Capstone Project
Final Project Report

Automatic Ticket Assignment using NLP

Presented to the University of Texas at Austin and Great lakes institute for PGP in Artificial intelligence and Machine Learning

Team members:

Swapnil Paithankar
Muralitharan R
Abhijeet Banerjee
Mayur Mhaske
Akash Chanda

4 July, 2020

Abstract

In this project we have implemented a deep learning architecture to classify tickets into appropriate support groups. The dataset consists of tickets with the support group labeled in them. The implementation developed uses the LSTM network with transfer learning. At the same time, we have implemented combination of text regularization techniques in order to maximize the performance of our implementation for classification task. The results obtained in the classification section reflect a significant bias for L1 groups as these groups are usually the first respondents to customer issues.

Contents

1 Introduction

1.1 Context

Due to the rapid shift of companies towards superb customer experience and satisfaction, ticketing systems have come into a prominence and represent a strategic element in business competitiveness. Different software companies have developed very effective software tools for issue tracking, nevertheless, some sub-processes and tasks within the ticketing systems are still performed manually. These manually performed tasks represent bottlenecks, especially at large organizations they result in declined productivity and increased response time. Advancements in machine learning can be used in another way in which they are combined with the traditional issue tracking and ticketing systems on the market, to enable optimal operational efficiency in the Customer Service and Support(CSS) Department of large-scale businesses that deal with customer reports.

Since the enterprise's inception, technology has been paving the way for business innovation. Cognitive computing, along with constituent technologies, seems to be yet another promising catalyst for enterprise transformation. These technologies bring along the potential for high-level automation. They are poised to improve productivity across multiple functions.

The evolution of cognitive automation is due to an emergence of various interdependent streams of Artificial Intelligence (AI). Natural Language Processing (NLP) is one of the key components of cognitive automation that can be applied to a variety of industry segments to address their pain points. It may be used to uncover relevant insight from a chain of email communication. Or, it may be employed to enhance customer experience by allowing users to post queries in their own language about products, services, or applications and receive immediate and accurate answers.

Based on AI algorithms and driven by an increased need to manage unstructured enterprise information, NLP is influencing a rapid acceptance of more intelligent solutions in various business domains.

NLP—A Key Pillar of Cognitive Automation

Cognitive systems attempt to simulate functioning of human brain, imitating the way we learn from past experiences and use that knowledge for reasoning, making hypotheses, inferring, solving problems, or making decisions. Combined with automation, enterprises can leverage these systems to automate judgment-based activities involved in a business process. In this way, they can augment human expertise so that we can maximize the use of our time.

Natural language processing is one of the main abilities of a cognitive computing system. Its goal is to process the language we use and transform the text into structured data. It enables people and machines to interact more naturally with one another and can even expand human expertise. Over time, the NLP-based systems may learn how to provide expert assistance to scientists, engineers, lawyers, and other professionals to complete a task in a fraction of a time that a human might require.

This goal is not easy to reach. Understanding language means, among other things, identifying what a word or a phrase stands for and knowing how to properly link those concepts together in a logical way. This involves a lot of complex processing. NLP research started in 1930 and we have since made significant gains in the field, though a huge communication gap between humans and computers has yet to be bridged. In recent times, thanks to various technology and business drivers, there seems to be a surge in the interest and demand for NLP. With this revived interest, many organizations are investing in the development of algorithms to process textual data and make their information accessible to computer applications.

Solving Business Challenges with NLP

Analyzing textual data is a complex and resource-intensive task. A huge amount of information is hidden within textual data repositories under the guise of emails, documents, chats or may exist in the textual data of a business. Going the extra mile can provide organizations with valuable insight for developing business strategies. If used correctly, these technologies can simplify and accelerate many business processes and pave the way for business innovations and increased agility.

