AIML ONLINE CAPSTONE

GROUP NUMBER: SEPT SUN GROUP 4 B

Natural Language Processing

AUTOMATIC IT SUPPORT TICKET ASSIGNMENT

TEAM

RADHIKA KULKARNI

SYED TALIBUDDIN SAIFI

TANVEER QURAISHI

SHANMUGAM SAMPATH

KANWAL RAI

MENTOR

ASHISH VYAS

Table of Content

Part A : Project Planning Phase

1. Problem Background, Data and Objectives

1.1 Project Objectives

1.2 Exploratory Data Analysis

1.3 Features and Target Variables

2. Model Selection

Part B: Project Execution Phase

3. Dataset Preparation

3.1 Data Pre-Processing

3.2 Data Cleaning

3.3 Translation

3.4 Lemmatization & Stop Words Removal

3.5 Topic Modelling

3.6 Word Cloud Visualization

3.7 Word Embedding/Vectorization

4. Model Development

4.1 Machine Learning: SVM, RFC, NB

4.2 Deep Learning: LSTM, GRU, RNN

4.3 Transformers: BERT

5. Model Evaluation

5.1 Comparative Benchmarking

5.2 Implications

5.3 Limitations

5.4 Closing Reflection

Interim Report (Submitted on 16th August 2020)

Final Report (Submitted on 6th September 2020)

A

Project Planning Phase

Project Background, Data and Objectives

Background

One of the key activities of any IT function is to "Keep the lights on" to ensure there is no impact to the Business operations. IT leverages Incident Management process to achieve the above Objective. An incident is something that is unplanned interruption to an IT service or reduction in the quality of an IT service that affects the Users and the Business. The main goal of Incident Management process is to provide a quick fix / workarounds or solutions that resolves the interruption and restores the service to its full capacity to ensure no business impact. In most of the organizations, incidents are created by various Business and IT Users, End Users/ Vendors if they have access to ticketing systems, and from the integrated monitoring systems and tools. Assigning the incidents to the appropriate person or unit in the support team has critical importance to provide improved user satisfaction while ensuring better allocation of support resources. The assignment of incidents to appropriate IT groups is still a manual process in many of the IT organizations. Manual assignment of incidents is time consuming and requires human efforts. There may be mistakes due to human errors and resource consumption is carried out ineffectively because of the misaddressing. On the other hand, manual assignment increases the response and resolution times which result in user satisfaction deterioration / poor customer service.

In the support process, incoming incidents are analyzed and assessed by organization's support teams to fulfill the request. In many organizations, better allocation and effective usage of the valuable support resources will directly result in substantial cost savings. Currently the incidents are created by various stakeholders (Business Users, IT Users and Monitoring Tools) within IT Service Management Tool and are assigned to Service Desk teams (L1 / L2 teams). This team will review the incidents for right ticket categorization, priorities and then carry out initial diagnosis to see if they can resolve. Around ~54% of the incidents are resolved by L1 / L2 teams. 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. 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. 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.

Data

Details about the data and dataset files are given in below link: https://drive.google.com/file/d/1OZNJm81JXucV3HmZroMq6qCT2m7ez7lJ/view

Objectives

In this capstone project, the goal is to build a classifier that can classify the tickets by analyzing text. The objective of the project is, (a) Learn how to use different classification models. (b) Use transfer learning to use pre-built models. (c) Learn to set the optimizers, loss functions, epochs, learning rate, batch size, checkpointing, early stopping etc. (d) Read different research papers of given domain to obtain the knowledge of advanced models for the given problem. As per the background, the existing system assigns 70-75% of the support tickets effectively. Our objective in terms of effectiveness would be to build AI models to classify these tickets with an accuracy of at lest 80-85%.

Ticket Classification Using ML & DL Algorithms

When an issue or support ticket drops into your help desk, first it needs to be processed and assigned a tag or category so that it's routed to the correct team member. This involves reading the ticket, so that agents know which category to choose. However, manual classification systems are often complicated and cluttered with too many categories for agents to choose from. After spending endless hours going through tickets, agents will often end up assigning tickets the 'Other' category to sort them faster and avoid spending precious time searching for the correct category.

The assignment of tickets to a group is basically a multi-class text classification problem. This problem is a widely studied problem in which various algorithms and feature extraction techniques can be used. However the proposed system is to be language independent based on word embeddings and deep learning methods of classification

When an issue or support ticket drops into your help desk, first it needs to be processed and assigned a tag or category so that it's routed to the correct team member. This involves reading the ticket, so that agents know which category to choose. However, manual classification systems are often complicated and cluttered with too many categories for agents to choose from. After spending endless hours going through tickets, agents will often end up assigning tickets the 'Other' category to sort them faster and avoid spending precious time searching for the correct category. Ticket classification with machine learning avoids this problem, firstly because machine learning tools will only assign a tag that you've previously defined, and secondly because ticket categorization models will assign these tags automatically, without you needing to scroll through a long list of tags. Instead of humans interpreting content and categorizing it accordingly, automatic ticket classification uses Natural Language Processing, a subfield of machine learning, which helps machines process, understand, and potentially generate human language in a fast and cost-effective way.

Automated ticket classification can be done in various ways, depending on the problem you're trying to solve. Perhaps you want to improve customer response times by routing your tickets to the correct teams as quickly as possible. Using text classifiers, you can tag tickets as and when they drop into your help desk. For example, you might categorize tickets by language, topic, or specific channel (Twitter, email, live chat, etc.), and route them to the correct team member or department based on these tags. Let's say you have a ticket from a customer who is asking for a refund. A topic classifier would tag this ticket as Refunds, in which case it would be sent to the accounts department. You could also use an urgency detection model, which can flag urgent tickets by analyzing their content for expressions such as right away, immediately or ASAP, and route these to teams that deal with urgent tickets.

Jupyter Notebooks Description

The project notebooks divided in to 7 parts. The description of each notebook is as below;

1. **Part 1/6: Pre-Processing, Data Visualization and EDA**
    - Exploring the given Data files
    - Understanding the structure of data
    - Missing points in data
    - Finding inconsistencies in the data
    - Visualizing different patterns
    - Visualizing different text features
    - Dealing with missing values
    - Text preprocessing
    - Creating word vocabulary from the corpus of report text data
    - Creating tokens as required
2. **Part 2/6 : Word Cloud, Topic Modelling & N-Grams, Word Embedding & Vectorization**
3. **Part 3/6 : This part deals with data resampling as the target class is highly imbalanced as analyzed in Part 1/6.**
4. **Part 4/6 : Model Building - Deep Learning**
5. **Part 5/6 : Model Building with Machine Learning Models**
6. **Part 6/6 : Hyper Parameter Tuning for Machine Learning Models**
7. **BERT Model**

Exploratory Data Analysis

- **Dataset**

Dataset Structure (4 Variables, 8500 Incident Records): Caller, Short Description, Description and Assignment Group. Data Features: Caller (2950 unique count), Short Description, Description & Label/Target Class: Assignment Group (74)

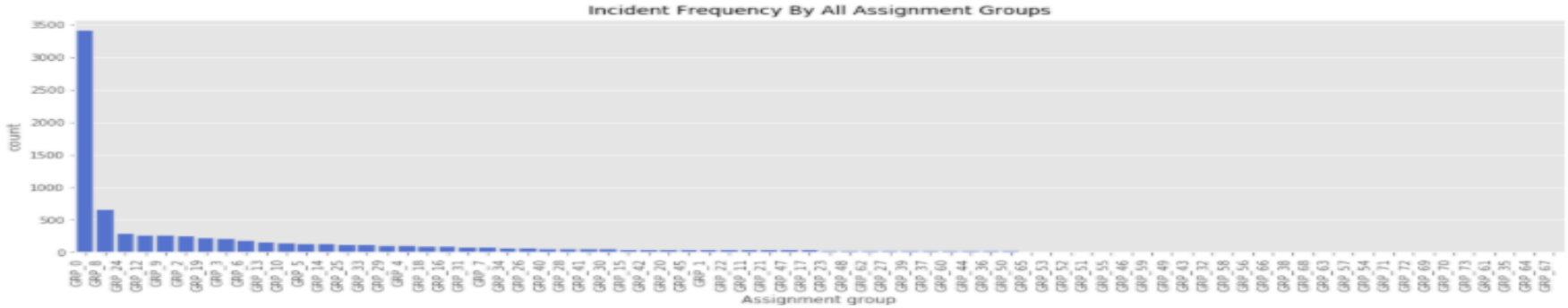
- **Features**

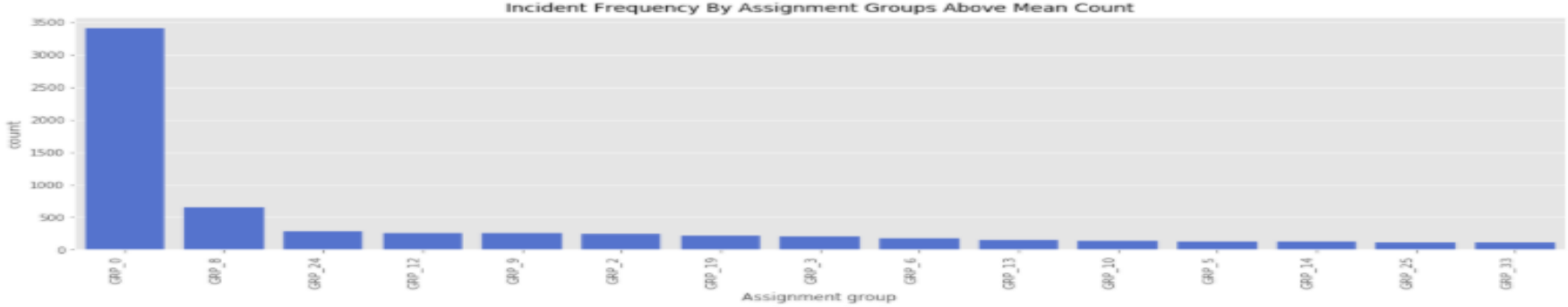
Caller names are ad-hoc and without proper references to any master data. It is not advised to be used as a feature as it will amount to overfitting the data. We expect these user or call names to be anything in the current scope of themes as making a prediction or classification of a ticket based on the caller names is not justifiable rationally. We plan to consider Short Description, Description as the data for feature engineering and ignore Caller related information in the dataset. Description are predominantly in English with words from other European languages as well. We plan to keep the data unmodified as received from the source. However for the sake of experimentation, we plan to detect the languages and try to convert certain words into English. Descriptions are unstructured akin to reviews, chats, e-mails or tweets and its not clean. There are for instance special characters, emails, dates, symbols, hyper links, URL, IP Addresses and missing values or excessive whitespaces and stop words. We plan to remove the stop words and unnecessary noise through regular expression before vectorization of the feature data. We used information in the Short Description along with the information in the Description column by concatenation of the data. Descriptions in some cases are though same as the data in the Short Description. Description contains conjugated words i.e. they are not properly separated, and the text is submitted without prior spell corrections (spelling errors) including the fact that there are descriptions that contain sentences which are not grammatically obvious or otherwise meaningful to be analyzed through human intervention.

- **Labels/Target Class**

The label data provided in the target class is imbalanced and skewed. Most of the tickets (3976, almost 50%) are being assigned to a single group (GRP_0). There are also assignment groups which have been assigned only a single ticket in the given dataset. We are earlier planned to merge these smaller groups into a single group to reduce the imbalance. This could have solved the imbalance problem. However, we are avoiding add-on group creations as we want the models to predict on the feature data as it is because these assignment groups are dynamic data and would keep changing. Hence using group names as a basis of reclassification will only serve a temporary purpose and even otherwise, it will lead to overfitting of the data. Hence, we have kept the data as it is and addressed the imbalance through methods like SMOTE and RESAMPLING. In some cases the number of incident records are less than six in a target class and hence SMOTE can not be applied as it is. We need a suitable combination of sampling methods to prepare the dataset which is balanced and right one to be fed to the models.

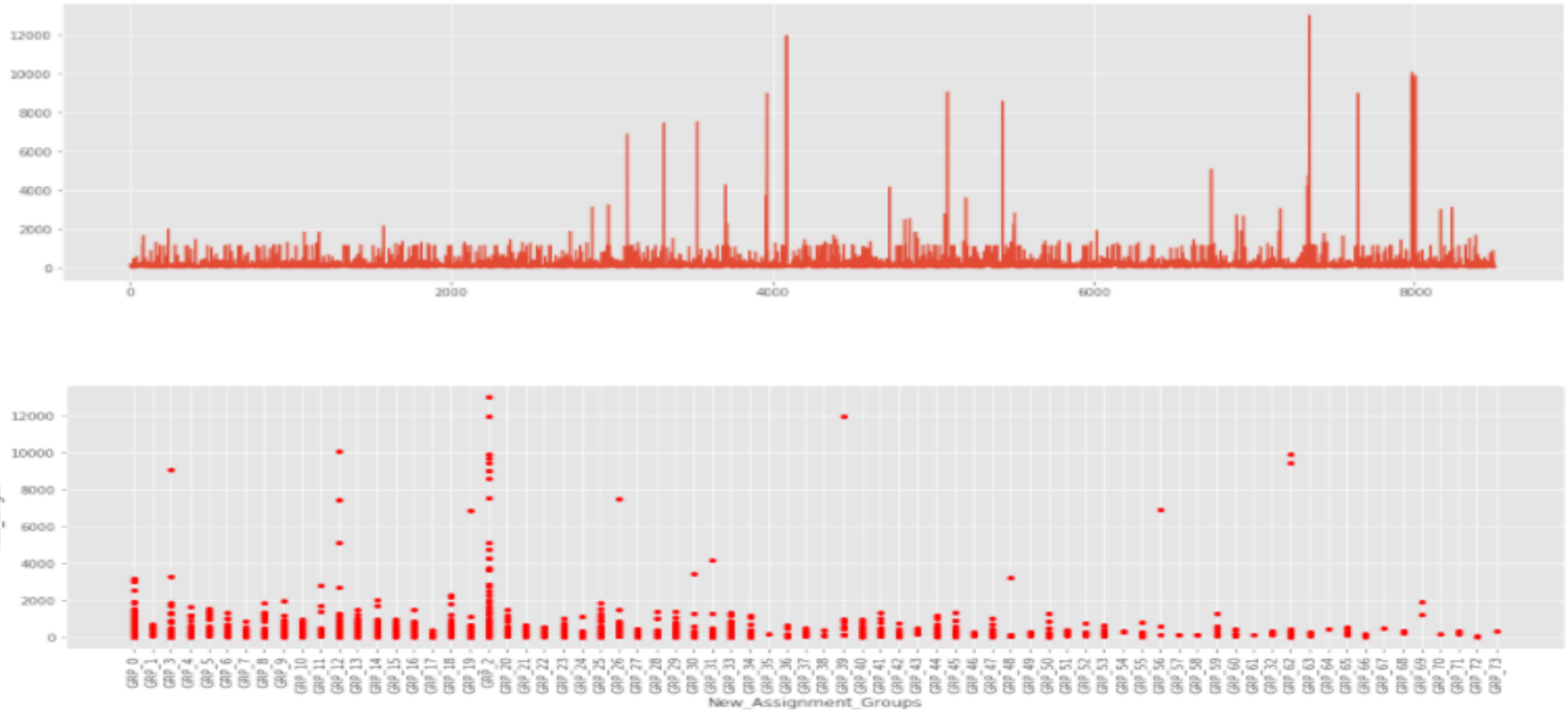
Incident Frequency By Assignment Groups (Imbalanced Multiclass Dataset)





Data Pre-Processing & Input Dataset

Feature Set : We have dropped the Caller information and concatenated the two description columns. Also there were 25 assignment groups having less than 10 incidents. However we not regrouped them and kept the assignment groups intact as provided in the dataset and planned to used resampling methods to make them balanced. We also measured the length of the text used in the incidents across the assignment groups. The length varied from 1 (after the null value being replaced with a space) to 13001 words with an average length of 217 words. There were 5 incidents with less than 3 words. We kept them as included in the dataset as an input to the models.



