831
Epoch 22/30
142/142 [==============================] - 3s 18ms/step - loss: 0.5702 - accuracy: 0.8097 - val_loss: 0.6784 - val_accurac
y: 0.8042
Epoch 23/30
142/142 [==============================] - 3s 18ms/step - loss: 0.5457 - accuracy: 0.8075 - val_loss: 0.6706 - val_accurac
y: 0.7972
Epoch 24/30
142/142 [==============================] - 3s 18ms/step - loss: 0.5254 - accuracy: 0.8292 - val_loss: 0.6945 - val_accurac
y: 0.8042
Epoch 25/30
142/142 [==============================] - 3s 18ms/step - loss: 0.5261 - accuracy: 0.8215 - val_loss: 0.7058 - val_accurac
y: 0.7972
Epoch 26/30
142/142 [==============================] - 3s 18ms/step - loss: 0.5218 - accuracy: 0.8209 - val_loss: 0.6862 - val_accurac
y: 0.7981
Epoch 27/30
142/142 [==============================] - 3s 18ms/step - loss: 0.5095 - accuracy: 0.8312 - val_loss: 0.6812 - val_accurac
y: 0.8113
Epoch 28/30
142/142 [==============================] - 3s 18ms/step - loss: 0.5028 - accuracy: 0.8249 - val_loss: 0.7040 - val_accurac
y: 0.8069
Epoch 29/30
142/142 [==============================] - 3s 18ms/step - loss: 0.4888 - accuracy: 0.8298 - val_loss: 0.7308 - val_accurac
y: 0.7919
Epoch 30/30
142/142 [==============================] - 3s 18ms/step - loss: 0.4847 - accuracy: 0.8262 - val_loss: 0.7142 - val_accurac
y: 0.7981
```

Model	Categories	Train_accuracy	Test_accuracy
Gensim phrases	TOP 5 Classes	96.13%	91.56
	TOP 8 Classes	95.48	83.12
	ALL 74 Classes	86.79	63.03

ELMO Model Embedding's with Neural Net:

Embeddings from Language Models, or ELMo, is a type of deep contextualized word representation that models both (1) complex characteristics of word use (e.g., syntax and semantics).

ELMO preprocessing:

The sample text was passed through the ELMO module to generate 1024 size vector embeddings for each text and the generated embeddings will be stored in pickle file and used in a Bir-Directional LSTM model

```
In [4]: # Importing Elmo
import tensorflow as tf
import tensorflow_hub as hub
import numpy as np
# import tensorflow_text

# elmo = hub.Module("C:/Users/Kalpesh/Great Lakes/Capstone/module/", trainable=True)
elmo = hub.Module("https://tfhub.dev/google/elmo/2", trainable=True)
```

```
In [5]: def elmo_vectors(x):
    embeddings = elmo(x,signature="default",as_dict=True)["default"]
    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())
        sess.run(tf.tables_initializer())
        y = sess.run(embeddings)
    return y
#     return(tf.reduce_mean(embeddings,1))
```

```
In [7]: list_train = [x_train[i:i+50] for i in range(0,x_train.shape[0],50)]
list_test = [x_test[i:i+50] for i in range(0,x_test.shape[0],50)]
```

```
In [ ]: # Extract ELMo embeddings
elmo_train = []
for x in list_train:
  vectors = elmo_vectors(x)
  elmo_train.append(vectors)

elmo_test = []
for x in list_test:
  vectors = elmo_vectors(x)
  elmo_test.append(vectors)
```

```
In [ ]: # Extract ELMo embeddings
elmo_train = [elmo_vectors(x) for x in list_train]
elmo_test = [elmo_vectors(x) for x in list_test]
```

```
In [ ]: elmo_train_new = np.concatenate(elmo_train, axis = 0)
        elmo_test_new = np.concatenate(elmo_test, axis = 0)
```

```
In [ ]: pickle_out = open("elmo_train_03032019.pickle","wb")
        pickle.dump(elmo_train_new, pickle_out)
        pickle_out.close()

        # save elmo_test_new
        pickle_out = open("elmo_test_03032019.pickle","wb")
        pickle.dump(elmo_test_new, pickle_out)
        pickle_out.close()
```

Model & Accuracy:

Architecture:

```
In [12]: from tensorflow.keras.layers import LSTM,Embedding,Flatten,Input,Dense,InputLayer
         from tensorflow.keras.models import Sequential

         model = Sequential()
         model.add(InputLayer(input_shape = (1024,)))
         model.add(Dense(500))
         model.add(Dense(8,activation='softmax'))
         print(model.summary())

Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 500)               512500
_________________________________________________________________
dense_1 (Dense)              (None, 8)                 4008
=================================================================
Total params: 516,508
Trainable params: 516,508
Non-trainable params: 0
_________________________________________________________________
None
```

Model	Categories	Train_accuracy	Test_accuracy
ELMO	TOP 5 Classes	94.5	90.1
	TOP 8 Classes	93.8	85.2
	ALL 74 Classes	70.7	61.7

FastText Model

FastText supports training continuous bag of words (CBOW) or Skip-gram models using negative sampling, softmax or hierarchical softmax loss functions.

FastText is able to achieve really good performance for word representations and sentence classification, specially in the case of rare words by making use of character level information.
Each word is represented as a bag of character n-grams in addition to the word itself, so for example, for the word `matter`, with n = 3, the fastText representations for the character n-grams is `<ma`, `mat`, `att`, `tte`, `ter`, `er>`.

FastText preprocessing:

In [ ]:
```
from io import StringIO
import csv

col = ['Assignment group', 'Cleaned']

testdata = test_data[col]
testdata['Assignment group']=['__label__'+ s for s in testdata['Assignment group']]
testdata.to_csv(r'testdata.txt', index=False, sep=' ', header=False, quoting=csv.QUOTE_NONE, quotechar="", escapechar=" ")

traindata = train_data[col]
traindata['Assignment group']=['__label__'+ s for s in traindata['Assignment group']]
#new_data['Cleaned']= new_data['Cleaned'].replace('\n',' ', regex=True).replace('\t',' ', regex=True)
traindata.to_csv(r'traindata.txt', index=False, sep=' ', header=False, quoting=csv.QUOTE_NONE, quotechar="", escapechar=" ")
```

Model & Accuracy:

Model	Categories	Train_accuracy	Test_accuracy
Fasttext Models	TOP 5 Classes	90.89	88
	TOP 8 Classes	89	83
	ALL 74 Classes	75	59

```
import fasttext

model = fasttext.train_supervised(input="traindata.txt")
```

[18]
```
model.test("testdata.txt")
```

```
(1684, 0.584916864608076, 0.584916864608076)
```

GLove Embeddings with Bi-Directional LSTM:

GLove preprocessing:

GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.

GloVe is essentially a log-bilinear model with a weighted least-squares objective. The main intuition underlying the model is the simple observation that ratios of word-word co-occurrence probabilities have the potential for encoding some form of meaning. For example, consider the co-occurrence probabilities for target words *ice* and *steam* with various probe words from the vocabulary. Here are some actual probabilities from a 6 billion word corpus:

We have used the 300 dimension glove vector embeddings to form the embedding marix for the given vocabulary.

```
In [17]: # Function for loading and reading GLove Embeddings
def read_glove_vector(glove_vec):
    with open(glove_vec, 'r', encoding='UTF-8') as f:
        words = set()
        word_to_vec_map = {}
        for line in f:
            w_line = line.split()
            curr_word = w_line[0]
            word_to_vec_map[curr_word] = np.array(w_line[1:], dtype=np.float64)
    return word_to_vec_map
```

```
In [21]: # GLove Embedding vector
word_to_glove_map = read_glove_vector('glove.42B.300d.txt')
word_to_glove_map
```