Natural language processing can be applied in a variety of industry segments—media, publishing, advertising, healthcare, banking, and insurance—to improve key activities. It can be used to monitor social media, which helps a company know what their clients are discussing on social media and could reveal relevant information for their business. Another application could be a question-answering tool that could reduce effort of many of administrative jobs.

Business Domain Value

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. Incase L1 / L2 is unable to resolve, they will then escalate / assign the tickets to Functional teams from Applications and Infrastructure (L3 teams). Some portions of incidents are directly assigned to L3 teams by 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.

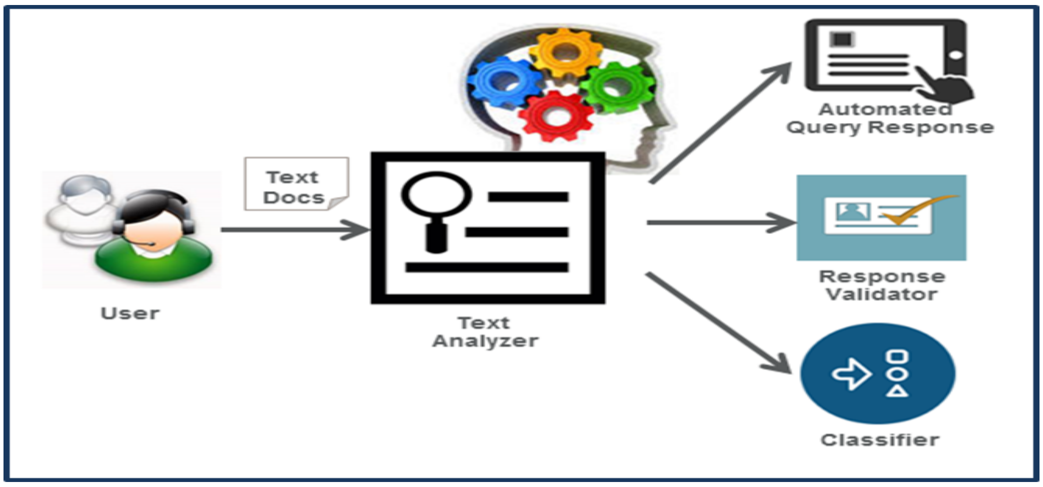


Figure 1.1: Typical ticket classification workflow.

1.2 Objectives

The overall objective of the project is, to build a classifier that predicts the support group to which an incoming incident should be assigned to. The following lists some high level objectives

- **Learn how to use different classification models** – Different classification models ranging from traditional ML models along with Deep learning models can be validated for this classification task.

- **Use transfer learning to use pre-built models –** Transfer learning is a research problem in machine learning (ML) that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem. This helps to create powerful ML models with less time spent in training.

- **Learn to set the different hyper parameters** – Hyper parameter tuning is a critical step in building any machine learning model. Choice of correct values of these parameters can help in achieving god accuracy and precision benchmarks. Parameters like optimizers, loss functions, epochs, learning rate, batch size, check pointing, early stopping etc can be tuned to achieve desired outcomes.

- Read different **research papers** of given domain to obtain the knowledge of advanced models for the given problem – A lot of research has already being done in this area. We can leverage the work done on similar problems to build a solid understanding of the challenges in this domain. We have listed the research papers and online blogs that we have read while working on this project.

.

1.3 Project Report Outline

The present document has been structured in 8 chapters.

In this first chapter, we have introduced the project offering a vision of the context and outlining the objectives and contributions of the project.

In the second chapter, we will provide the reader with a background on the use of deep networks and LSTM, and later, we introduce basic concepts on Deep Learning.

Chapter 3 provides background information to enable a better understanding of the other chapters.

Our implementation is structured in 2 modules: preprocessing module and classification module. Chapter 4 and 5 cover these modules.

Chapter 6 will unify all the modules offering a complete view of the whole implementation and we will evaluate the results obtained with respect to the context.

Finally, in chapter 7 we will dedicate it to conclusions drawn from the project and the introduction of possible future lines.