Data Pre-Processing & Input Dataset

Missing Values & Duplicates : We have dropped duplicate records as they don't additional information or value for classification. We have replaced the null values a space. There were 83 duplicate records across all columns, 682 duplicate records by the value in Description column and 661 duplicate records having same values in Description and Assignment Group together. This basically means there were 27 incidents which were same but were assigned to different assignment group. After removing duplicate records we had a dataset reduced from 8500 to 7839. There were 5 incidents without any short description and 1 incident without a description. We converted these null values into spaces.

Upper- & Lower-Case Text: We have converted the feature set in lower case as this helps simplify addressing the cleaning function through regular expression. Otherwise we would land up building more and more filtering criteria to account for the two different cases. Also the feature data-based prediction as such does not influence the classification methods in any way.

Special Characters & Stop Words : Stop words are words that do not add to the meaning of the text. These are filler words, or casually inserted words that add noise to the text and only complicate the text. NLP includes processes to identify and skip these when trying to understanding the text. We have removed various characters like hashtags and kept many of them in the concatenated feature set to subject to the cleaning using regular expression. We planned to use the NLTK for removal of stop words.

Language & Spelling Errors & Lemmatization: Lemma refers to the common part or the basic form of related words used in different grammatical forms. For example, the words 'greater' and 'greatly' will have 'great' as the lemma. Many texts or sentences may contain the root word as well as its different grammatical forms. NLP figures out the lemma from each token (word) and uses it to identify the words, categories them and find their meaning. From language and spell check perspective even though we have defined the steps for the same, but we have kept the text as it is for a feed to the model. However for experimentation and learning perspective we have tried to translate some of the words from non-English to English. We have also tried correcting some of the typical spelling errors. We have then used lemmatization on the processed or clean text.

Word Cloud & Topic Modeling & N-Grams: We have used word cloud to visualize the frequently existing words and have also performed topic modeling (using Gensim LDA Model) to understand the correlated words in the text across the incident reports i.e. grouping the tickets or incidents based on the words used in the text. We have also tried to plot the number of words used in each of the tickets to study the outliers.

Word Embedding & Vectorization: We plan to use N-Grams (bi and tri) from the bag of words along with Word2Vec and Glove Embedding for vectorization and preparation of dataset.

B

Project Execution Phase

3

Dataset Preparation

Data Preparation

Once the data has been collected, it is in a raw form. It is collected from various sources and could have many duplicate entries or typographical errors. There could be missing data too or this could be present in different formats. With an uncleaned dataset, no matter what type of algorithm we try, we will never get accurate results. Hence it has to be cleaned and label as accurately as possible. This involves exploration as well as pre-processing. The steps and techniques for data pre-processing varies for each dataset. Here at a higher level we are identifying relevant data within the data set and removing irrelevant data, fixing irregular cardinality and structural errors, identifying outliers (skewing the dataset) and missing data (sparse values) and treatment of the data using transformation is done . Post this data pre-processing, we done data preparation to prepare neat datasets using vectorization methods and then used them in the model's starting phase of data splitting into further dataset generation for training and validation or testing purpose.

Duplicate entries Data is usually collected from many sources and joined to form one dataset. This might lead to the duplicity of observations. This might not be a problem if the observations are repeated a few times, particularly in the case of large dataset. However, if observation(s) are repeated far too many times, it could lead to an erroneous behavior. Hence, it is better to remove duplicate observations to have a cleaner dataset.

Irrelevant observations: The dataset would contain observations which might not be useful for your specific task. For example, if you were analyzing the shopping behavior of women only, you would not require observations for men in your dataset. Similarly, your data might have a whole column of data that is not useful in making predictions and could be dropped. This is a hard par in case he data sources are not managed properly and hence in order to avoid too much of time being spent in filtering relevant data from the irrelevant one (like to old and unnecessary incidents as the associated systems are not more in the production), it should be fixed at data gathering level itself.

Cardinality is the number of elements, or members, in a set of data, i.e. the number of different values there are for a particular feature. If in a dataset any feature or column had all entries made with the same type of value (or class), it will have a cardinality of 1. Then that feature or column does not have any effect of the model or predictions. Hence it is suggested that such columns be dropped. The same is the case if the variance is too low to have any impact i.e. the difference between the highest and lowest value is very small. So it is if the cardinality is 0, i.e. there are no values available in that column. The columns could have a high number of types of values (or classes) due to typo-errors or inconsistent capitalization. For example, the gender column might have many classes like male, female, m, f, M, and F, these represent only two levels — Male and Female. Such classes should be mapped to proper levels and the rest of the levels should be dropped. Whitespaces in some of the values entered, some numeral values entered as text or inconsistency in date formats could be some more reasons for an irregularly high number of classes.

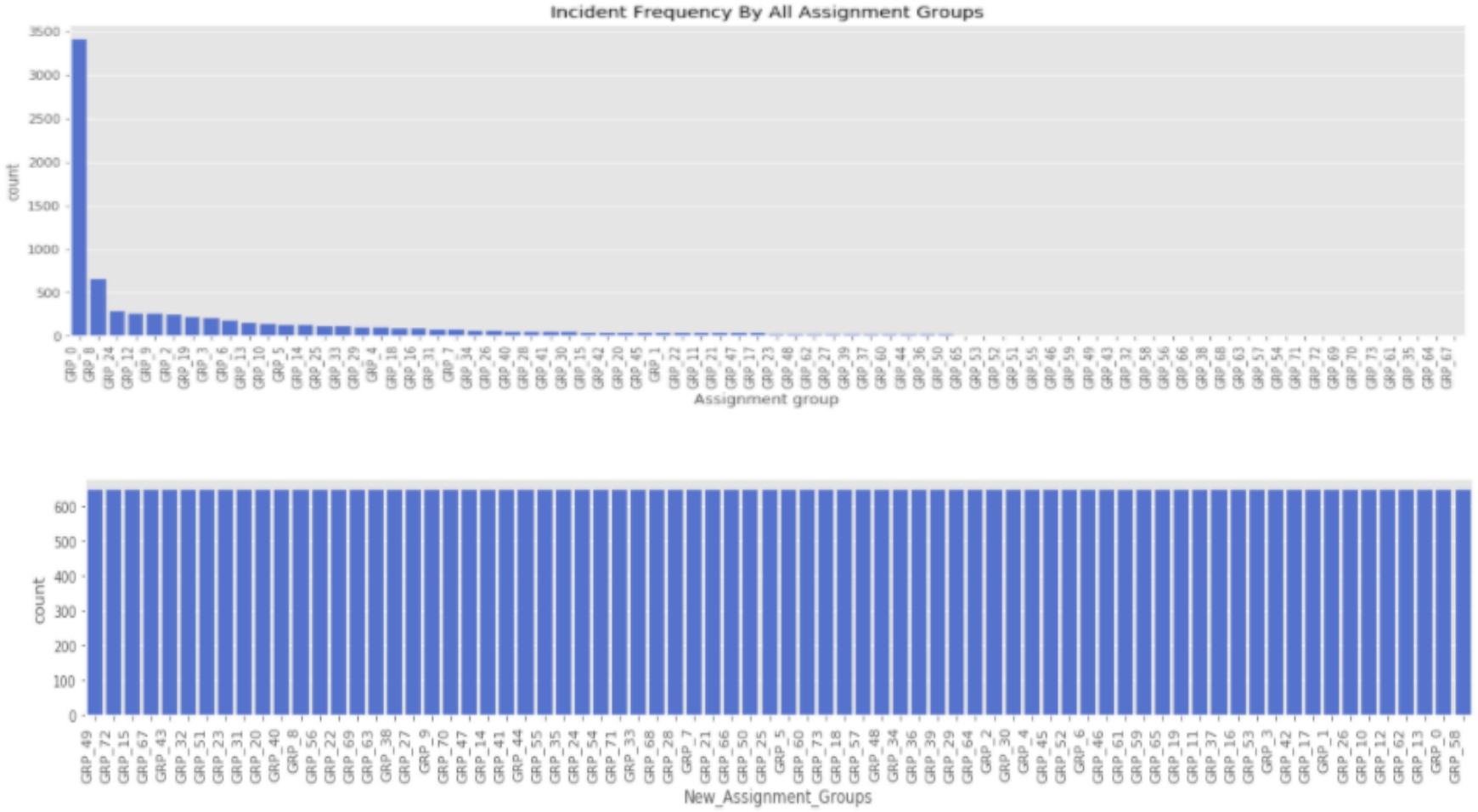
Data Pre-Processing (Text Cleaning and Class Imbalance)

The target class is imbalanced. We have many classes with less than 10 incidents. We have avoided the temptation to group such scantly populated classes in order to balance the input dataset. Instead we planned to use the resampling and SMOTE and class weights to address the target class imbalance. Our earlier plan was to build three different dataset. First to have GRP_0 and rest of the records classified into one and use two step classification model and the second one to have non-GRP_0 groups resampled for the second order classification and third dataset in which we keep all the groups (74) as it is and address the imbalance issue. After due considerations and discussions, we finally decided to pursue only the third dataset so that we don't tend to force fit the data into model. In future, these group names could be changed any time and fulcrum of imbalance could shift to some other group. In such a situation, this hard-coded approach as followed in case of the first two datasets, we fail to deliver the right classification. It could fail miserably. However, all the data related views are presented below.

We created a function to clean the dataset using regular expressions after converting the feature data into lower case. We removed various unwanted text or characters as identified noise in the information. We initially thought to use translation and spell correction function as well as there were many non-English words and typing errors. However we decided not to touch the actual data and leave these actions to be undertaken at the source. At the same time converting the data from one form to another, won't have any significant impact on the classification model. At the same time, we planned to use BERT and try the same dataset with it compare the performance of the models on the uniform or same dataset. We also did not filter the records having less than certain threshold value. However we analyzed the same and the details are provided below. The code used a variable to define this threshold value and hence could be used in future in case one wants to filter the records based on the number of words in the feature data. We used NLTK to remove the stop words and later analyzed the same to see these are not appearing the word cloud.

Lemmatization was carried out subsequently before tokenization for vectorization using word embeddings. We limited ourselves to use lemmatization instead of stemming based on the quality of the data to avoid further dilution of the input feeds. Spacy was used for lemmatization. PyLDAvis was used to plot the topics and analyze the texts using N-Gram models for clustering relevant data using Gensim. We also restricted usage of spelling errors on the same ground as there are multi-lingual texts which might as well are not spelled accurately. Hence both the translation and spelling corrections might not be effective at the stage from classification perspective. Based on our research, we have found that these two aspect of language processing won't provide any substantial improvement in the model accuracy.

Data Pre-Processing (Re-Sampling for Unbalanced classes)

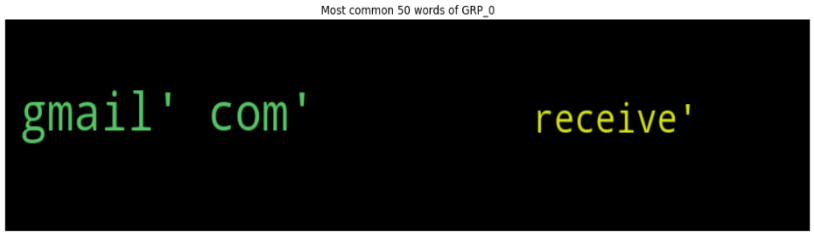


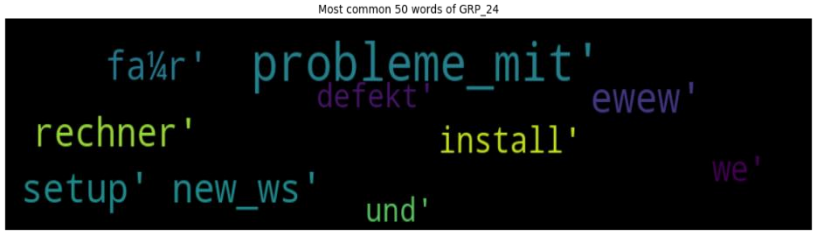
Word Cloud

We used word cloud visualization for checking if the stop words are removed effectively as well as identify the words that represent the target class based on the frequency or importance. We generated the word cloud for the groups; however we are sharing a snapshot in the report for few of them having texts with certain threshold value as set as a criteria while generating them.