```
        1.8785e-01,  2.7849e-03, -1.8411e-01, -1.1514e-01, -7.8581e-01]),
 ',': array([ 0.013441,  0.23682 , -0.16899 ,  0.40951 ,  0.63812 ,  0.47709 ,
       -0.42852 , -0.55641 , -0.364   , -0.23938 ,  0.13001 , -0.063734,
       -0.39575 , -0.48162 ,  0.23291 ,  0.090201, -0.13324 ,  0.078639,
       -0.41634 , -0.15428 ,  0.10068 ,  0.48891 ,  0.31226 , -0.1252  ,
       -0.037512, -1.5179  ,  0.12612 , -0.02442 , -0.042961, -0.28351 ,
        3.5416  , -0.11956 , -0.014533, -0.1499  ,  0.21864 , -0.33412 ,
       -0.13872 ,  0.31806 ,  0.70358 ,  0.44858 , -0.080262,  0.63003 ,
        0.32111 , -0.46765 ,  0.22786 ,  0.36034 , -0.37818 , -0.56657 ,
        0.044691,  0.30392 ]),
 '.': array([ 1.5164e-01,  3.0177e-01, -1.6763e-01,  1.7684e-01,  3.1719e-01,
        3.3973e-01, -4.3478e-01, -3.1086e-01, -4.4999e-01, -2.9486e-01,
        1.6608e-01,  1.1963e-01, -4.1328e-01, -4.2353e-01,  5.9868e-01,
        2.8825e-01, -1.1547e-01, -4.1848e-02, -6.7989e-01, -2.5063e-01,
        1.8472e-01,  8.6876e-02,  4.6582e-01,  1.5035e-02,  4.3474e-02,
       -1.4671e+00, -3.0384e-01, -2.3441e-02,  3.0589e-01, -2.1785e-01,
        3.7460e+00,  4.2284e-03, -1.8436e-01, -4.6209e-01,  9.8329e-02,
       -1.1907e-01,  2.3919e-01,  1.1610e-01,  4.1705e-01,  5.6763e-02,
       -6.3681e-05,  6.8987e-02,  8.7939e-02, -1.0285e-01, -1.3931e-01,
```

```
In [38]: embedding_matrix = np.zeros((num_words, 300))
for word, i in tk.word_index.items():
    embedding_vector = embeddings.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector
```

Model & Accuracy:

Architecture:

vocab_len will be the no. of words in our embedding matrix therefore taking it from the 0th index of the shape of embedding matrix

```
In [41]: from tensorflow.keras.layers import LSTM,Embedding,Flatten,Input,Dense,Bidirectional
         from tensorflow.keras.models import Sequential
```

Importing LSTM, Embedding, Flatten, Input, Dense, Bidirectional and Sequential

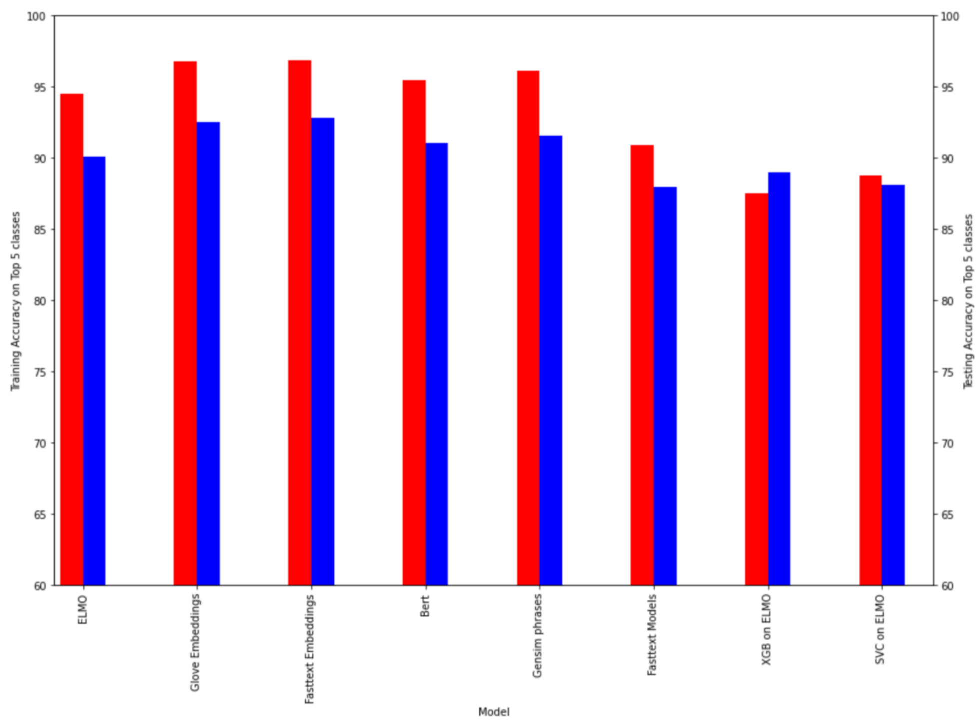
```
In [71]: model = Sequential()
         model.add(Embedding(input_dim = vocab_len,output_dim = embedding_size,weights = [embedding_matrix],input_length=maxlen,trainable
         model.add(Bidirectional(LSTM(100,return_sequences=False)))