2 Background

2.1 Introduction to Deep Learning

In this section we will briefly review the fundamentals of Deep Learning.
We will begin by relating the principles on which it is based. Then we will offer a more practical view introducing the most relevant aspects of the training process and optimization of neural networks.
Finally, we will focus on convolutional neural networks and provide an overview of the state of the art in the use of these networks in classification tasks.

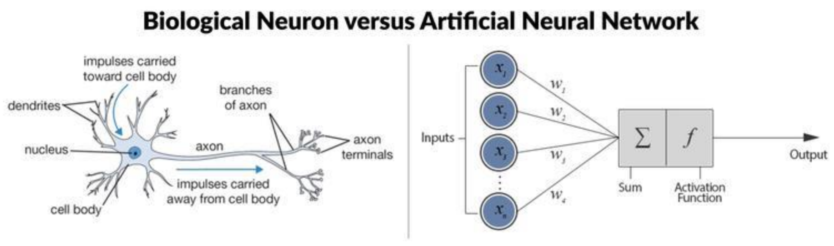


Figure 2.1: Biological Neuron versus Artificial Neural Network.

2.1.1 Deep Learning: Machine learning inspired by the human brain

Deep Learning is a branch of Machine Learning. The main and common feature is their ability to automatically learn representative features and functions from input information, which is called "representation learning" or "feature learning".
Deep Learning carries out the training and with it the learning through multiple nodes or neurons distributed in multiple layers, simulating the functioning of the human nervous system, this similarity is shown in Figure 2.1. In the case of artificial neural networks (ANN) we start from a basic processing unit called node or neuron. Each neuron is itself a simple classifier model which generates an output as a function of

the evidences of the previous layers. Each neuron performs three actions: Linear operation. A propagation function adds the signals coming from the previous layer with their respective weights and a bias value.

Non-linear operation. On the propagation function applies an activation function which determines whether or not the output of the neuron is activated. There are several activation functions and depending on the space occupied by the node within the neural network it is convenient to use one or the other. The most common activation functions are: Sigmoid, Tanh, ReLU and Softmax.

Output. The result of the activation function is located at the exit of the node and will propagate to all nodes located in the next layer.

These nodes or neurons are organized in layers and through the interaction with other neurons distributed in different layers allows to obtain more complex and hierarchical representations. Each layer essentially acts on the characteristic constructions of the layers prior to it generating a higher level feature construction.

Thus, in a neural network we encounter "N" nodes by layer and "C" different layers. In the simplest structure the nodes of the "C-1" layer are all connected to all the neurons of the "C" layer, this structure is called "feedforward" or "Multilayer Perceptrons" (MLP) and is the basic structure of Deep Learning. Figure 2.2 shows the feedforward structure

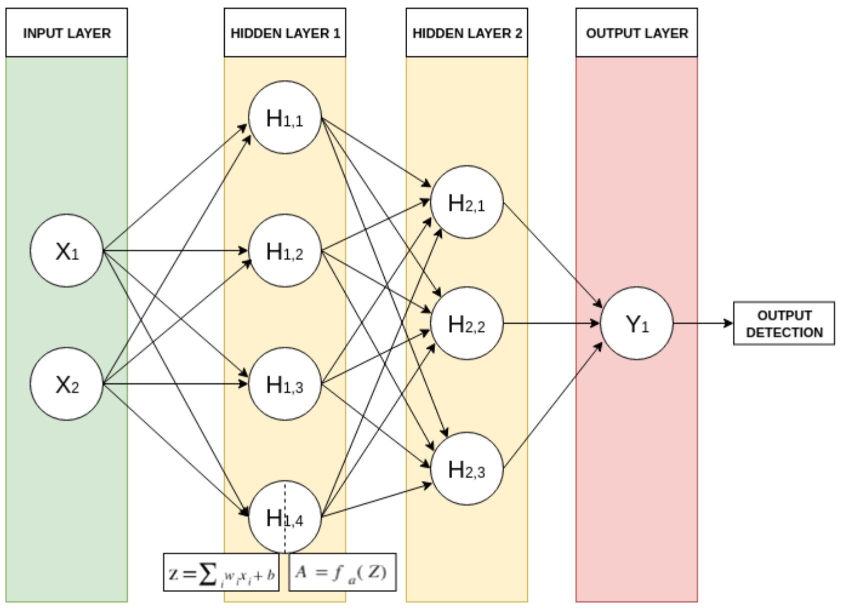


Figure 2.2: Feedforward structure with two hidden layers.