BI-GRAMS

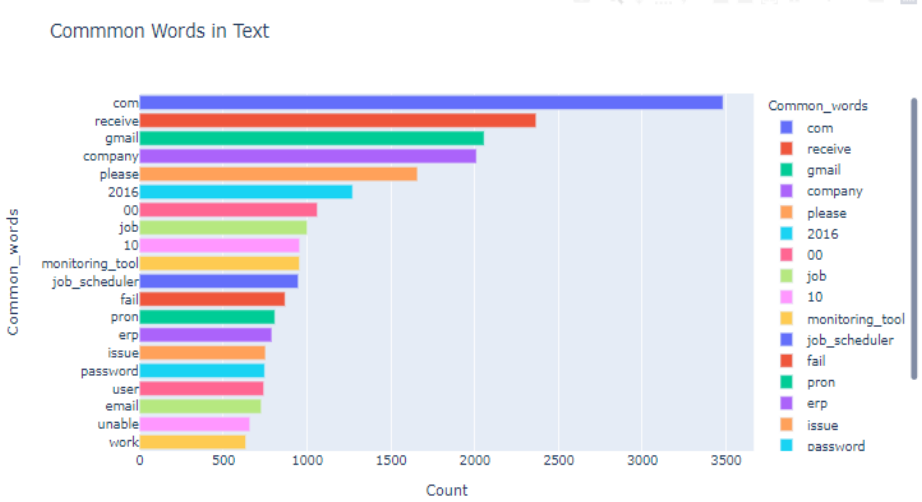
TRI-GRAMS

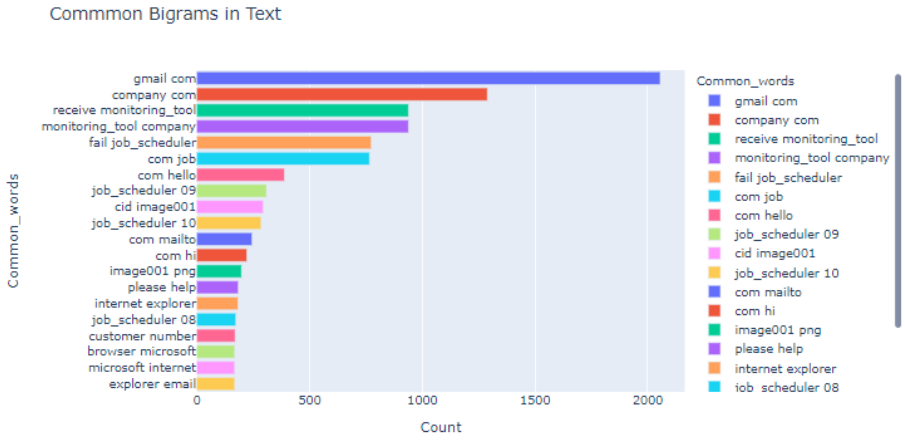


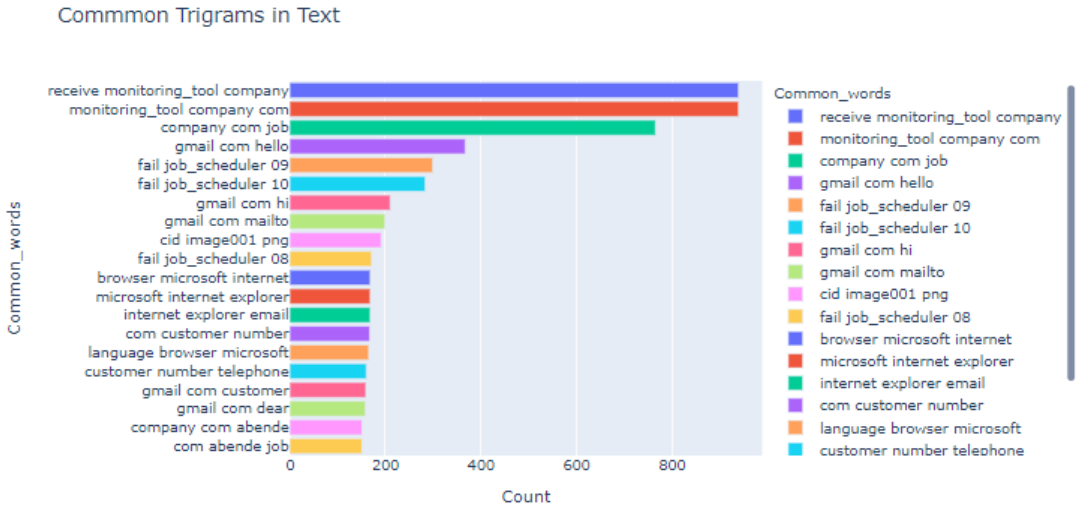


N-Gram Visualization

COMMON WORDS, BI-GRAMS & TRI-GRAMS



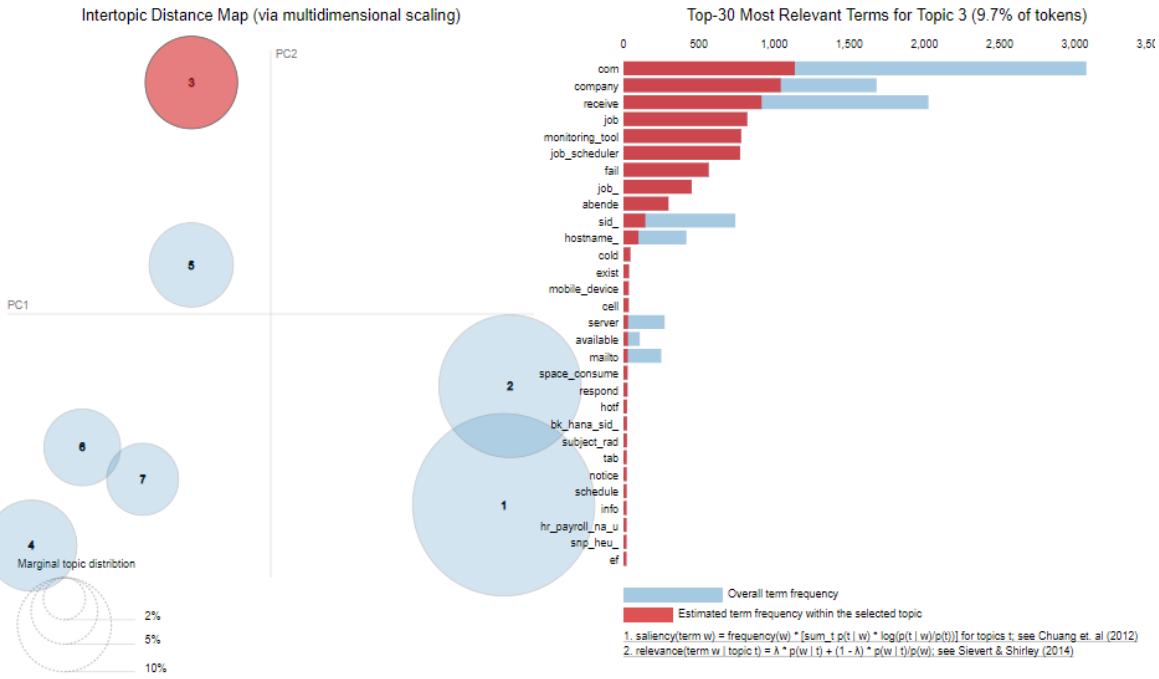




Topic Modeling

*Typically in a text domain, EDA can have many meanings: What are the topics? How frequent they are? The process will involve some level of preprocessing steps. We analyzed the incidents based on the number of words present in them. We planned to extend its utility in dimensionality reduction in next iteration. In natural language processing, latent Dirichlet allocation (LDA) is a "generative statistical model" that allows sets of observations to be explained by unobserved groups that explain why some parts of the data are similar. So this is categorized as unsupervised learning. For example, if observations are words collected into documents, it posits that each document is a mixture of a small number of topics and that each word's presence is attributable to one of the document's topics. LDA is an example of a topic model. In this project, we used the description in the dataset to generate topics. In this way, we can know about what users are typically raising incidents about and what they are concerned about or focusing on, and perhaps where one should make progress or priority. A model with higher log-likelihood and lower perplexity (exp(-1. * log-likelihood per word)) is good.*

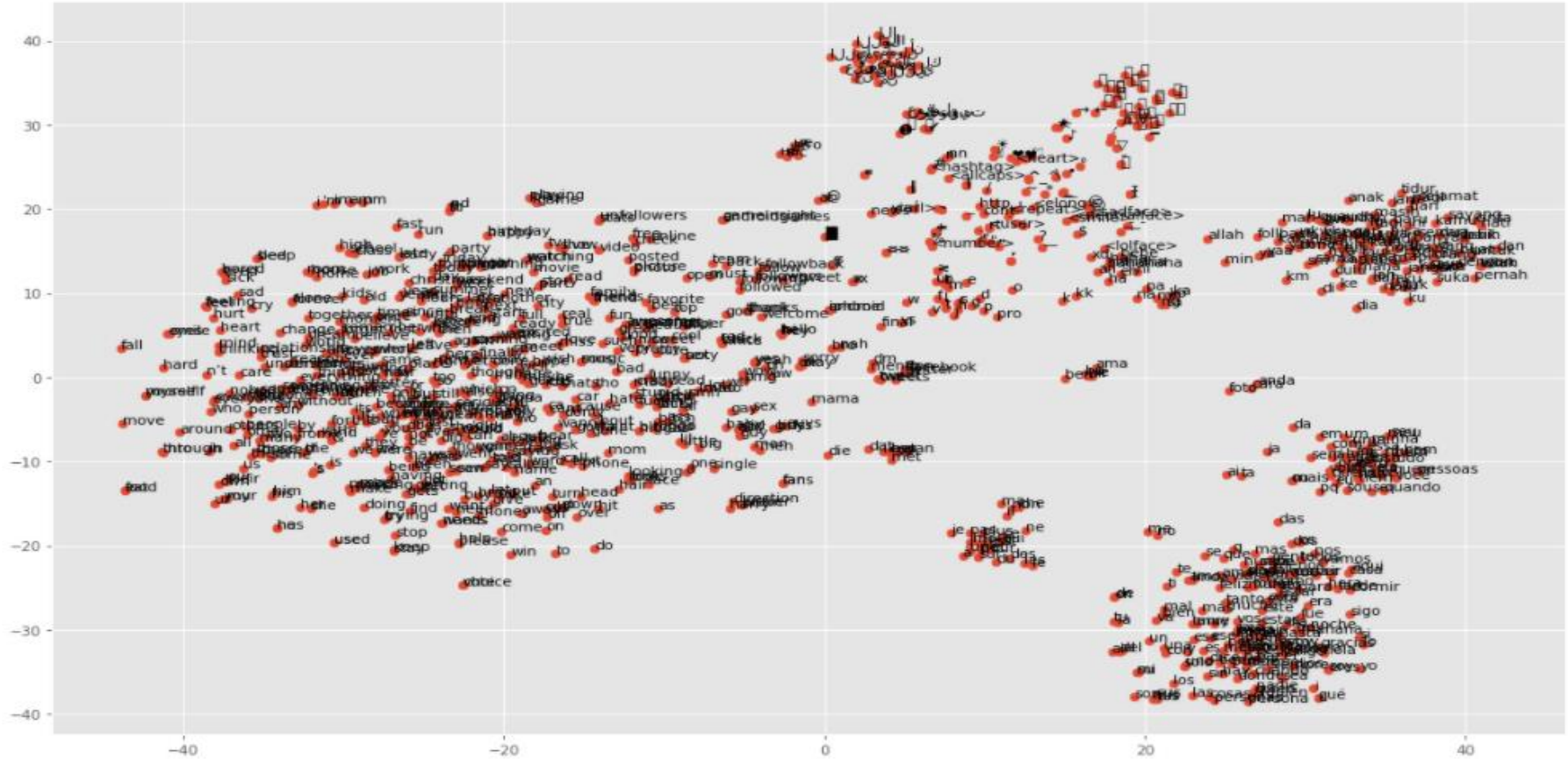
On a different note, perplexity might not be the best measure to evaluate topic models because it doesn't consider the context and semantic associations between words.



To classify a document as belonging to a particular topic, a logical approach is to see which topic has the highest contribution to that document and assign it. In current scenario, the number of topic as 3 are most non-overlapping for the incident text i.e. if the count is increased further, there are certain incidents which will start showing propensity to belong to multiple topics. Topic modelling can help identify how the incidents are being drafted and how close the incidents are in terms of the usage of words to describe them and hence these topics can give an idea about the optimal number of assignment groups that can form to attain an improved classification using the ML/DL models. The topics can be labelled based on the problem domain or the type of the incidents with inputs from the domain team. At the same time, we suggest having more features like application name, user role, location etc. to further improve the accuracy of the classification.

Word Embedding

Word Embedding is the efficient way for text data vectorization, and we have used both Word2Vec and GloVe embedding to experiment with the models. The first one is shallow two-layer NN trained on large corpus of text to produce a vector for a text (each word) in the corpus. GloVe is unsupervised learning-based vector representation of words. The training is performed on an aggregated global word-word co-occurrence statistics from a corpus and the resulting representations have linear substructures of the words in the vector space. We have explored both the types of embedding with LSTM model. We have not encountered significant difference in terms of its impact on the model performance.



4

Model Development

Model Selection & Development

Natural Language Processing (NLP) is the subfield of AI that aims to develop the ability of computers to analyze, understand, process and generate human language, including speech for various purposes. The subsequent stage of NLP may be seen as natural language interaction, where humans can communicate with computers using everyday language, intelligently. For now and as per the given scope limited to classification, we have planned to use both Deep Learning and Machine Learning models for classifying the incidents by assignment groups with the existing and same dataset after suitable cleaning and vectorization.:

Deep Learning: Deep Learning uses massive neural networks with many layers of processing units, relying on considerable computing power along with improved training techniques to learn complex patterns in large amounts of data. Typical applications include computer vision, natural language processing and speech recognition.

- *Bidirectional LSTM – With Word2Vec and Glove Embedding*
- *GRU*
- *RNN*
- *BERT*

Machine Learning Models: Machine Learning is the part of AI that involves automating analytical model building. It uses various methods such as neural networks, statistics, operations research and physics in its attempt to find hidden insights in given data without explicitly being programmed for what conclusions to reach and the path to follow.

- *Random Forest Classifier – With and Without Class Weight*
- *Support Vector Machine*
- *Naïve Bayes (MultinomialNB and ComplementNB)*
- *AdaBoost*
- *Bagging*
- *GradientBoost*
- *KNN*
- *XGBoost*
- *CatBoost*
- *Logistic Regression*

5

Model Evaluation with Classification Report

Deep Learning Models:

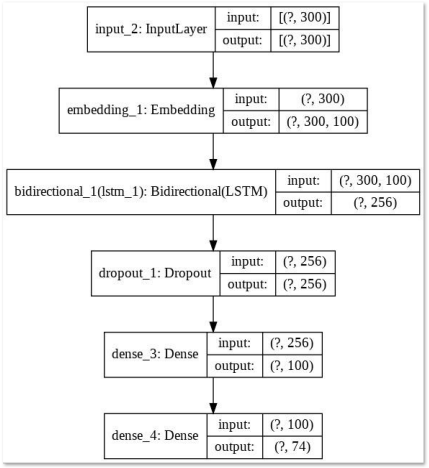
Bi-Directional LSTM Model [Word2Vec]
Bi-Directional LSTM Model [GloVe]
GRU
RNN
BERT

Bi-Directional LSTM Model [Word2Vec]

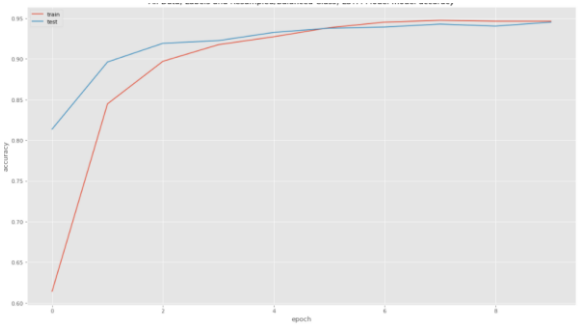
LSTM stands for "long short-term memory" and it's a type of recurrent unit that has become very popular in recent years due to its superior performance and the fact that it doesn't as easily succumb to the vanishing gradient problem.

Bi-Directional LSTM are a variant of traditional LSTM but improve the model performance by using the training inputs (texts/words) in a sequence (sentence) first as a normal sentence and then reading the tokens in the sentence in a reverse order. In short, for every token, in the sentence, there is a data about the tokens ahead as well as behind it with a memory parameters as defined in the model. This improves the effectiveness of the classification.

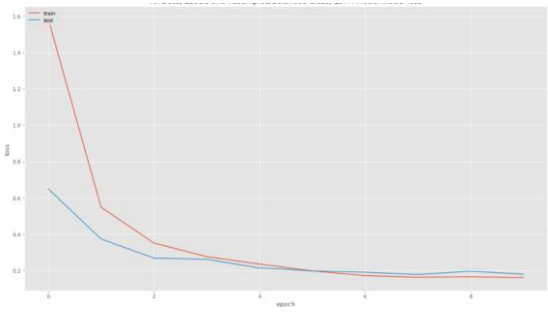
LSTMs help preserve the error that can be backpropagated through time and layers. By maintaining a more constant error, they allow recurrent nets to continue to learn over many time steps (over 1000), thereby opening a channel to link causes and effects remotely. This is one of the central challenges to machine learning and AI, since algorithms are frequently confronted by environments where reward signals are sparse and delayed, such as life itself.




```
Number of Samples: 47730
Number of Labels:  47730
Number of train Samples: 38184
Number of val Samples: 9546
Model: "functional_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_5 (InputLayer)         [(None, 300)]             0
_________________________________________________________________
embedding_4 (Embedding)      (None, 300, 100)          900100
_________________________________________________________________
bidirectional_4 (Bidirection (None, 256)               234496
_________________________________________________________________
dropout_4 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_8 (Dense)              (None, 100)               25700
_________________________________________________________________
dense_9 (Dense)              (None, 74)                7474
=================================================================
Total params: 1,167,770
Trainable params: 1,167,770
Non-trainable params: 0
```



All Data, Labels and Resampled/Balanced Class) LSTM Model model accuracy



All Data, Labels and Resampled/Balanced Class) LSTM Model model loss

Bi-Directional LSTM Model [Word2Vec]