         model.add(Dense(64))
         model.add(Dense(74,activation='softmax'))
         print(model.summary())
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_6 (Embedding)      (None, 50, 300)           2986200
_________________________________________________________________
lstm_6 (LSTM)                (None, 50)                70200
_________________________________________________________________
dense_12 (Dense)             (None, 64)                3264
_________________________________________________________________
dense_13 (Dense)             (None, 74)                4810
=================================================================
Total params: 3,064,474
Trainable params: 78,274
Non-trainable params: 2,986,200
```

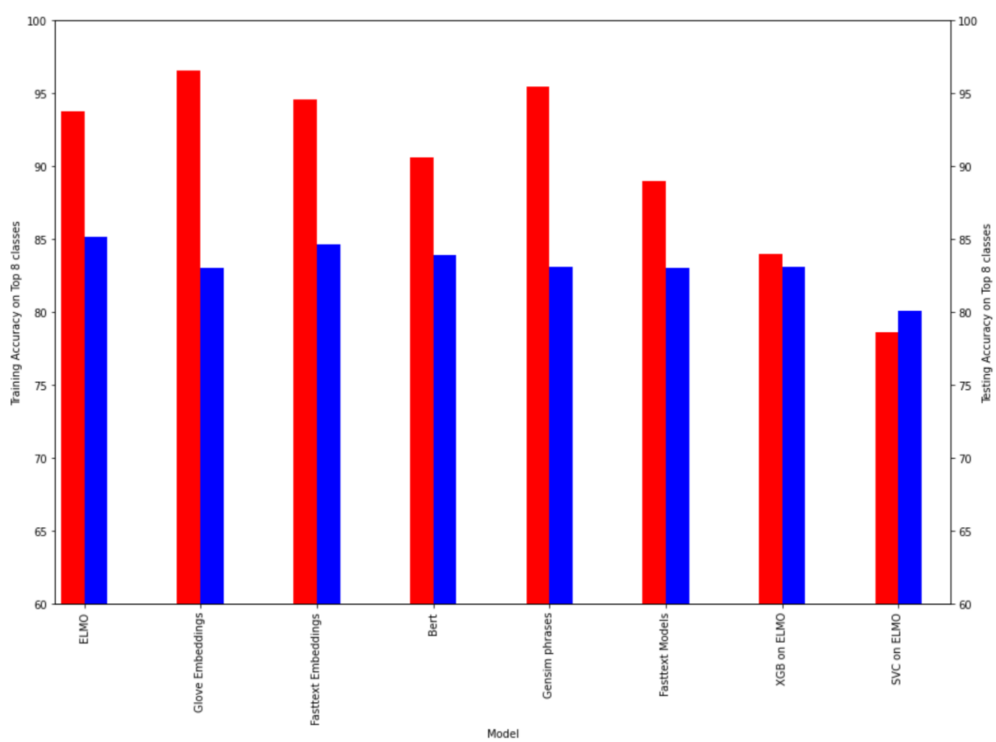
Model	Categories	Train_accuracy	Test_accuracy
Glove Embeddings	TOP 5 Classes	96.8	92.5
	TOP 8 Classes	96.56	83.06
	ALL 74 Classes	92.37	65.4

Model Comparison

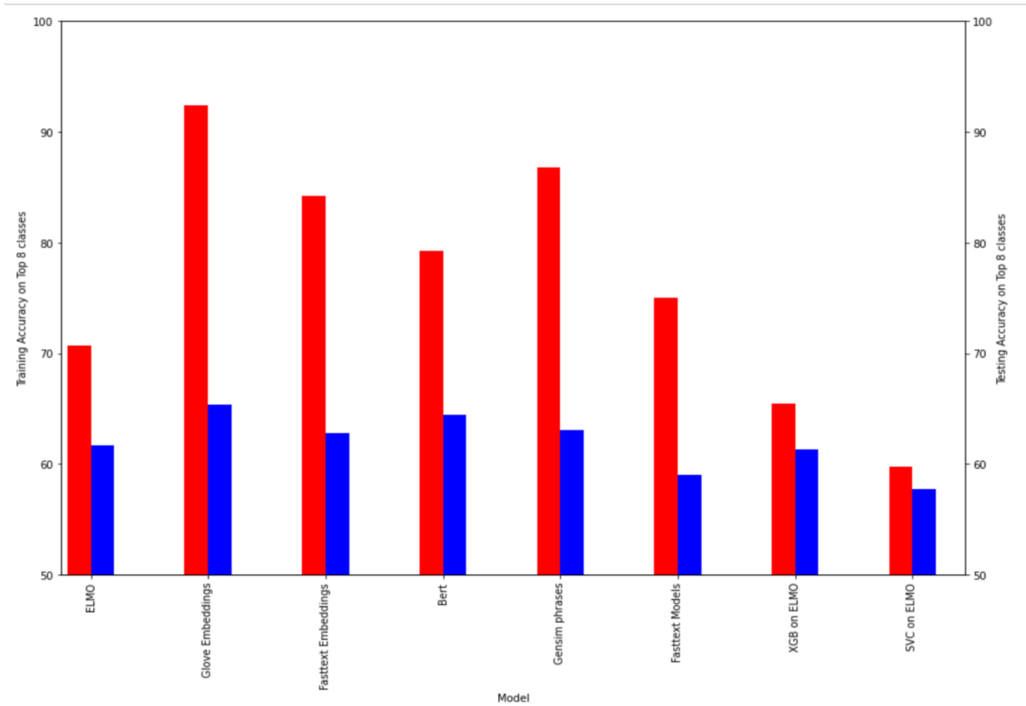
Performance on Top 5 classes (With maximum sample points):



Performance on Top 8 classes (With maximum sample points):



Performance on Top 8 classes:



- Comparing various models as seen in the graphs above we selected the GLove Embedding's with the Bi-Directional LSTM Model as it gave the maximum accuracy for all the 74 classes which is around 65% and also shows the best F1 scores for most of the classes in the classification.
- Also the Fast text model can be used as faster alternative if required as consumes lesser time for processing and training and delivers similar results to the deep learning models.

5. Comparison to Benchmark

At the outset, looking the dataset and its imbalance we targeted an accuracy of around 70 % in predicting all the 74 classes in the dataset and also aimed F1 score above 65 % in top 8 classes and overall F1 score of around 50%