where the three main parts of a neural network are differentiated: Input Layer, Hidden Layers and Output Layer.

The number of input and output layers are fixed and their shape depends on the input dataset and the formulation of the problem or target function searched for respectively. But the size of hidden layers can be variable. If the model uses a single hidden layer it will be called "shallow neural network" and introducing more layers we will have a deeper model.

2.1.2 Types of Deep Learning algorithm

According to their purpose and the type of data used in the training process we find different types of algorithms, the main types are:

Supervised algorithm. They work with labeled data and the model is trained to minimize the cost function by comparing the prediction obtained by the model with the real value. Within this type we find: the feedforward neural network (FFNN) that we have commented previously, the Convolutional Neural Networks (CNN) used mainly in computer vision and Long Short Term Memory (LSTM) for applications based on time series data.

Semi-supervised algorithm. Only a small part of the samples have the annotations necessary to train the algorithm. This algorithm builds a self-learning scheme where the algorithm itself generates its own annotations. Within this type we find Deep reinforcement learning (DRL) and Generative Adversarial Networks (GAN).

Unsupervised algorithm. The model discovers the structure or relationships of the input information without which it has annotations or labels. Auto Encoders (AE) and Restricted Boltzmann Machines (RBM) are two models that are located within this type and perform clustering or dimensionality reduction functions.

2.1.3 How are supervised Deep Learning algorithms trained?

As we have already indicated, in Deep Learning it is not necessary to define a rule of prediction or selection of characteristics, but the model learns them by itself.

The optimization algorithm used is called gradient descent and will seek to optimize the weights of the nodes until finding a local minimum of a function, i.e., minimize the cost function.

The descent gradient method consists of 3 steps:

Forward propagation. Based on the current weights of each node a prediction is obtained.

Evaluation. The obtained prediction is compared with the objective value, obtaining an estimated value from the error of the current weights configuration in the nodes.

Backward propagation. This error is propagated in the opposite direction from the output to the input by applying slight changes in the weights proportional to the error and in the direction that minimizes the error.
These 3 steps will be repeated constantly until the end of the training process.

2.1.4 Improving Deep Neural Network: Cases of underfitting or over-fitting

Underfitting. The model has not been sufficiently trained or its structure is too simple to create an accurate representation of the input data. Showing a high bias, i.e. the difference between the predicted value and the actual value is high.

Overfitting. The model has been overtrained and the nodes have ended up "memo-rizing" the input data. Showing a high variance, i.e. a significant difference between the error of the train set and the test set.

To conclude, if our model is correctly trained we must pay attention to the error obtained in both the train set and with the test set.
In order to face the cases of overfitting or underfitting there are a series of strategies that allow to minimize them:

In the case of underfitting. Possible strategies are to use a deeper network, use a more advanced optimization algorithm, improve its configuration or change the neural network architecture.

In the case of overfitting. It is usually related to a small dataset size, so it is reduced by increasing the number of samples in the train set or applying regularization techniques.

2.1.5 Improving Deep Neural Network (II): Optimization

As we have seen in the previous subsection, choosing the correct optimization configuration of our neural network will allow us to reduce the difference between the prediction obtained and the correct value. In this subsection we will briefly show the different options we find when optimizing a neural network.