– Classification report

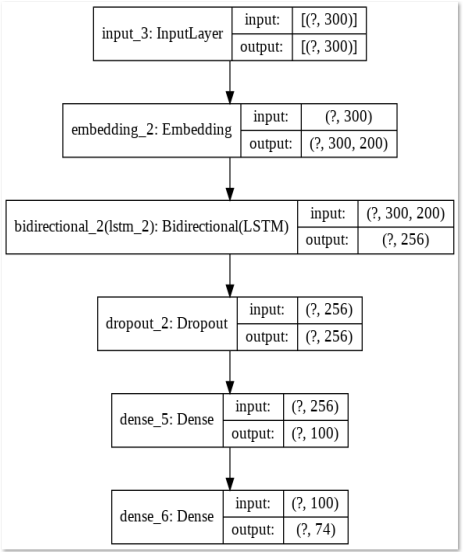
class	precision	recall	f1-score	support
0	0.89	0.53	0.66	110
1	0.88	1.00	0.94	124
2	0.99	0.77	0.87	142
3	0.98	1.00	0.99	131
4	0.89	0.92	0.91	117
5	0.98	0.93	0.96	128
6	0.99	0.98	0.98	133
7	1.00	1.00	1.00	130
8	0.97	1.00	0.99	139
9	1.00	1.00	1.00	128
10	0.96	0.97	0.97	145
11	0.92	0.94	0.93	128
12	0.93	0.91	0.92	111
13	0.98	1.00	0.99	119
14	0.99	1.00	1.00	130
15	0.98	1.00	0.99	134
16	0.99	1.00	1.00	114
17	0.99	0.98	0.99	140
18	0.98	1.00	0.99	131
19	0.98	0.98	0.98	128
20	0.98	1.00	0.99	134
21	0.95	1.00	0.98	141
22	0.98	0.98	0.98	140
23	0.96	0.94	0.95	140
24	0.91	0.84	0.87	127
25	0.97	0.85	0.91	135
26	1.00	1.00	1.00	126
27	0.97	0.97	0.97	125
28	0.98	0.97	0.97	135
29	1.00	1.00	1.00	143
30	0.99	1.00	1.00	133
31	0.99	1.00	1.00	122
32	1.00	1.00	1.00	122
33	0.98	0.95	0.96	111
34	0.98	0.93	0.95	132
35	0.93	1.00	0.96	123

class	precision	recall	f1-score	support
36	1.00	1.00	1.00	111
37	1.00	1.00	1.00	134
38	1.00	1.00	1.00	143
39	1.00	0.93	0.96	130
40	0.99	0.85	0.91	123
41	1.00	1.00	1.00	144
42	0.75	0.98	0.85	131
43	0.88	1.00	0.93	133
44	1.00	1.00	1.00	123
45	0.87	0.61	0.72	120
46	0.99	1.00	1.00	134
47	1.00	1.00	1.00	118
48	1.00	1.00	1.00	142
49	1.00	1.00	1.00	129
50	1.00	1.00	1.00	133
51	1.00	1.00	1.00	114
52	1.00	1.00	1.00	105
53	1.00	0.46	0.63	114
54	1.00	1.00	1.00	146
55	1.00	1.00	1.00	118
56	0.94	0.50	0.65	137
57	0.97	0.79	0.87	120
58	1.00	1.00	1.00	122
59	0.99	1.00	1.00	132
60	1.00	1.00	1.00	122
61	1.00	1.00	1.00	116
62	0.99	1.00	1.00	133
63	1.00	1.00	1.00	121
64	1.00	1.00	1.00	130
65	1.00	1.00	1.00	144
66	0.93	1.00	0.96	136
67	0.98	1.00	0.99	120
68	1.00	1.00	1.00	126
69	1.00	1.00	1.00	142
70	0.97	1.00	0.99	134
71	1.00	1.00	1.00	134
72	0.86	0.55	0.68	146
73	0.28	0.85	0.43	124
accuracy			0.95	9535
macro avg	0.96	0.94	0.95	9535
weighted avg	0.96	0.95	0.95	9535

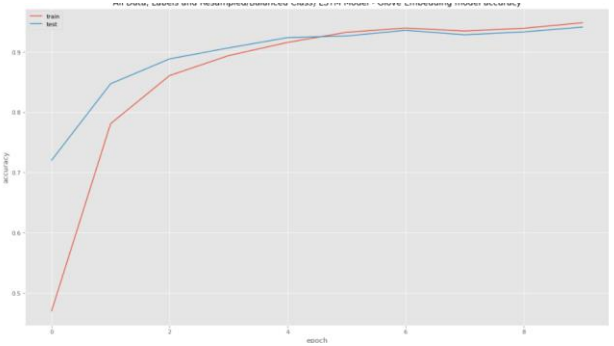
Bi-Directional LSTM Model [GloVe]

LSTMs contain information outside the normal flow of the recurrent network in a gated cell. Information can be stored in, written to, or read from a cell, much like data in a computer's memory. The cell makes decisions about what to store, and when to allow reads, writes and erasures, via gates that open and close. Unlike the digital storage on computers, however, these gates are analog, implemented with element-wise multiplication by sigmoids, which are all in the range of 0-1. Analog has the advantage over digital of being differentiable, and therefore suitable for backpropagation.

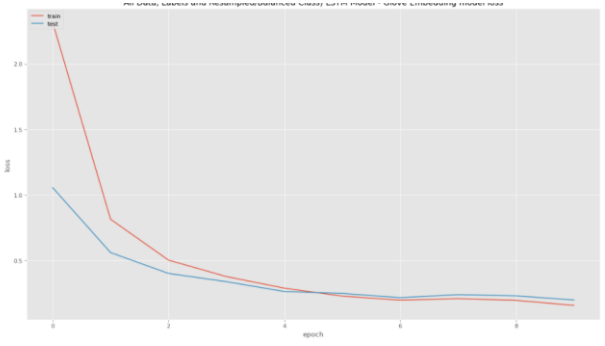
Those gates act on the signals they receive, and like the neural network's nodes, they block or pass on information based on its strength and import, which they filter with their own sets of weights. Those weights, like the weights that modulate input and hidden states, are adjusted via the recurrent networks learning process. That is, the cells learn when to allow data to enter, leave or be deleted through the iterative process of making guesses, backpropagating error, and adjusting weights via gradient descent. It should be noted that while feedforward networks map one input to one output, recurrent nets can map one to many, as above (one image to many words in a caption), many to many (translation), or many to one (classifying a voice).




```
Number of Samples: 47730
Number of Labels:  47730
Number of train Samples: 30547
Number of val Samples: 7637
Model: "functional_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_6 (InputLayer)         [(None, 300)]             0
_________________________________________________________________
embedding_5 (Embedding)      (None, 300, 200)          1800200
_________________________________________________________________
bidirectional_5 (Bidirection (None, 256)               336896
_________________________________________________________________
dropout_5 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_10 (Dense)             (None, 100)               25700
_________________________________________________________________
dense_11 (Dense)             (None, 74)                7474
=================================================================
Total params: 2,170,270
Trainable params: 2,170,270
Non-trainable params: 0
```



All Data, Labels and Resampled/Balanced Class) LSTM Model - Glove Embedding model accuracy



All Data, Labels and Resampled/Balanced Class) LSTM Model - Glove Embedding model loss

LSTM- GLOVE Classification report

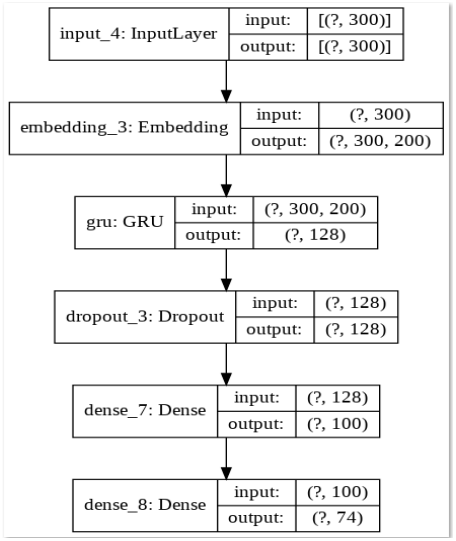
class	precision	recall	f1-score	support
0	0.79	0.56	0.65	135
1	0.88	1.00	0.94	144
2	0.99	0.81	0.89	126
3	0.97	1.00	0.99	133
4	0.97	0.82	0.89	127
5	0.96	0.92	0.94	131
6	0.97	0.96	0.97	120
7	1.00	1.00	1.00	141
8	0.95	0.98	0.97	115
9	0.99	1.00	1.00	123
10	0.99	0.97	0.98	114
11	0.93	0.93	0.93	147
12	0.91	0.84	0.87	138
13	1.00	0.95	0.98	151
14	0.98	1.00	0.99	120
15	0.99	1.00	1.00	114
16	0.99	1.00	0.99	138
17	1.00	0.91	0.96	129
18	0.98	0.97	0.98	147
19	0.96	1.00	0.98	128
20	0.98	1.00	0.99	130
21	0.95	1.00	0.97	115
22	0.95	0.96	0.96	131
23	0.91	0.92	0.92	133
24	0.96	0.86	0.90	111
25	0.92	0.94	0.93	138
26	1.00	1.00	1.00	119
27	0.95	0.97	0.96	143
28	0.92	0.96	0.94	124
29	0.99	1.00	1.00	128
30	1.00	1.00	1.00	143
31	1.00	1.00	1.00	132
32	1.00	1.00	1.00	126
33	0.98	1.00	0.99	125
34	0.97	0.89	0.93	131
35	0.98	1.00	0.99	123
36	0.97	1.00	0.98	128
37	0.98	1.00	0.99	124
38	1.00	1.00	1.00	130
39	1.00	0.94	0.97	111
40	0.94	0.82	0.88	136
41	0.99	1.00	1.00	144
42	0.94	0.79	0.86	112
43	0.92	1.00	0.96	132
44	1.00	1.00	1.00	154
45	0.92	0.61	0.74	140
46	0.98	1.00	0.99	118
47	1.00	1.00	1.00	135
48	0.99	1.00	1.00	141
49	1.00	1.00	1.00	112
50	0.99	1.00	1.00	119
51	0.98	1.00	0.99	129
52	1.00	1.00	1.00	130
53	0.33	1.00	0.50	125
54	1.00	1.00	1.00	147
55	1.00	1.00	1.00	139
56	0.92	0.55	0.69	130
57	0.96	0.87	0.91	123
58	1.00	1.00	1.00	107
59	0.99	1.00	0.99	137
60	1.00	1.00	1.00	127
61	1.00	1.00	1.00	119
62	0.98	1.00	0.99	127
63	1.00	1.00	1.00	143
64	1.00	1.00	1.00	126
65	1.00	1.00	1.00	134
66	0.99	1.00	1.00	125
67	0.98	1.00	0.99	122
68	1.00	1.00	1.00	112
69	1.00	1.00	1.00	126
70	0.94	1.00	0.97	124
71	1.00	1.00	1.00	132
72	0.77	0.55	0.64	126
73	0.58	0.45	0.51	116
accuracy			0.94	9535
macro avg	0.96	0.94	0.94	9535
weighted avg	0.96	0.94	0.94	9535

GRU

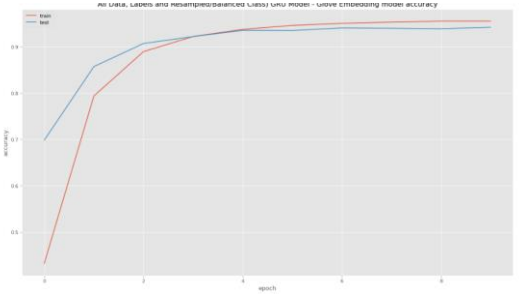
GRU (Gated Recurrent Unit) aims to solve the vanishing gradient problem which comes with a standard recurrent neural network. It can also be considered as a variation on the LSTM because both are designed similarly and, in some cases, produce equally excellent results. The GRU is like a long short-term memory (LSTM) with a forget gate but has fewer parameters than LSTM, as it lacks an output gate. A GRU has two gates, an LSTM has three gates.

This model got rid of the cell state and used the hidden state to transfer information. It uses only 2 gates, update gate and reset gate. Basically, these are two vectors which decide what information should be passed to the output. The update gate acts like the forget and input gate of an LSTM. It decides what information to throw away and what new information to add. The reset gate is another gate used to decide how much past information to forget. GRU's has fewer tensor operations; therefore, they are a little speedier to train than LSTM's.

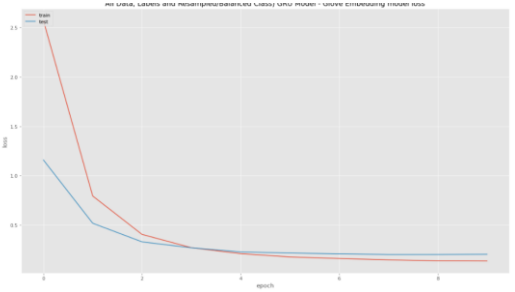
Using word embeddings such as word2vec and GloVe is a popular method to improve the accuracy of your model. Instead of using one-hot vectors to represent our words, the low-dimensional vectors learned using word2vec or GloVe carry semantic meaning – similar words have similar vectors. Using these vectors is a form of pre-training. Intuitively, you are telling the network which words are similar so that it needs to learn less about the language. Using pre-trained vectors is particularly useful if you don't have a lot of data because it allows the network to generalize to unseen words.




```
Number of Samples: 47730
Number of Labels:  47730
Number of train Samples: 30547
Number of val Samples: 7637
Model: "functional_15"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_8 (InputLayer)         [(None, 300)]             0
_________________________________________________________________
embedding_7 (Embedding)      (None, 300, 200)          1800200
_________________________________________________________________
gru_1 (GRU)                  (None, 128)               126720
_________________________________________________________________
dropout_7 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_14 (Dense)             (None, 100)               12900
_________________________________________________________________
dense_15 (Dense)             (None, 74)                7474
=================================================================
Total params: 1,947,294
Trainable params: 1,947,294
Non-trainable params: 0
_________________________________________________________________
```



All Data, Labels and Resampled/Balanced Class) GRU Model - Glove Embedding model accuracy



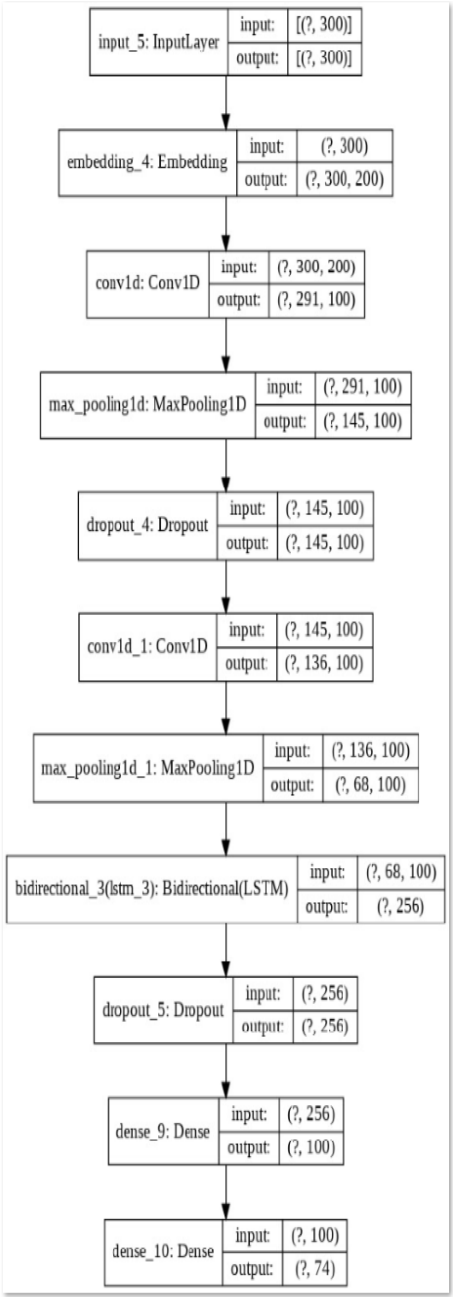
All Data, Labels and Resampled/Balanced Class) GRU Model - Glove Embedding model loss