But we were able to achieve accuracy around 65 % over all the 74 classes and we were able to achieve F1 score above 70 % on 3 classes out of top 8 classes.

1) We were able to achieve minor benchmark of achieving higher F1 score only for few classes because this 3 classes had the highest number of samples (above 250 samples) thus the model generalized well on the three classes but failed on the others which had fewer samples.

2) Overall Accuracy also stalled at 65 % because of the heavy imbalance in the classes. The data set constituting 40 % of data samples belonging to a single class i.e. Group_0 and also the fact that the classes do not have enough data samples for the model to learn and develop a cognitive detection for the same.

3) Out of the 65 % accuracy achieved the data being constituent of high Group_0 values contributed greatly to the accuracy thus although the accuracy is high for the Group_0 the model cannot perform better on other classes. Thus led to a very low Overall F1 score of 40% only

6. Deployment

Deployment of machine learning models or putting models into production means making your models available to the end users or systems. Here we have deployed the application using FlaskWeb application.

Following are the steps for model deployment:

- First, we saved the model and its weights (i.e a python object on the disk that can be transferred anywhere and later calles back by a python script)

- Write a Flask Application which has below parts:

  - app.py — This contains Flask APIs that receives ticket details through GUI or API calls, computes the predicted value based on our model and returns it.
  - Functions to clean the input text before loaded to the model
    - preprocess the text by cleaning,
    - removing stop words and
    - translating the text to English
    - Lemmatizing