A. Gradient descent optimizer.
The function of the optimizer is to update the weights of the nodes in order to minimize the error of the cost function. At the time of selecting the optimizer we find several options: SGD, Adagrad, RMSprop or Adam.
We pay special attention to the Adam optimizer which combines the advantages of two optimization methods that are AdaGrad and RMSProp, obtaining a robust optimizer that adapts to a large number of problems [40].

B. Variants of Gradient Descent.
Depending on the amount of data used before updating the weights, we find 3 variants of gradient descent:

Batch Gradient Descent. The cost function is not applied until all samples of the dataset have been passed, calculating the average error of the whole dataset to update each node by the backpropagation method.

Stochastic Gradient Descent. The weights are updated with each new sample.

Mini-batch Gradient descent. It is an intermediate point between the two previous ones. First the dataset is subdivided into mini-batches. Through the hyperparameter "batch-size" we configure how many samples will compose a mini-batch and the descending gradient method will calculate the average error and will update the weights once the samples that compose each mini-batch are passed.
The most common method is mini-batch gradient descent. This method combines the robustness of stochastic gradient descent and the efficiency of batch gradient descent. The "batch-size" parameter is the parameter with which it is configured.

C. Learning rate.
It is one of the hyperparameters that most affects the training process. A learning rate too small gives rise to a very slow convergence, while a learning rate that is too high cause divergence.
In addition, the optimal learning rate is not a constant value throughout the entire training process. It is important to start with a sufficiently high learning rate and reduce it as the model converges to global optimum.
To regulate and modify the learning value you can implement a learning rate scheduler. The general functioning of the various learning rate scheduler is based on the fact that after each epoch an action policy is applied in which it is defined in what circumstances it is necessary or not to reduce the value of the learning rate and in what proportion it is reduced.

D. Weight Initialization Techniques - Transfer Learning.
When building a neural network the nodes, a previous step to train the model is to initialize the weights of the nodes. There are different options ranging from Random initialization to more sophisticated methods such as applying the He or Xavier initialization techniques.
Another widely used technique with satisfactory results is called Transfer Learning. Transfer Learning consists of reusing the weights of a previously trained model against a very large dataset such as the Imagenet dataset and with those reused weights start the training process being able to distinguish certain forms and characteristics inherited from the previous training in the early stages of training.

2.1.6 Improving Deep Neural Network (III): Regularization

To cope with an increase in variance when training as a result of overfitting we have several techniques to deal with it. Next, we will show a limited number of them corresponding to the techniques that we have used in the present work.

A. Data Augmentation.
A general rule when we are training any model is that the bigger the training dataset the better performance we will achieve. But sometimes the dataset is limited and it is not possible to obtain new samples with their corresponding annotations.

B. Dropout or Dropblock.
Dropout is a regularization technique in which a number of nodes are randomly cancelled with each new sample in the training process. The ultimate goal is to reduce the interdependent learning amongst the neurons by facilitating generalization in the training process. Dropout is an effective regularization method and provides important improvements to the training process, but its use is limited to fully connected layers. In convolutional networks where information is spatially correlated, although we apply dropout information continues to flow resulting in overfitting without dropout having any beneficial effect.
For convolutional networks there is a dropout variant called Dropblock. Dropblock cancels blocks of nodes avoiding the propagation of some features maps towards later layers and with it achieving its function of regularization.

C. Early stopping.
Early stopping consists of monitoring a certain metric in the middle of the training process. When the monitoring detects that the model is not improving or on the contrary that the variance increases, at that moment the training process stops automatically.

2.2 LSTM

Sequence prediction problems have been around for a long time. They are considered as one of the hardest problems to solve in the data science industry. These include a wide range of problems; from predicting sales to finding patterns in stock markets' data, from understanding movie plots to recognizing your way of speech, from language translations to predicting your next word on your iPhone's keyboard.

With the recent breakthroughs that have been happening in data science, it is found that for almost all of these sequence prediction problems, Long short Term Memory networks, a.k.a LSTMs have been observed as the most effective solution.