GRU- Classification Report

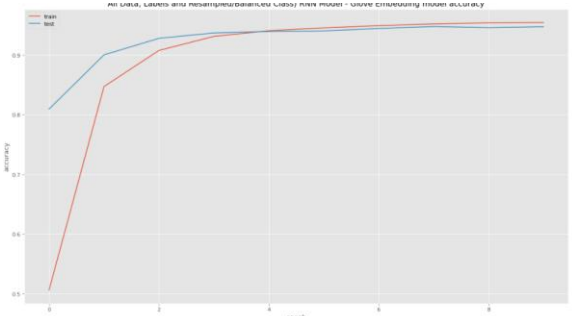
class	precision	recall	f1-score	support
0	0.91	0.54	0.68	132
1	0.86	1.00	0.93	124
2	0.98	0.76	0.86	150
3	0.98	1.00	0.99	122
4	0.87	0.78	0.82	116
5	0.96	0.94	0.95	134
6	0.97	0.96	0.97	131
7	0.99	1.00	1.00	129
8	0.93	1.00	0.97	115
9	0.99	1.00	1.00	129
10	0.96	0.94	0.95	128
11	0.90	0.94	0.92	118
12	0.83	0.87	0.85	126
13	0.99	0.99	0.99	142
14	0.97	1.00	0.98	152
15	0.99	1.00	1.00	130
16	0.98	1.00	0.99	131
17	0.99	0.97	0.98	124
18	0.93	0.98	0.95	129
19	0.97	1.00	0.98	129
20	1.00	1.00	1.00	121
21	0.96	1.00	0.98	122
22	1.00	0.93	0.96	130
23	0.96	0.90	0.93	122
24	0.92	0.90	0.91	126
25	0.95	0.94	0.95	135
26	1.00	1.00	1.00	124
27	0.96	0.99	0.97	133
28	0.96	0.96	0.96	129
29	1.00	1.00	1.00	132
30	1.00	1.00	1.00	140
31	1.00	1.00	1.00	126
32	1.00	1.00	1.00	120
33	0.97	1.00	0.98	119
34	0.94	0.92	0.93	128
35	0.99	1.00	1.00	132
36	0.99	1.00	1.00	129
37	0.99	1.00	1.00	109
38	1.00	1.00	1.00	140
39	0.97	0.91	0.94	117
40	0.97	0.83	0.90	131
41	1.00	1.00	1.00	123
42	0.75	0.97	0.85	141
43	0.90	1.00	0.95	134
44	1.00	1.00	1.00	133
45	0.94	0.59	0.72	134
46	0.96	1.00	0.98	124
47	1.00	1.00	1.00	137
48	1.00	1.00	1.00	120
49	0.99	1.00	1.00	146
50	1.00	1.00	1.00	139
51	0.99	1.00	1.00	126
52	1.00	1.00	1.00	142
53	0.32	1.00	0.49	120
54	1.00	1.00	1.00	122
55	1.00	1.00	1.00	133
56	0.82	0.51	0.63	110
57	0.97	0.85	0.90	131
58	1.00	1.00	1.00	127
59	0.99	1.00	1.00	127
60	1.00	1.00	1.00	153
61	1.00	1.00	1.00	138
62	1.00	1.00	1.00	118
63	1.00	1.00	1.00	106
64	1.00	1.00	1.00	132
65	1.00	1.00	1.00	128
66	1.00	1.00	1.00	132
67	0.98	1.00	0.99	134
68	1.00	1.00	1.00	107
69	1.00	1.00	1.00	132
70	0.97	1.00	0.98	150
71	1.00	1.00	1.00	123
72	0.79	0.50	0.61	125
73	1.00	0.30	0.46	132
accuracy			0.94	9535
macro avg	0.96	0.94	0.94	9535
weighted avg	0.96	0.94	0.94	9535

RNN

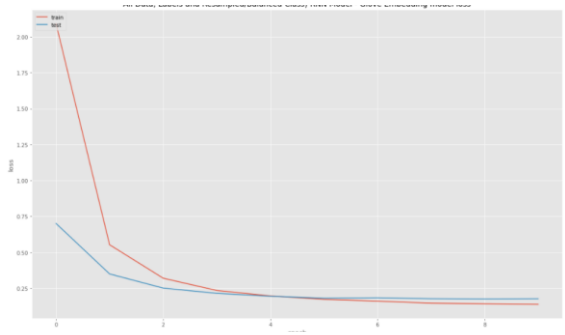
Recurrent Neural Networks (RNNs) are a family of neural networks designed specifically for sequential data processing. The RNN model does prediction of the next word in a sequence based on the previous ones. This operation is performed recurrently which is why it is called as Recurrent Neural Networks. It repetitively performs the same task for every element of a sequence, with the output being dependent on the previous computations. In short it has a memory of the text used in a sequence used for classification of the text both from the type of texts and the sequence of appearance as well. There are known inherent issues in RNN which are addressed in the subsequent model of LSTM. Basic RNNs are a network of neuron-like nodes organized into successive layers. Each node in each layer relates to a directed (one-way) connection to every other node in the next successive layer. Each node (neuron) has a time-varying real-valued activation. Each connection (synapse) has a modifiable real-valued weight. Nodes are either input nodes (receiving data from outside of the network), output nodes (yielding results), or hidden nodes (that modify the data en route from input to output). For supervised learning in discrete time settings, sequences of real-valued input vectors arrive at the input nodes, one vector at a time. At any given time step, each non-input unit computes its current activation (result) as a nonlinear function of the weighted sum of the activations of all units that connect to it. Supervisor-given target activations can be supplied for some output units at certain time steps. For example, if the input sequence is a speech signal corresponding to a spoken digit, the final target output at the end of the sequence may be a label classifying the digit.




```
Number of Samples: 1432
Number of Labels:  1432
Number of train Samples: 1145
Number of val Samples: 287
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_8 (Embedding)      (None, 300, 200)          1800200
_________________________________________________________________
conv1d (Conv1D)              (None, 291, 100)          200100
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 145, 100)          0
_________________________________________________________________
dropout_8 (Dropout)          (None, 145, 100)          0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 136, 100)          100100
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 68, 100)           0
_________________________________________________________________
bidirectional_6 (Bidirection (None, 256)               234496
_________________________________________________________________
dropout_9 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_16 (Dense)             (None, 100)               25700
_________________________________________________________________
dense_17 (Dense)             (None, 74)                7474
=================================================================
Total params: 2,368,070
Trainable params: 2,368,070
Non-trainable params: 0
```



All Data, Labels and Resampled/Balanced Class) RNN Model - Glove Embedding model accuracy



All Data, Labels and Resampled/Balanced Class) RNN Model - Glove Embedding model loss

RNN Classification Report

class	precision	recall	f1-score	support
0	0.88	0.56	0.68	131
1	0.98	0.87	0.92	125
2	0.98	0.76	0.86	121
3	1.00	1.00	1.00	123
4	0.95	0.92	0.94	126
5	0.97	0.94	0.95	125
6	0.97	0.98	0.98	128
7	1.00	1.00	1.00	142
8	0.98	1.00	0.99	130
9	1.00	1.00	1.00	122
10	0.98	0.94	0.96	125
11	0.88	0.90	0.89	137
12	0.89	0.93	0.91	118
13	0.99	1.00	1.00	141
14	1.00	1.00	1.00	118
15	0.99	1.00	1.00	129
16	0.99	1.00	1.00	134
17	1.00	0.95	0.97	122
18	0.97	1.00	0.98	143
19	0.98	1.00	0.99	132
20	1.00	1.00	1.00	127
21	0.99	1.00	1.00	124
22	0.98	0.99	0.98	127
23	0.96	0.94	0.95	141
24	0.98	0.88	0.93	120
25	0.88	0.93	0.90	117
26	1.00	1.00	1.00	131
27	0.97	0.98	0.97	122
28	0.98	0.97	0.97	125
29	1.00	1.00	1.00	120
30	1.00	1.00	1.00	112
31	0.99	1.00	1.00	146
32	1.00	1.00	1.00	139
33	1.00	1.00	1.00	136
34	0.97	0.92	0.95	131
35	1.00	1.00	1.00	123
36	0.99	1.00	1.00	124
37	1.00	1.00	1.00	130
38	1.00	1.00	1.00	126
39	1.00	0.94	0.97	136
40	0.97	0.83	0.89	117
41	0.99	1.00	1.00	131
42	1.00	0.78	0.88	132
43	0.92	1.00	0.96	145
44	1.00	1.00	1.00	127
45	0.77	0.77	0.77	127
46	1.00	1.00	1.00	135
47	1.00	1.00	1.00	145
48	1.00	1.00	1.00	112
49	1.00	1.00	1.00	138
50	1.00	1.00	1.00	137
51	0.99	1.00	1.00	130
52	1.00	1.00	1.00	119
53	0.33	1.00	0.50	133
54	1.00	1.00	1.00	125
55	1.00	1.00	1.00	130
56	0.67	0.66	0.67	153
57	0.93	0.91	0.92	113
58	1.00	1.00	1.00	122
59	0.97	1.00	0.99	117
60	1.00	1.00	1.00	128
61	1.00	1.00	1.00	140
62	0.98	1.00	0.99	122
63	1.00	1.00	1.00	135
64	1.00	1.00	1.00	131
65	1.00	1.00	1.00	135
66	1.00	1.00	1.00	122
67	0.98	1.00	0.99	111
68	1.00	1.00	1.00	123
69	1.00	1.00	1.00	141
70	0.95	1.00	0.97	133
71	1.00	1.00	1.00	131
72	0.83	0.50	0.62	124
73	1.00	0.31	0.47	142
accuracy			0.95	9535
macro avg	0.96	0.95	0.95	9535
weighted avg	0.96	0.95	0.95	9535

BERT with Classification Report

Bidirectional Encoder Representations from Transformers (BERT) is a method of pre-training language representations, meaning that we train a general-purpose "language understanding" model on a large text corpus (like Wikipedia), and then use that model for downstream NLP tasks that we care about (like question answering). BERT is designed to pretrain deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers. BERT differs from the previous methods because it is the first unsupervised, deeply bidirectional system for pre-training NLP. Unsupervised means that BERT was trained using only a plain text corpus, which is important because an enormous amount of plain text data is publicly available on the web in many languages. Tokenization, numericalization and embeddings do not differ from the way it is done with RNNs. Pre-trained representations can also either be context-free or contextual, and contextual representations can further be unidirectional or bidirectional.

Context-free models such as word2vec or GloVe generate a single "word embedding" representation for each word in the vocabulary, so bank would have the same representation in bank deposit and river-bank. 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. Contextual models instead generate a representation of each word that is based on the other words in the sentence.

BERT has inspired many recent NLP architectures, training approaches and language models, such as Google's TransformerXL, OpenAI's GPT-2, XLNet, ERNIE2.0, RoBERTa, etc.

In 2015, ResNet-50 and ResNet-100 were introduced with 23M and 45M parameters, respectively. Fast forward to 2018, the BERT-Large model has 330M parameters. NLP training is resource intense. Some BERT models are trained with 64 GB TPU using multiple nodes. Unfortunately, the computer processing speed has not caught up and it takes months to train the BERT-large model with a single GPU.

We built a model with BERT architecture using Ktrain to classify the unsampled data. BERT achieves 92% Training accuracy and 66% Validation accuracy, which is quite a bit higher than the 57% validation accuracy achieved by Random Forest DL model - HyperParameter tuned on unsampled data.

	precision	recall	f1-score	support
0	0.75	0.92	0.83	329
1	0.00	0.00	0.00	3
2	0.56	0.56	0.56	9
3	0.50	0.50	0.50	2
4	0.57	0.55	0.56	31
5	0.69	0.48	0.56	23
6	0.31	0.45	0.37	11
7	0.00	0.00	0.00	5
8	0.57	0.50	0.53	8
9	1.00	1.00	1.00	1
10	0.83	0.50	0.62	10
11	0.56	0.23	0.32	22
12	0.36	0.25	0.29	20
13	0.00	0.00	0.00	6
14	0.50	0.40	0.44	5
15	0.50	1.00	0.67	2
16	0.33	1.00	0.50	2
17	0.86	0.84	0.85	38
18	0.67	0.46	0.55	13
19	0.25	0.20	0.22	5
20	0.00	0.00	0.00	2
21	0.00	0.00	0.00	6
22	0.80	0.67	0.73	6
23	0.53	0.33	0.41	27
24	0.00	0.00	0.00	1
25	0.00	0.00	0.00	4
27	0.50	0.57	0.53	14
28	0.67	0.22	0.33	9
30	0.00	0.00	0.00	1
31	0.50	0.20	0.29	5
33	0.00	0.00	0.00	3
34	0.44	0.80	0.57	5
35	0.25	0.25	0.25	4
36	1.00	0.67	0.80	6
37	0.20	0.14	0.17	7
39	0.00	0.00	0.00	1
40	0.00	0.00	0.00	3
42	0.00	0.00	0.00	1
44	0.00	0.00	0.00	2
45	0.85	0.73	0.79	15
49	0.00	0.00	0.00	1
56	0.88	0.50	0.64	14
57	0.00	0.00	0.00	4
59	1.00	0.29	0.44	7
62	0.00	0.00	0.00	2
63	0.00	0.00	0.00	2
67	0.50	0.67	0.57	6
69	0.00	0.00	0.00	1
70	0.00	0.00	0.00	1
72	0.60	0.88	0.72	56
73	0.24	0.20	0.22	20
accuracy			0.66	781
macro avg	0.37	0.33	0.33	781
weighted avg	0.62	0.66	0.63	781

https://github.com/google-research/bert/blob/master/README.md

Machine Learning Models where accuracy > 80 %

Random Forest Classification

Bagging

XGBoost

CatBoost

Random Forest Classifier

The Random Forest Algorithm is composed of different decision trees, each with the same nodes, but using different data that leads to different leaves. It merges the decisions of multiple decision trees in order to find an answer, which represents the average of all these decision trees. Tree based models work by learning in hierarchical manner.

Random forests is a supervised learning algorithm which can be used both for classification as well as regression. It is also the most flexible and easy to use algorithm. It creates decision trees on randomly selected data samples, gets prediction from each tree and selects the best solution by means of voting. It also provides a pretty good indicator of the feature importance. We have received good accuracy using RFC. Also there is not much difference between with and without assigning the class weights which we plan to explore further.

Random Forest is considered ensemble learning, meaning it helps to create more accurate results by using multiple models to come to its conclusion. The algorithm uses the leaves, or final decisions, of each node to conclude its own. This increases the accuracy of the model since it's looking at the results of many different decision trees and finding an average. When performing Random Forests based on classification data, you should know that you are often using the Gini index, or the formula used to decide how nodes on a decision tree branch. This formula uses the class and probability to determine the Gini of each branch on a node, determining which of the branches is more likely to occur. We can also use entropy to determine how nodes branch in a decision tree. This is what we have used as a criterion.