```python
def Formatting(text):
    text = str(text)
    text=text.lower()
    # Removing date from the text
    text = ' '.join([w for w in text.split() if not is_date(w)])
    # Remove numbers
    text = re.sub(r'\d+','' ,text)
    #Remove email
    text = re.sub(r'\S*@\S*\s?', '', 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. &amp;)
    text = re.sub(r'\&\w*;', '', text)
    # Remove hyperlinks
    text = re.sub(r'https?:\/\/.*\/\w*', '', text)
    # Removing addressings
    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 characters beyond Readable formart by Unicode:
    text= ''.join(charac for charac in text if charac <= '\uFFFF')
    text = text.strip()
    # Remove unreadable characters  (also extra spaces)
    text = ' '.join(re.sub("[^\u0030-\u0039\u0041-\u005a\u0061-\u007a]", " ", text).split())
    text = re.sub(r"\s+[a-zA-Z]\s+", ' ', text)
    text = re.sub(' +', ' ', text)
    text = text.strip()
    return text
```

```python
def lemmatize(text):
    doc = nlp(text)
    allowed_postags=['NOUN', 'ADJ', 'VERB', 'ADV']
    temp = []
    for word in doc:
        if word.pos_ in allowed_postags:
            temp.append(word.lemma_)
    return(' '.join(temp))


def translate(text):
    translator = google_translator(url_suffix="ind",timeout=5)
    if (translator.detect(text))[0] != 'en':
        translatedText = translator.translate(text)
        time.sleep(0.5)
        return translatedText.lower()
    else:
        return(text.lower())
```

- Function to Load serialized model

```python
def load_model():
    model = keras.models.load_model('C:\\Users\\Kalpesh\\Great lakes\\Capstone\\model')
    return(model)
```

- This creates a route that receives input via GUI

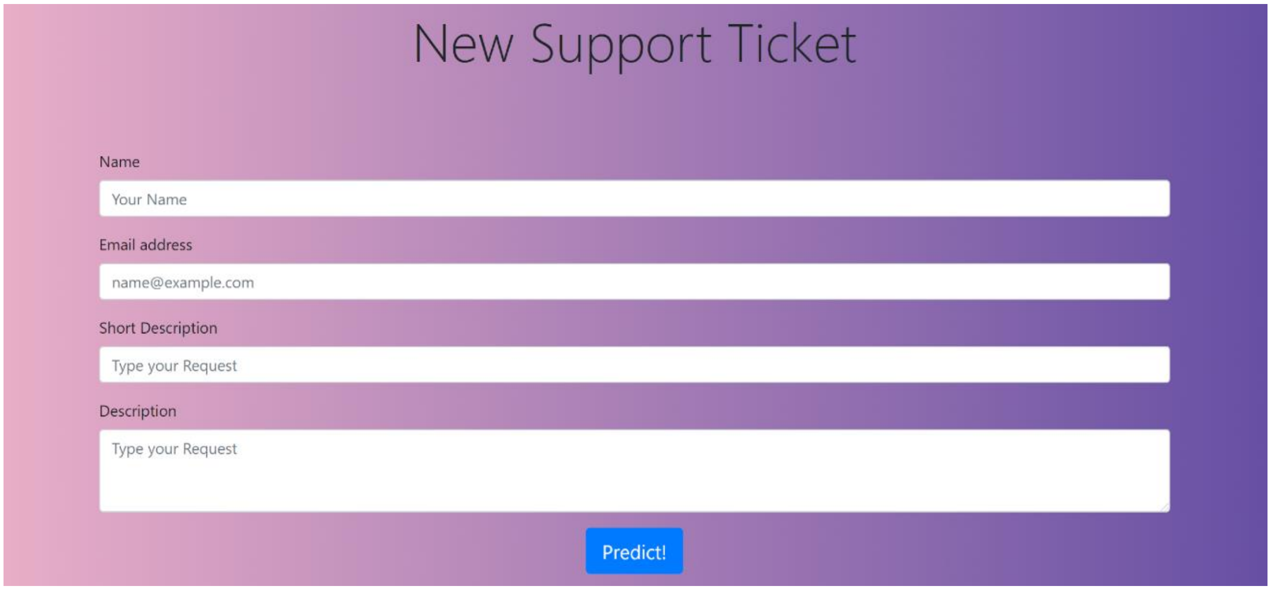
```python
app = Flask(__name__)
Bootstrap(app)

@app.route('/')
def man():
    model = load_model()
    return render_template('home.html')


@app.route('/predict', methods=['POST'])
def home():
    maxlen = 50
    embedding_size = 300
    S_D = translate(Formatting(request.form['b']))
    D = translate(Formatting(request.form['c']))
    if str(S_D) == str(D):
        Desc = str(D)
    else:
        Desc = str(S_D) + str(D)
    Desc = " ".join(word for word in Desc.split(' ') if word not in stopwords.words('english'))
    Desc = lemmatize(Desc)
    tk = Tokenizer()
    with open('C:\\Users\\Kalpesh\\Great lakes\\Capstone\\tokenizer.pickle', 'rb') as handle:
        tk = pickle.load(handle)
    X = tk.texts_to_sequences(Desc)
    X = pad_sequences(X,maxlen=maxlen,padding='post')
    Labels = load_labels()
    model = load_model()
    pred = np.argmax(reduce_mean(model.predict(X),0))

    return render_template('after.html', data=Labels[pred])