LSTMs have an edge over conventional feed-forward neural networks and RNN in many ways. This is because of their property of selectively remembering patterns for long durations of time.

2.2.1 Architecture of LSTMs

The functioning of LSTM can be visualized by understanding the functioning of a news channel's team covering a murder story. Now, a news story is built around facts, evidence and statements of many people. Whenever a new event occurs you take either of the three steps.

Let's say, we were assuming that the murder was done by 'poisoning' the victim, but the autopsy report that just came in said that the cause of death was 'an impact on the head'. Being a part of this news team what do you do? You immediately forget the previous cause of death and all stories that were woven around this fact.

What, if an entirely new suspect is introduced into the picture. A person who had grudges with the victim and could be the murderer? You input this information into your news feed, right?

Now all these broken pieces of information cannot be served on mainstream media. So, after a certain time interval, you need to summarize this information and output the relevant things to your audience. Maybe in the form of "XYZ turns out to be the prime suspect.".

Now let's get into the details of the architecture of LSTM network:

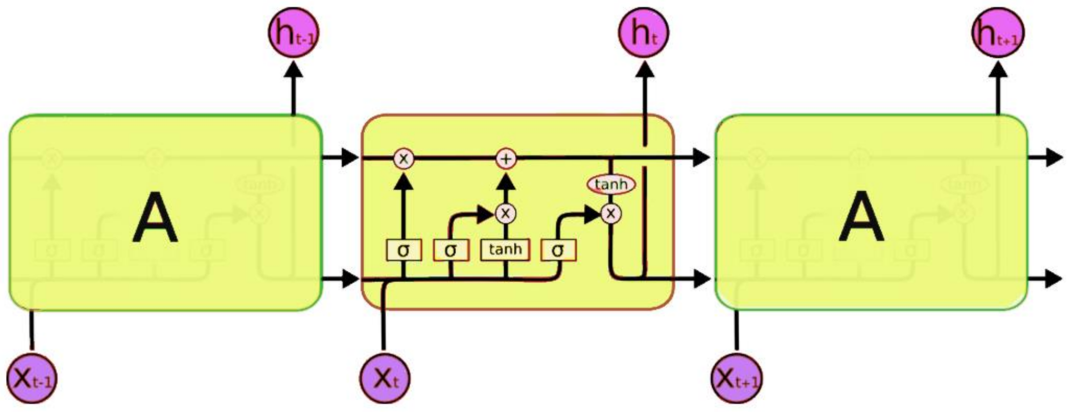


Figure 2.3: LSTM architecture.

A typical LSTM network is comprised of different memory blocks called cells (the rectangles that we see in the image). There are two states that are being transferred to the next cell; the cell state and the hidden state. The memory blocks are responsible for remembering things and manipulations to this memory is done through three major mechanisms, called gates. Each of them is being discussed below.

2.2.2 Forget Gate

Taking the example of a text prediction problem. Let's assume an LSTM is fed in, the following sentence:

Bob is a nice person. Dan on the other hand is evil.

As soon as the first full stop after "person" is encountered, the forget gate realizes that there may be a change of context in the next sentence. As a result of this, the subject of the sentence is forgotten and the place for the subject is vacated. And when we start speaking about "Dan" this position of the subject is allocated to "Dan". This process of forgetting the subject is brought about by the forget gate.

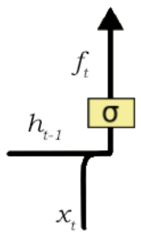


Figure 2.4: Forget gate.