RFC

	precision	recall	f1-score	support
0	0.7928	0.6241	0.6984	141
1	0.8881	1.0000	0.9407	119
2	0.9821	0.8271	0.8980	133
3	1.0000	1.0000	1.0000	128
4	0.9091	0.8759	0.8922	137
5	0.9365	0.9593	0.9478	123
6	0.9746	0.9746	0.9746	118
7	1.0000	1.0000	1.0000	137
8	0.9786	1.0000	0.9892	137
9	0.9932	1.0000	0.9966	145
10	0.9508	0.9431	0.9469	123
11	0.9350	0.9504	0.9426	121
12	0.9286	0.9369	0.9327	111
13	0.9916	1.0000	0.9958	118
14	1.0000	1.0000	1.0000	132
15	1.0000	1.0000	1.0000	129
16	0.9783	1.0000	0.9890	135
17	0.9348	0.9627	0.9485	134
18	1.0000	0.9917	0.9958	120
19	1.0000	1.0000	1.0000	139
20	0.9922	1.0000	0.9961	128
21	1.0000	1.0000	1.0000	131
22	0.9776	0.9776	0.9776	134
23	0.9417	0.9187	0.9300	123
24	0.9861	0.8987	0.9404	79
25	0.9481	0.9624	0.9552	133
26	1.0000	1.0000	1.0000	125
27	0.9771	0.9922	0.9846	129
28	0.9844	0.9767	0.9805	129
29	1.0000	1.0000	1.0000	133
30	1.0000	1.0000	1.0000	114
31	1.0000	1.0000	1.0000	147
32	1.0000	1.0000	1.0000	132
33	0.9748	1.0000	0.9872	116
34	0.9811	0.9541	0.9674	109
35	1.0000	1.0000	1.0000	133
36	1.0000	1.0000	1.0000	149
37	0.9925	1.0000	0.9962	132
38	1.0000	1.0000	1.0000	146
39	0.9922	0.9407	0.9658	135
40	0.9902	0.8279	0.9018	122
41	1.0000	1.0000	1.0000	133
42	0.7000	0.9573	0.8087	117
43	0.7805	1.0000	0.8767	32
44	1.0000	1.0000	1.0000	134
45	0.9417	0.6879	0.7951	141
46	0.9922	1.0000	0.9961	127
47	1.0000	1.0000	1.0000	109
48	1.0000	1.0000	1.0000	141
49	1.0000	1.0000	1.0000	114
50	1.0000	1.0000	1.0000	134
51	1.0000	1.0000	1.0000	122
52	1.0000	1.0000	1.0000	138
53	1.0000	0.5390	0.7005	141
54	1.0000	1.0000	1.0000	140
55	1.0000	1.0000	1.0000	133
56	0.9155	0.4815	0.6311	135
57	0.9732	0.8385	0.9008	130
58	1.0000	1.0000	1.0000	127
59	1.0000	1.0000	1.0000	107
60	1.0000	1.0000	1.0000	117
61	1.0000	1.0000	1.0000	131
62	1.0000	1.0000	1.0000	127
63	1.0000	1.0000	1.0000	149
64	1.0000	1.0000	1.0000	126
65	1.0000	1.0000	1.0000	126
66	1.0000	1.0000	1.0000	120
67	0.9926	1.0000	0.9963	135
68	1.0000	1.0000	1.0000	126
69	1.0000	1.0000	1.0000	128
70	0.9624	1.0000	0.9808	128
71	1.0000	1.0000	1.0000	122
72	0.9851	0.5280	0.6875	125
73	0.2869	0.8400	0.4277	125
accuracy			0.9495	9399
macro avg	0.9654	0.9509	0.9523	9399
weighted avg	0.9679	0.9495	0.9528	9399

RFC With Class Weights

	precision	recall	f1-score	support
0	0.8426	0.6454	0.7309	141
1	0.8881	1.0000	0.9407	119
2	0.9649	0.8271	0.8907	133
3	1.0000	1.0000	1.0000	128
4	0.9453	0.8832	0.9132	137
5	0.9219	0.9593	0.9402	123
6	0.9829	0.9746	0.9787	118
7	1.0000	1.0000	1.0000	137
8	0.9856	1.0000	0.9928	137
9	0.9932	1.0000	0.9966	145
10	0.9748	0.9431	0.9587	123
11	0.9504	0.9504	0.9504	121
12	0.8595	0.9369	0.8966	111
13	1.0000	1.0000	1.0000	118
14	1.0000	1.0000	1.0000	132
15	1.0000	1.0000	1.0000	129
16	0.9854	1.0000	0.9926	135
17	0.9214	0.9627	0.9416	134
18	0.9917	0.9917	0.9917	120
19	1.0000	1.0000	1.0000	139
20	0.9922	1.0000	0.9961	128
21	1.0000	1.0000	1.0000	131
22	0.9776	0.9776	0.9776	134
23	0.9576	0.9187	0.9378	123
24	0.9861	0.8987	0.9404	79
25	0.9481	0.9624	0.9552	133
26	1.0000	1.0000	1.0000	125
27	0.9697	0.9922	0.9808	129
28	0.9921	0.9767	0.9844	129
29	1.0000	1.0000	1.0000	133
30	1.0000	1.0000	1.0000	114
31	1.0000	1.0000	1.0000	147
32	1.0000	1.0000	1.0000	132
33	0.9748	1.0000	0.9872	116
34	0.9905	0.9541	0.9720	109
35	0.9925	1.0000	0.9963	133
36	1.0000	1.0000	1.0000	149
37	1.0000	1.0000	1.0000	132
38	1.0000	1.0000	1.0000	146
39	0.9922	0.9407	0.9658	135
40	0.9902	0.8279	0.9018	122
41	1.0000	1.0000	1.0000	133
42	0.7000	0.9573	0.8087	117
43	0.7805	1.0000	0.8767	32
44	1.0000	1.0000	1.0000	134
45	0.9417	0.6879	0.7951	141
46	0.9922	1.0000	0.9961	127
47	1.0000	1.0000	1.0000	109
48	0.9930	1.0000	0.9965	141
49	1.0000	1.0000	1.0000	114
50	1.0000	1.0000	1.0000	134
51	1.0000	1.0000	1.0000	122
52	1.0000	1.0000	1.0000	138
53	1.0000	0.5390	0.7005	141
54	1.0000	1.0000	1.0000	140
55	1.0000	1.0000	1.0000	133
56	0.9028	0.4815	0.6280	135
57	0.9732	0.8385	0.9008	130
58	1.0000	1.0000	1.0000	127
59	1.0000	1.0000	1.0000	107
60	1.0000	1.0000	1.0000	117
61	1.0000	1.0000	1.0000	131
62	1.0000	1.0000	1.0000	127
63	1.0000	1.0000	1.0000	149
64	1.0000	1.0000	1.0000	126
65	1.0000	1.0000	1.0000	126
66	1.0000	1.0000	1.0000	120
67	0.9926	1.0000	0.9963	135
68	1.0000	1.0000	1.0000	126
69	1.0000	1.0000	1.0000	128
70	0.9624	1.0000	0.9808	128
71	1.0000	1.0000	1.0000	122
72	0.9701	0.5200	0.6771	125
73	0.2861	0.8400	0.4268	125
accuracy			0.9498	9399
macro avg	0.9658	0.9512	0.9526	9399
weighted avg	0.9683	0.9498	0.9532	9399

Note: Precision - is the fraction of the relevant documents among the retrieved documents. Recall - is the fraction of relevant documents retrieved over the total relevant documents. F1-score - is the harmonic mean of precision and recall

Bagging

Bagging (Bootstrap aggregating) was proposed by Leo Breiman in 1994 to improve classification by combining classifications of randomly generated training sets. This is a sequential model ensembling method. First, a training dataset is split into subsets. Then models are trained on each of these subsets.

After this, predictions are combined using mean or majority voting. Bagging helps reduce variance error and avoid model overfitting. Although it is usually applied to decision tree methods, it can be used with any type of method. Bagging is a special case of the model averaging approach. As an example, the random forest algorithm could combine random decision trees with bagging to achieve relatively higher classification accuracy.

In bagging the samples are generated in such a way that the samples are different from each other however replacement is allowed. Replacement means that an instance can occur in multiple samples a multiple times or it can not appear in some samples at all. These samples are then given to multiple learners and then the results from each learner are combined in the form of voting. Because bagging and boosting each rely on collections of classifiers, they're known as 'ensemble' methods. Neither of these are a type of classifier in their own right — rather, they each produce class predictions by aggregating the predictions of several other sub-classifiers.

Bagging is about building multiple models (typically of the same type) from different subsamples of the training dataset while boosting is about building multiple models (typically of the same type) each of which learns to fix the prediction errors of a prior model in the chain.

Bagging performs best with algorithms that have high variance. A popular example are decision trees, often constructed without pruning. Random forest is an extension of bagged decision trees. Samples of the training dataset are taken with replacement, but the trees are constructed in a way that reduces the correlation between individual classifiers. Specifically, rather than greedily choosing the best split point in the construction of the tree, only a random subset of features are considered for each split. Extra Trees are another modification of bagging where random trees are constructed from samples of the training dataset. Bagging reduces overfitting (variance) by averaging or voting, however, this leads to an increase in bias, which is compensated by the reduction in variance though. Bagging and boosting are two techniques that can be used to improve the accuracy of Classification & Regression Trees (CART).

Sci-kit learn's implementation of the bagging ensemble is BaggingClassifier, which accepts as an input the designation of a base classifier which the bagging ensemble will replicate n times. BaggingClassifier's primary tuning hyper-parameter is the number of base classifiers created & aggregated in the meta-prediction.

	precision	recall	f1-score	support
0	0.8608	0.4823	0.6182	141
1	0.8815	1.0000	0.9370	119
2	0.9735	0.8271	0.8943	133
3	0.9846	1.0000	0.9922	128
4	0.9370	0.8686	0.9015	137
5	0.9516	0.9593	0.9555	123
6	0.9829	0.9746	0.9787	118
7	0.9786	1.0000	0.9892	137
8	0.9580	1.0000	0.9786	137
9	0.9667	1.0000	0.9831	145
10	0.9831	0.9431	0.9627	123
11	0.8779	0.9504	0.9127	121
12	0.9444	0.9189	0.9315	111
13	1.0000	1.0000	1.0000	118
14	1.0000	1.0000	1.0000	132
15	1.0000	1.0000	1.0000	129
16	0.9783	1.0000	0.9890	135
17	0.9692	0.9403	0.9545	134
18	0.9675	0.9917	0.9794	120
19	0.9653	1.0000	0.9823	139
20	0.9922	1.0000	0.9961	128
21	1.0000	1.0000	1.0000	131
22	0.9850	0.9776	0.9813	134
23	0.9504	0.9350	0.9426	123
24	0.9861	0.8987	0.9404	79
25	0.9481	0.9624	0.9552	133
26	0.9921	1.0000	0.9960	125
27	0.9697	0.9922	0.9808	129
28	0.9921	0.9767	0.9844	129
29	0.9852	1.0000	0.9925	133
30	0.9913	1.0000	0.9956	114
31	1.0000	1.0000	1.0000	147
32	1.0000	1.0000	1.0000	132
33	0.9748	1.0000	0.9872	116
34	0.9905	0.9541	0.9720	109
35	1.0000	1.0000	1.0000	133
36	1.0000	1.0000	1.0000	149
37	0.9925	1.0000	0.9962	132
38	1.0000	1.0000	1.0000	146
39	0.9845	0.9407	0.9621	135
40	0.9712	0.8279	0.8938	122
41	1.0000	1.0000	1.0000	133
42	0.6957	0.9573	0.8058	117
43	0.7805	1.0000	0.8767	32
44	1.0000	1.0000	1.0000	134
45	0.9423	0.6950	0.8000	141
46	0.9769	1.0000	0.9883	127
47	0.9820	1.0000	0.9909	109
48	0.9930	1.0000	0.9965	141
49	1.0000	1.0000	1.0000	114
50	1.0000	1.0000	1.0000	134
51	0.9839	1.0000	0.9919	122
52	1.0000	1.0000	1.0000	138
53	0.9870	0.5390	0.6972	141
54	1.0000	1.0000	1.0000	140
55	0.9925	1.0000	0.9963	133
56	0.9155	0.4815	0.6311	135
57	0.9561	0.8385	0.8934	130
58	1.0000	1.0000	1.0000	127
59	1.0000	1.0000	1.0000	107
60	1.0000	1.0000	1.0000	117
61	1.0000	1.0000	1.0000	131
62	1.0000	1.0000	1.0000	127
63	1.0000	1.0000	1.0000	149
64	1.0000	1.0000	1.0000	126
65	1.0000	1.0000	1.0000	126
66	0.9917	1.0000	0.9959	120
67	0.9854	1.0000	0.9926	135
68	1.0000	1.0000	1.0000	126
69	1.0000	1.0000	1.0000	128
70	0.9552	1.0000	0.9771	128
71	1.0000	1.0000	1.0000	122
72	0.9565	0.5280	0.6804	125
73	0.2853	0.8400	0.4260	125
accuracy			0.9470	9399
macro avg	0.9628	0.9487	0.9494	9399
weighted avg	0.9652	0.9470	0.9498	9399

XGBoost

XGBoost stands for "Extreme Gradient Boosting", where the term "Gradient Boosting" originates from the paper Greedy Function Approximation: A Gradient Boosting Machine, by Friedman. XGBoost algorithm was developed as a research project at the University of Washington. Tianqi Chen and Carlos Guestrin presented their paper at SIGKDD Conference in 2016 and caught the Machine Learning world by fire.

The gradient boosted trees has been around for a while, and there are a lot of materials on the topic. It explains boosted trees in a self -contained and principled way using the elements of supervised learning. Three main forms of gradient boosting are supported: Gradient Boosting algorithm also called gradient boosting machine including the learning rate. Stochastic Gradient Boosting with sub-sampling at the row, column and column per split levels. Regularized Gradient Boosting with both L1 and L2 regularization. Generally, XGBoost is fast. Really fast when compared to other implementations of gradient boosting.

XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way.

The same code runs on major distributed environment (Hadoop, SGE, MPI) and can solve problems beyond billions of examples.

XGBoost approaches the process of sequential tree building using parallelized implementation. The stopping criterion for tree splitting within GBM framework is greedy in nature and depends on the negative loss criterion at the point of split. XGBoost uses 'max_depth' parameter as specified instead of criterion first and starts pruning trees backward. This 'depth-first' approach improves computational performance significantly. It penalizes more complex models through both LASSO (L1) and Ridge (L2) regularization to prevent overfitting. XGBoost naturally admits sparse features for inputs by automatically 'learning' best missing value depending on training loss and handles different types of sparsity patterns in the data more efficiently. The algorithm comes with built-in cross-validation method at each iteration, taking away the need to explicitly program this search and to specify the exact number of boosting iterations required in a single run.

This algorithm has been designed to make efficient use of hardware resources. This is accomplished by cache awareness by allocating internal buffers in each thread to store gradient statistics. Further enhancements such as 'out-of-core' computing optimize available disk space while handling big data-frames that do not fit into memory.