if __name__ == "__main__":
    app.run(debug=True)
```

Then it uses the trained model to make a prediction, and returns that prediction, which can be accessed through the API endpoint.

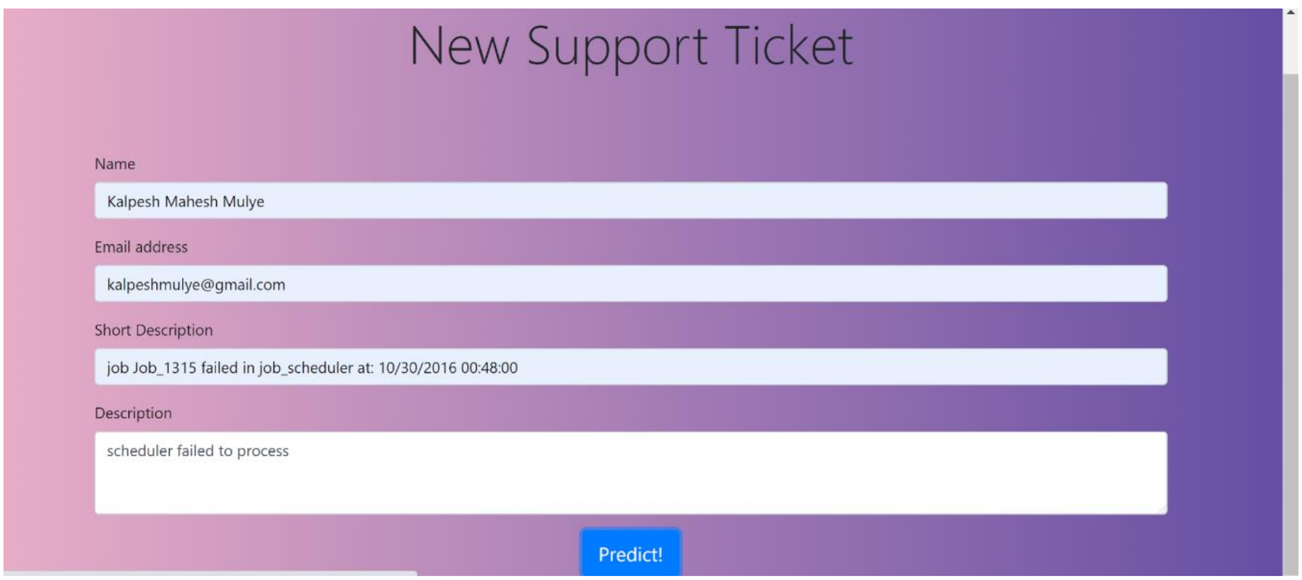
- HTML/CSS — This contains the HTML template and CSS styling to allow users to enter issue/ticket details and displays the predicted assignment group
    - Web application is hosted on localhost

Below are few snapshots of web application host on Flask Framework:

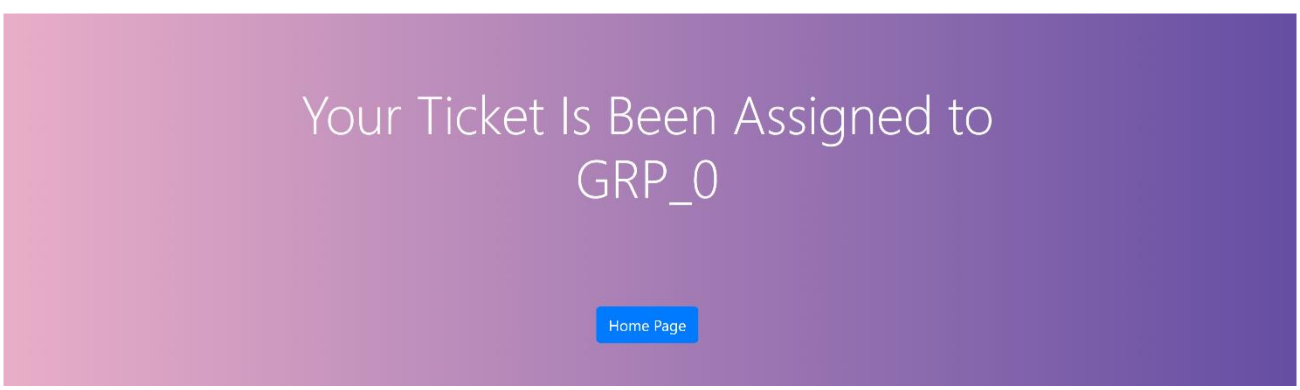
HOME PAGE:



Test Case:



Predict:



7.Implications

We first analysed the dataset provided to us, understood the structure of the data provided - number of columns, field, data types etc.

- We did Exploratory Data Analysis to derive further insights from this data set and we found that:
    - Data is very much imbalanced, there are around ~45% of the Groups with less than 20 tickets.
    - Few of the tickets are in foreign language like German
    - The data has a lot of noise in it, for eg- few tickets related to account setup are spread across multiple assignment groups41
- We performed the data cleaning and preprocessing
    - Translation: A small number of tickets were written in German. Hence, we used the Google translate python api to convert German to English to generate the input data for the next steps. However, the google translator rest api can only process a limited number of texts on a daily basis, so we translated the text in batches and saved the file for further processing.
    - Make text all lowercase so that the algorithm does not treat the same words in different cases as different
    - Removing Noise i.e everything that isn't in a standard number or letter i.e Punctuation, Numerical values
    - Removing extract spaces
    - Removed punctuations
    - Removed words containing numbers
    - Stopword Removal: Sometimes, some extremely common words which would appear to be of little value in helping select documents matching a user need are excluded from the vocabulary entirely. These words are called stop words
    - Lemmatization
    - Tokenization: Tokenization is just the term used to describe the process of converting the normal text strings into a list of tokens i.e words that we actually want. Sentence tokenizer can be used to find the list of sentences and Word tokenizer can be used to find the list of words in strings.
- We then ran a basic benchmark model using the cleaned and preprocessed dataset

- Since the dataset is very imbalanced, We considered a subset of groups for predictions. In 74 groups, 46% of tickets belong to group 1 and 16 groups just have more than 100 tickets, rest of the Assignment groups have very less ticket counts which might not add much value to the model prediction. If we conducted random sampling towards all the subcategories, then we would face a problem that we might miss all the tickets in some categories. Hence, we considered the groups that have more than 100 tickets.

- We trained the data using below models:
    - Bi-Directional Embedding
    - Glove Embedding
    - Fasttext

- Even after downsampling the data we see that the predictions are biased towards GRP_0 which has a majority of samples.

- Imbalance causes two problems:

    - Training is inefficient as most samples are easy examples that contribute no useful learning signal;

    - The easy examples can overwhelm training and lead to degenerate models. A common solution is to perform some form of hard negative mining that samples hard examples during training or more complex sampling/re weighing schemes. In order to handle the imbalance problem we used class_weight=balanced hyper parameter while training the model, which tells the model to "pay more attention" to samples from an under-represented class.

- Although, the accuracy and f1_score went down. This ensured that the classes were being correctly classified with lesser number of misclassification and good precision/recall scores for all the classes.

- Next, we also tried using pre trained word embedding, but the only challenge was that we could not find any embeddings trained on ITSM data. We used the glove model with 50d for this but, the scores were poorer than the benchmark model.

- Then, we also tried 'fasttext' modeling with 300d

- In most cases results were pretty similar but for some of the models, bidirectional modelling and Glove modelling performed much better.

- We also tried an alternative approach, as it's mentioned that around ~54% of the incidents are resolved by L1 / L2 teams and the rest will be resolved as L3. So the assumption is that GRP_0 and GRP_8 which contribute 54% of the tickets are related to L1/L2 teams and the rest of the tickets belongs to L3 teams

- we used Approach 2 where the ticket would be classified into L1/L2 or L3 classes and then it would be further classified into one of the given assignment groups.

- We first created a model to classify the given tickets as l1/l2 or l3 tickets

- Next, another model was trained considering only the l1/l2 level of incidents consisting of GRP_0 and GRP_8.

- Finally , a third model was trained considering l3 level of tickets 74

8 Limitations

We also tried embedding implementations with focal loss as a loss function to handle the class imbalance problem, which helps in giving more weightage to groups with less samples, but the results were not satisfactory. In our dataset, 'texts' are domain-specific and texts are quite rough in nature.

9 Closing Reflections

- Machine learning model is able to predict the assignment groups instantaneously for the new tickets
- The prediction is accurate for ~74%(73.98%) of the tickets
- Machine learning-based automation of triaging has the ability to reduce the load on the triage team to a greater extent
- As the data is from the IT Organization most of the text has IT jargons in it. So we didn't able to map expressive embedding for those words. Due to this embeddings don't have a proper contextual understanding which is a downside for model's performance. In future we can use pre-trained word embeddings on IT Corpus and use those embeddings in our task which will definitely increase our model's performance.
- In the future we can improve the performance of models if we get a minimum of 300 tickets for every group. Due to less data models are not able to learn required patterns from the data. So with the decent amount of data our models eventually improves and performs better in the future.

END of Final Report.