0	0.5233	0.3191	0.3965	141
1	0.8156	0.9664	0.8846	119
2	0.9186	0.5940	0.7215	133
3	0.9275	1.0000	0.9624	128
4	0.8242	0.5474	0.6579	137
5	0.8165	0.7236	0.7672	123
6	0.7857	0.7458	0.7652	118
7	0.8618	0.9562	0.9066	137
8	0.8880	0.8102	0.8473	137
9	0.9603	1.0000	0.9797	145
10	0.8362	0.7886	0.8117	123
11	0.6444	0.4793	0.5498	121
12	0.7423	0.6486	0.6923	111
13	0.8837	0.9661	0.9231	118
14	0.9231	1.0000	0.9600	132
15	0.8716	1.0000	0.9314	129
16	0.9507	1.0000	0.9747	135
17	0.7226	0.7388	0.7306	134
18	0.7627	0.7500	0.7563	120
19	0.9385	0.8777	0.9071	139
20	0.8533	1.0000	0.9209	128
21	0.8675	1.0000	0.9291	131
22	0.8718	0.7612	0.8127	134
23	0.6620	0.3821	0.4845	123
24	0.8636	0.9620	0.9102	79
25	0.6723	0.8947	0.7677	133
26	1.0000	1.0000	1.0000	125
27	0.8273	0.8915	0.8582	129
28	0.9744	0.8837	0.9268	129
29	1.0000	1.0000	1.0000	133
30	0.9421	1.0000	0.9702	114
31	0.9735	1.0000	0.9866	147
32	1.0000	1.0000	1.0000	132
33	0.9134	1.0000	0.9547	116
34	0.7959	0.7156	0.7536	109
35	0.8408	0.9925	0.9103	133
36	0.9586	0.9329	0.9456	149
37	0.8980	1.0000	0.9462	132
38	1.0000	1.0000	1.0000	146
39	0.9549	0.9487	0.9478	135
40	0.9151	0.7951	0.8509	122
41	1.0000	1.0000	1.0000	133
42	0.6829	0.9573	0.7972	117
43	1.0000	0.8438	0.9153	32
44	1.0000	1.0000	1.0000	134
45	0.8700	0.6170	0.7220	141
46	0.9549	1.0000	0.9769	127
47	0.9909	1.0000	0.9954	109
48	0.9724	1.0000	0.9860	141
49	0.9661	1.0000	0.9828	114
50	1.0000	1.0000	1.0000	134
51	0.9457	1.0000	0.9721	122
52	1.0000	1.0000	1.0000	138
53	1.0000	0.5390	0.7005	141
54	1.0000	1.0000	1.0000	140
55	1.0000	1.0000	1.0000	133
56	0.7800	0.2889	0.4216	135
57	0.9732	0.8385	0.9008	130
58	1.0000	1.0000	1.0000	127
59	0.9386	1.0000	0.9683	107
60	1.0000	1.0000	1.0000	117
61	1.0000	1.0000	1.0000	131
62	1.0000	1.0000	1.0000	127
63	1.0000	1.0000	1.0000	149
64	1.0000	1.0000	1.0000	126
65	1.0000	1.0000	1.0000	126
66	1.0000	1.0000	1.0000	120
67	0.7764	0.9259	0.8446	135
68	1.0000	1.0000	1.0000	126
69	1.0000	1.0000	1.0000	128
70	0.9624	1.0000	0.9808	128
71	1.0000	1.0000	1.0000	122
72	0.7326	0.5840	0.5972	125
73	0.2542	0.7280	0.3768	125
accuracy			0.8822	9399
macro avg	0.8944	0.8825	0.8803	9399
weighted avg	0.8950	0.8822	0.8803	9399

CatBoost

CatBoost is a machine learning algorithm that uses gradient boosting on decision trees. "CatBoost" name comes from two words "Category" and "Boosting". "Boost" comes from gradient boosting machine learning algorithm as this library is based on gradient boosting library. Gradient boosting is a powerful machine learning algorithm that is widely applied to multiple types of business challenges like fraud detection, recommendation items, forecasting and it performs well also.

CatBoost is a recently open-sourced machine learning algorithm from Yandex. It can easily integrate with deep learning frameworks like Google's TensorFlow and Apple's Core ML. It can work with diverse data types to help solve a wide range of problems that businesses face today. The library works well with multiple Categories of data, such as audio, text, image including historical data. To top it up, it provides best-in-class accuracy. It is especially powerful in two ways: It yields state-of-the-art results without extensive data training typically required by other machine learning methods and Provides powerful out-of-the-box support for the more descriptive data formats that accompany many business problems.

CatBoost has both CPU and GPU implementations. The GPU implementation allows for much faster training and is faster than both state-of-the-art open-source GBDT GPU implementations, XGBoost and LightGBM, on ensembles of similar sizes.

The library also has a fast CPU scoring implementation, which outperforms XGBoost and LightGBM implementations on ensembles of similar sizes. In addition to feature importance, which is quite popular for GBDT models to share, Catboost provides feature interactions and object (row) importance CatBoost has two modes for choosing the tree structure, Ordered and Plain.

	precision	recall	f1-score	support
0	0.0000	0.0000	0.0000	141
1	0.0000	0.0000	0.0000	119
2	0.9444	0.2556	0.4024	133
3	0.0000	0.0000	0.0000	128
4	0.0000	0.0000	0.0000	137
5	0.0000	0.0000	0.0000	123
6	0.0000	0.0000	0.0000	118
7	0.0000	0.0000	0.0000	137
8	0.0000	0.0000	0.0000	137
9	0.0000	0.0000	0.0000	145
10	0.0000	0.0000	0.0000	123
11	0.0000	0.0000	0.0000	121
12	0.0000	0.0000	0.0000	111
13	0.0000	0.0000	0.0000	118
14	0.0000	0.0000	0.0000	132
15	0.0000	0.0000	0.0000	129
16	0.0000	0.0000	0.0000	135
17	0.0000	0.0000	0.0000	134
18	0.0000	0.0000	0.0000	120
19	0.0000	0.0000	0.0000	139
20	0.0000	0.0000	0.0000	128
21	0.0000	0.0000	0.0000	131
22	0.0000	0.0000	0.0000	134
23	0.0000	0.0000	0.0000	123
24	0.0000	0.0000	0.0000	79
25	0.0000	0.0000	0.0000	133
26	1.0000	1.0000	1.0000	125
27	0.0000	0.0000	0.0000	129
28	0.0000	0.0000	0.0000	129
29	1.0000	1.0000	1.0000	133
30	0.0000	0.0000	0.0000	114
31	0.0000	0.0000	0.0000	147
32	1.0000	1.0000	1.0000	132
33	1.0000	0.1810	0.3066	116
34	0.0000	0.0000	0.0000	109
35	0.0000	0.0000	0.0000	133
36	0.0000	0.0000	0.0000	149
37	0.0000	0.0000	0.0000	132
38	1.0000	0.3767	0.5473	146
39	0.0000	0.0000	0.0000	135
40	0.0000	0.0000	0.0000	122
41	1.0000	0.1353	0.2384	133
42	0.3239	0.1966	0.2447	117
43	0.0000	0.0000	0.0000	32
44	0.0000	0.0000	0.0000	134
45	0.0000	0.0000	0.0000	141
46	0.0000	0.0000	0.0000	127
47	1.0000	0.1651	0.2835	109
48	0.0000	0.0000	0.0000	141
49	0.0000	0.0000	0.0000	114
50	1.0000	1.0000	1.0000	134
51	0.0000	0.0000	0.0000	122
52	1.0000	1.0000	1.0000	138
53	1.0000	0.5390	0.7005	141
54	1.0000	1.0000	1.0000	140
55	0.0000	0.0000	0.0000	133
56	0.0000	0.0000	0.0000	135
57	0.9545	0.1615	0.2763	130
58	1.0000	1.0000	1.0000	127
59	0.0173	1.0000	0.0340	107
60	1.0000	1.0000	1.0000	117
61	1.0000	1.0000	1.0000	131
62	0.0000	0.0000	0.0000	127
63	1.0000	1.0000	1.0000	149
64	1.0000	1.0000	1.0000	126
65	1.0000	1.0000	1.0000	126
66	1.0000	1.0000	1.0000	120
67	0.0000	0.0000	0.0000	135
68	1.0000	1.0000	1.0000	126
69	1.0000	1.0000	1.0000	128
70	0.9697	1.0000	0.9846	128
71	1.0000	1.0000	1.0000	122
72	0.0000	0.0000	0.0000	125
73	0.1735	0.4480	0.2489	125
accuracy			0.2798	9399
macro avg	0.3295	0.2764	0.2739	9399
weighted avg	0.3356	0.2798	0.2798	9399

Comparative Model Performance

Model	Test accuracy
Random Forest	0.949463
Random Forest Classifier - Weighted	0.949782
Adaboost Classifier	0.082349
Bagging Classifier	0.947016
GradientBoost Classifier	0.874455
XGBoost Classifier	0.882222
CatBoost Classifier	0.931908
SVM Classifier	0.279817
SGD Classifier	0.274391
MultinomialNB Classifier	0.299394
ComplementNB Classifier	0.107245
KNN Classifier	0.810725
Logistic Regression	0.417917

Model	Test accuracy
Bidirectional LSTM – With Word2Vec	0.951652
Bidirectional LSTM – Glove Embedding	0.939171
GRU Glove	0.939171
RNN Glove	0.945884

Model	Test accuracy
BERT (unsampled data)	0.66

Hyper parameter Tuning - Background

We have resampled the records. Total number of records were 46991.Random forest performed well as compared to other Machine Learning models. Hence, Performance tuning is performed for Random Forest Model.

Google collab and on local machine, performance tuning was taking all the resources for sampled (46991) records. Hence, it was decided to do performance tuning on unsampled data. Since dataset has highly imbalanced class frequencies, it was suggested to group the classes having less than 10 incidents into one class. This is to recommend that the number of groups can be reduced and clubbed into one group.

The Random forest classifier is implemented on unsampled data (with 50 classes, less than ten incident classes clubbed into one group). KFold cross validation is implemented. Stratified Kfold validation is also performed since class was imbalanced.

Random Forest Classifier (Without resampling)

Number of Samples: 7839

Number of Labels: 7839

Number of Training Samples: 6271

Number of Validation Samples: 1568

Time Taken To Train RFC on dataset 3 : 3.9492239952087402

Training Accuracy: 0.9467389571041301

Test Accuracy: 0.5478316326530612

Since testing accuracy is very less than training accuracy. The model is overfitting. Regularization is applied in the processing scripts.

Model	Train accuracy	Test accuracy
Random Forest	0.946739	0.547832

Random Forest Classifier (KFold Cross Validation)

KFold cross validation is done since target classes are imbalanced.

Number of Samples: 7839
Number of Labels: 7839
Time Taken To Train RFC on dataset 3 : 35.8132209777832
KFold Average accuracy: 0.5475146284567467

Average KFold Average Accuracy is 54 %. Let us implement regularization to find best parameters and improve accuracy.

Random Forest Classifier (StratifiedKFold)

Number of Samples: 7839
Number of Labels: 7839
1 of KFold 10
Accuracy: 0.5497448979591837
2 of KFold 10
Accuracy: 0.5459183673469388
3 of KFold 10
Accuracy: 0.5650510204081632
4 of KFold 10
Accuracy: 0.548469387755102
5 of KFold 10
Accuracy: 0.5510204081632653
6 of KFold 10
Accuracy: 0.5510204081632653
7 of KFold 10
Accuracy: 0.5625
8 of KFold 10
Accuracy: 0.5522959183673469
9 of KFold 10
Accuracy: 0.5586734693877551
10 of KFold 10
Accuracy: 0.5721583652618135
StratifiedKFold Average accuracy 0.5556852242812834

Average Accuracy using Stratified Kfold: 55.56 % on unsampled data with total 50 classes (classes with less than 10 incidents ,clubbed into one group)

Random Forest Classifier (HyperParameter tuned via GridSearch)

Parameters to Grid Search:

n_estimators = [100, 300, 500, 800, 1200]

max_depth = [5, 8, 15, 25, 30]

min_samples_split = [2, 5, 10, 15, 100]

min_samples_leaf = [1, 2, 5, 10]

Result of Grid Search:

Number of Samples: 7839

Number of Labels: 7839

Number of Training Samples: 6271

Number of Validation Samples: 1568

Best Parameters obtained:

{'max_depth': 30, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 800}

Random Forest results with best parameters from Grid Search

Best parameters from Grid Search:

'max_depth': 30
'min_samples_leaf': 1
'min_samples_split': 2
'n_estimators': 800

Model Accuracies:

Training Accuracy: 0.9437358276643991
Test Accuracy: 0.5708812260536399

Analysis:

Model is overfitting. The training accuracy if very high and testing accuracy is very low. It can be seen that even with hyperParameter tuned RandomForest Classifier using GridSeach and Stratified KFold on the original dataset(with resampling) is giving a maximum accuracy of around 57%. Hence we should go with up-sampling of the data where we are seeing the accuracy around 95%

DecisionTree (considering cost_complexity_pruning_path) [OPTIONAL]

Since Random forest was overfitting, it was suggested to consider cost complexity pruning path as suggested.

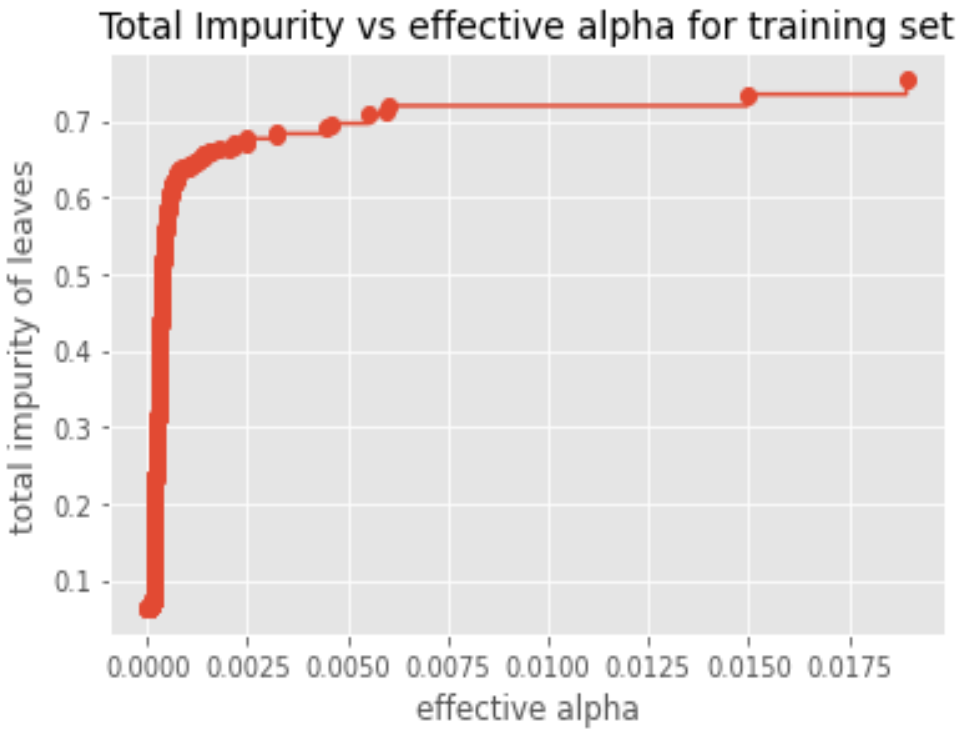
Decision Tree Algorithm results:
Number of Samples: 7839
Number of Labels: 7839
Number of Training Samples: 6271
Number of Validation Samples: 1568
Training Accuracy: 0.9467389571041301
Test Accuracy: 0.41581632653061223

Since Decision tree is overfitting, we will try to consider cost_complexity_pruning_path

Total Impurity vs effective alpha for training set

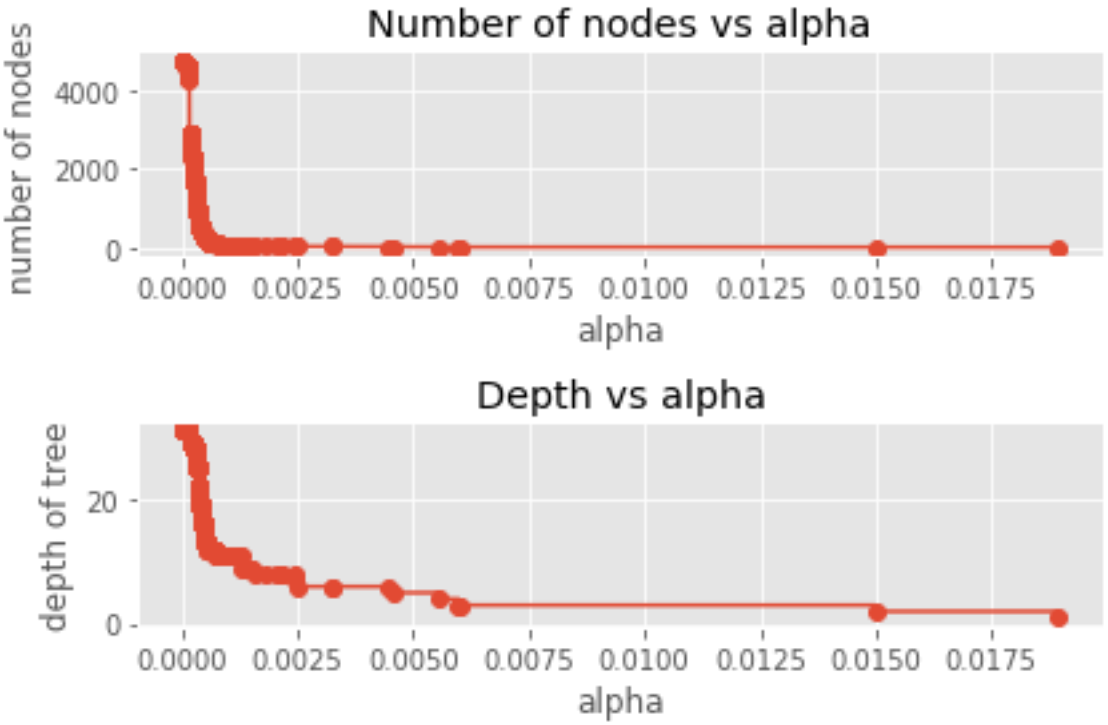
```
from sklearn.tree import DecisionTreeClassifier
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=10)

start = time.time()
clf = DecisionTreeClassifier(random_state=101)
path = clf.cost_complexity_pruning_path(X_train,
y_train)
ccp_alphas, impurities = path.ccp_alphas,
path.impurities
end = time.time()
```



Computing ccp_alpha

Number of nodes in the last tree is: 1 with ccp_alpha: 0.0399012813600651. Using this computed ccp_alpha and training Decision Tree model once again.



Decision Tree with ccp_alpha

Decision Tree model with ccp_alpha details:

Number of Samples: 7839
Number of Labels: 7839
Number of Training Samples: 6271
Number of Validation Samples: 1568
Time Taken To Train Decision Tree on dataset : 1.3434321880340576
Training Accuracy: 0.4785520650613937
Test Accuracy: 0.4789540816326531

The training and testing accuracy both reduced but model is not overfitting with ccp_alpha.
We can see that even with trying different techniques, the accuracy is not improving. Hence we should go with up-sampling of the data where we are seeing the accuracy around 95%

Hyper-Parameter tuned Machine Learning Model Performance

Model	Test accuracy
Random Forest	0.547194
Random Forest[KFold]	0.548917
Random Forest[Stratified KFold]	0.554409
Random Forest[HP]	0.570881
Decision Tree[HP]	0.478954
Decision Tree	0.415816
Decision Tree[HP]	0.478954

Hyper Tuning of Deep Learning Models

We believe the DL models could be regularized and experimented with L1,L2 parameters to be further tuned for better accuracies. Both these aspects are meant to be exercised with more hardware at disposal. Currently it look lot of time to run regularization on the Deep Learning Models suggested. Hence we believe that the below options can be explored further. Following can be considered while tuning deep learning models:

Dropout*: Slow down learning with regularization methods like dropout on the recurrent LSTM connections.*

Layers: *Explore additional hierarchical learning capacity by adding more layers and varied numbers of neurons in each layer.*

Regularization: *Explore how weight regularization, such as L1 and L2, can be used to slow down learning and overfitting of the network on some configurations.*

Optimization Algorithm: *Explore the use of alternate optimization algorithms, such as classical gradient descent, to see if specific configurations to speed up or slow down learning can lead to benefits.*

Loss Function: *Explore the use of alternative loss functions to see if these can be used to lift performance.*

Features and Timestep:*. Explore the use of lag observations as input features and input time steps of the feature to see if their presence as input can improve learning and/or predictive capability of the model.*

Larger Batch Size: *Explore larger batch sizes than 4, perhaps requiring further manipulation of the size of the training and test datasets.*

Epochs, number of neurons too can be explored further.

We have used BERT but it is computation intensive to train. We believe this model can be trained on text for even better prediction as observed in this project. And can also be fine tuned in the future. Transfer Learning can be done from scratch based on the current dataset.

Comparative Benchmarking

Once training is complete, it's time to see if the model is any good, using comparative benchmarking or evaluation as against the current baseline performance (automated or manual). This is where that the quality of the dataset & hyper tuned models that we set aside, for testing, comes into play. Evaluation allows us to test our model against data that has never been used for training. This metric allows us to see how the model might perform against data that it has not yet seen. This is meant to be representative of how the model might perform in the real world. By using different metrics for performance evaluation, we should be able to improve the overall predictive power of our model(s) before we roll it out for production on unseen data. Without doing a proper evaluation of the ML model using different metrics, and depending only on accuracy, can lead to a problem when the respective model is deployed on unseen data and can result in poor predictions. This happens because, in cases where the model does not learn but instead memorizes; hence, it is not able to generalize well on unseen data. Comparative performances results of different classifiers are presented in this section. Evaluation metrics are tied to learning tasks.

There are different metrics for the tasks of classification, regression, ranking, clustering, topic modeling, etc. Some key metrics are as follows: Model accuracy in terms of classification models can be defined as the ratio of correctly classified incidents to the total number of incidents in the respective assignment groups. Precision and Recall In a classification task, the precision for a class is the number of true positives (i.e. the % or number of items correctly labeled as belonging to the particular positive class or assignment group) divided by the total number of elements or incidents labeled/predicted as belonging to the particular or positive class (i.e. the sum of true positives and false positives i.e. total number of predictions made, which are items incorrectly labeled as belonging to the class or assignment group). In this context, recall is defined as the number of true positives (correct assignments to the group) divided by the total number of elements or incidents that actually belong to the positive or particular class (i.e. the sum of true positives (correct assignments) and false negatives (incorrect assignments i.e. tickets in that group labeled or assigned to other groups), which are items which were not labeled as belonging to the positive class or assignment group but should have been.

High recall means that an algorithm returned most of the relevant results. To fully evaluate the effectiveness of a model, it's necessary to examine both precision and recall collectively (F1 Score) and as we know, precision and recall are often in conflict i.e. improving precision typically reduces recall and vice versa. The F1 score is the harmonic mean of the precision and recall, where an F1 score reaches its best value at 1 (perfect precision and recall) and worst at 0. Since the harmonic mean of a list of numbers skews strongly toward the least elements of the list, it tends (compared to the arithmetic mean) to mitigate the impact of large outliers and aggravate the impact of small ones:

Proposed Model to be Productionized and Recommendations

Best Models explored so far based on criteria: Accuracy, F1 score, Precision and Recall , balanced model (Not overfitting):

Machine Learning:

Random Forest Classifier and Decision Tree based Model (Not overfitting) (94% Accuracy)

Deep Learning (All the DL models giving similar accuracies, picking up LSTM model as is was having better accuracy, embedding wise both word2Vec and Glove Embedding gave similar results)

Bidirectional LSTM – With Word2Vec (95 % Accuracy)

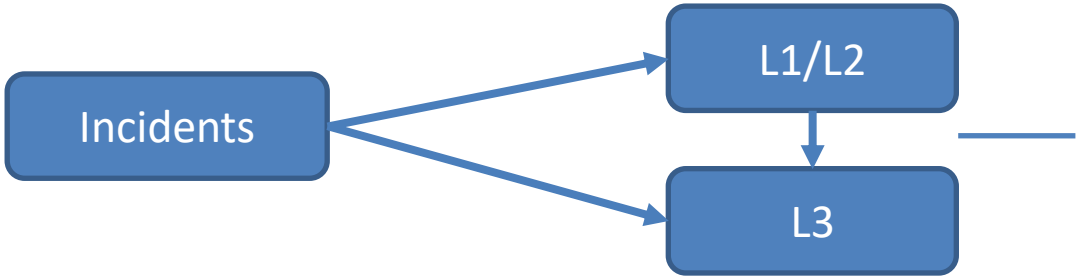
Bidirectional LSTM – Glove Embedding (93 % Accuracy)

Future Recommendation: Suggesting an Ensemble Model: *There has been a mixed response from DL and ML models. It could be hence recommended to choose the best predictors and using the ensemble pipeline which will be combination of the above mentioned ML and Deep Learning Models. Approach could be to train the ensemble model and compare the performance.*

We have explored BERT in this capstone project. It is giving 66% accuracy. In the future, BERT can also be explored further.

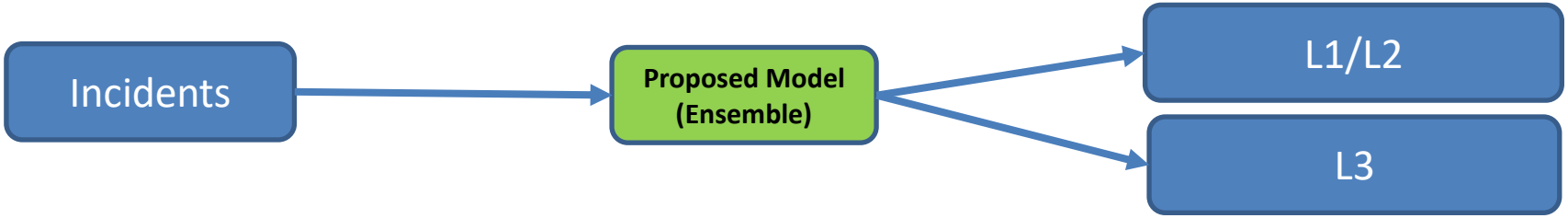
Efficiency Analysis – AI vs Manual Ticketing Categorization

Current Process



- Need 15 minutes review time for 25% to 30% before assigned to L3 teams
- Around 25% of tickets assigned incorrect.

Proposed Process



Key Considerations	Manual Allocation	AI Allocation (DL Model)
% Misclassification	~30%	~6%
SOP Review time (in minutes)	15 Minute	15 Minute
Time incur per 100 Tickets	(100*.30*15/60)= 7.5 Hrs	(100*.06*15/60)= 1.5 Hrs
% Misclassification 2nd Stage	~ 7.5% (25% of 30%)	None

Implications

Although Deep Learning models explored except BERT can classify the IT tickets with 90% + accuracy, however to achieve better accuracy in the real world it would be good if the business can collect additional data for each group (to address class imbalance).

The proposed system basically should contain two step classification process to assign issue ticket to related support unit.

1. *First step is to detect the related category of ticket which is directly related to the department or application (or type) or subject (or business or activity) of the issue (done through topic modeling if not tagged explicitly)*
2. *Second step is to determine the related subcategory or unit to be assigned under the specified category based on the type of the problem in the determined department.*

Ideally, we should also extract time stamps for each of the tickets and use the time latent information. In addition to that caller and its location or usage of system would also help improve the classification or assignment to operate from process execution perspective. Also in case the tickets post assignment by the system are rerouted to some other group i.e. different from the one being predicted. It should be picked up the classification model in future for the training purpose.

The assignment group can not be assumed as a static label. These groups could vary from time to time and hence training the model must be a continuous process. At the same time as and when the systems are removed from the production, corresponding tickets should not be used for classification purpose.

***Use of Clustering Method -** We need to also use clustering methods apart from the classification method to produce an optimal clustering models and number of assignment groups. In addition to that some of the tickets being resolved, could be eliminated by way of certain changes in the underlying systems or heling an automated response. These improvements along with the complexity of the tickets or the criticality of the user or system, the tickets could not only be queued into assignment groups but could also be prioritized for resolution considering the scale and volume of tickets (demand) and the limited number of staff (supply side or capacity). The prioritization would help in ensuring that the process overall helps in meeting the SLA most of the time.*

Implications Cont..

Tickets with inadequate information or cause explained – *We recommend use of NLG based methods to be used at the source to augment the quality of the tickets and at the same time, prompting with auto generation of tickets based on the past incidents which could repeat.*

In many cases, the tickets getting delayed or out of bound of SLA should also be trigged automatic response and the expected delay owing to the workload. It is important to keep the user informed as to where they are and how far they are from resolution. In addition to that there are tickets with inadequate data or information. Even if they are assigned correctly, they could not be resolved. In such cases the tickets are sent back for further information. We recommend use of NLG based methods to be used at the source to augment the quality of the tickets and at the same time, prompting with auto generation of tickets based on the past incidents which could repeat.

Adjustments to deal during surge in tickets *- during crisis the number of tickets could suddenly rise and hence its not just about assignment of tickets but also monitoring the queue length and velocity at any given time. In case of these crisis, the tickets generated would be mostly of the same time and hence would get assigned to the same group in the system. But, in crisis, we would like these tickets to gain priority and even though they are classified under same label but ideally should be allocated to multiple classes.*

In current system, it will not fail as it will again require manual intervention to re-route the tickets (with same label and assigned to a single group) to differently groups or classes. Hence, we would ideally recommend hierarchical labeling or classification problem resolution approach bringing dynamic adjustments in order to respond to the crisis or by default and by design to the process of using single label and multiple class classification.

Limitations

- *The manual assignment of issue tickets to appropriate unit or person in support team is not feasible for large organizations. It is time consuming and there may be mistakes due to human errors. The dataset quality and number of classes if large, could prevent or limit the models from being reasonably accurate. Hence the number of classes (given the class imbalance scenario) and sparse data, it might be recommended to use staggered approach to classification using other relevant features like application name, location, user role etc.*
- *We also faced computation issues as these models for to be trained need memory and computing power. Graphical Processing Units are fundamental to AI because they provide substantial compute power that is required for continuous and iterative processing. Training neural networks is a data-rich activity that requires big data plus the compute power to process it and the volume of activity involved. Access to such a hardware is an expensive affair. ML models, however, were comparatively faster to train as compared to the deep learning-based models.*
- *There has been a mixed response from NL and ML models. Some ML models have performed better predictions than others including DLs. It could be hence recommended to choose the best predictors and using the ensemble pipeline and train the ensemble model and compare the performance.*
- *In cases of crisis, the number of tickets could suddenly rise and hence the it is not just about assignment of tickets but also about monitoring the queue length and velocity at any given time (based on the priority). In moments of crisis, the tickets generated would mostly be of same time and hence would get assigned to the same group. I practice, such tickets would gain a priority and could be distributed to multiple assignment groups. Hence, we would ideally recommend hierarchical labeling and bring dynamic adjustment for responding to the assignment in a crisis (a case of a multiple labels being assigned to the same assignment groups)*

Closing Reflections

- *We found that the data was present in multiple languages and with texts in various formats such as emails, URLs etc. This makes data pre-processing code intensive which could be avoided through auto feed of incident text or providing tags that are relevant (like topics) to the type of problems and hence making it easier and accurate to assign.*
- *In current project, we have used both Gensim and NLTK. However it is recommended to explore other toolkits like Stanford toolkit etc.*
- *We also believe that the feature data set could not get limited to just text. The incidents could get reports as an audio or video log. Many a times these incidents might get reported with come kind of attachment or image or screen shot etc. In such cases the predictions could involve data that is a combination of text, speech, video and image. In such a case, how the methods that we have learnt and applied during the course, need to be further reflected and extended upon.*
- *We have used BERT but it is computation intensive to train. We believe this model can be trained on text for even better prediction as observed in this project. We also believe the DL models could be regularized and experimented with L1,L2 parameters for to be further tuned for better accuracies. Both these aspects are meant to be exercised with more hardware at disposal.*
- *We would need almost all the components of this project to be broken down into modular components and preferably designed to be offered as a microservices on the cloud for deployment and usage.*
- *We could also explore an advanced level of automation wherein methods like chatbot (text or voice based) could be used to interact with the user both for ticket creation and post submission of tickets for gaining any additional information on the ticket in an automated manner. Chatbots help businesses in saving time and money. Since most questions asked by customers are frequently repeated, they can be handled by chatbots. This helps customer service agents prioritize relevant customer queries, thereby ensuring overall customer satisfaction*
- *In many cases the text has been very limited (2-3 words) for any meaningful assignment. Hence, there is a need to set a minimum threshold limit for entering the incident.*

thank you!

RADHIKA KULKARNI[Radhikakulkarni844@gmail.com]
SYED TALIBUDDIN SAIFI [letters.syed@gmail.com]
TANVEER QURAISHI[tanveer.quraishi@gmail.com]
SHANMUGAM SAMPATH [shanmugam.Sampath@gmail.com]
KANWAL RAI [raikanwalrai@gmail.com]