A forget gate is responsible for removing information from the cell state. The information that is no longer required for the LSTM to understand things or the information that is of less importance is removed via multiplication of a filter. This is required for optimizing the performance of the LSTM network.
This gate takes in two inputs; h_t-1 and x_t.
h_t-1 is the hidden state from the previous cell or the output of the previous cell and x_t is the input at that particular time step. The given inputs are multiplied by the weight matrices and a bias is added. Following this, the sigmoid function is applied to this value. The sigmoid function outputs a vector, with values ranging from 0 to 1, corresponding to each number in the cell state. Basically, the sigmoid function is responsible for deciding which values to keep and which to discard. If a '0' is output for a particular value in the cell state, it means that the forget gate wants the cell state

to forget that piece of information completely. Similarly, a ‘1’ means that the forget gate wants to remember that entire piece of information. This vector output from the sigmoid function is multiplied to the cell state.

2.2.3 Input Gate

Okay, let’s take another example where the LSTM is analyzing a sentence:

Bob knows swimming. He told me over the phone that he had served the navy for 4 long years.

Now the important information here is that “Bob” knows swimming and that he has served the Navy for four years. This can be added to the cell state, however, the fact that he told all this over the phone is a less important fact and can be ignored. This process of adding some new information can be done via the input gate.
Here is its structure:

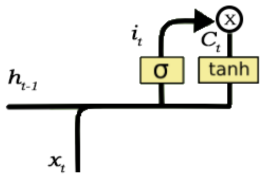


Figure 2.5: Input gate.

The input gate is responsible for the addition of information to the cell state. This addition of information is basically three-step process as seen from the diagram above. Regulating what values need to be added to the cell state by involving a sigmoid function. This is basically very similar to the forget gate and acts as a filter for all the information from h_t-1 and x_t.
Creating a vector containing all possible values that can be added (as perceived from h_t-1 and x_t) to the cell state. This is done using the tanh function, which outputs values from -1 to +1.

Multiplying the value of the regulatory filter (the sigmoid gate) to the created vector (the tanh function) and then adding this useful information to the cell state via addition operation.

Once this three-step process is done with, we ensure that only that information is added to the cell state that is important and is not redundant.

2.2.4 Output Gate

Not all information that runs along the cell state, is fit for being output at a certain time. We'll visualize this with an example:

Bob fought single handedly with the enemy and died for his country. For his contributions brave ______.

In this phrase, there could be a number of options for the empty space. But we know that the current input of 'brave', is an adjective that is used to describe a noun. Thus, whatever word follows, has a strong tendency of being a noun. And thus, Bob could be an apt output.

This job of selecting useful information from the current cell state and showing it out as an output is done via the output gate. Here is its structure:

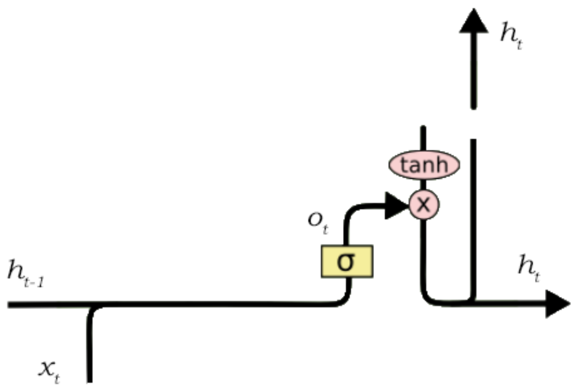


Figure 2.6: Output gate.

The functioning of an output gate can again be broken down to three steps:

Creating a vector after applying tanh function to the cell state, thereby scaling the values to the range -1 to +1.

Making a filter using the values of h_t-1 and x_t, such that it can regulate the values that need to be output from the vector created above. This filter again employs a sigmoid function.
Multiplying the value of this regulatory filter to the vector created in step 1, and sending it out as a output and also to the hidden state of the next cell.
The filter in the above example will make sure that it diminishes all other values but ‘Bob’. Thus the filter needs to be built on the input and hidden state values and be applied on the cell state vector.

LSTMs are a very promising solution to sequence and time series related problems. However, the one disadvantage that I find about them, is the difficulty in training them. A lot of time and system resources go into training even a simple model.

3 Preliminary considerations

In this chapter we will cover general aspects of our implementation.

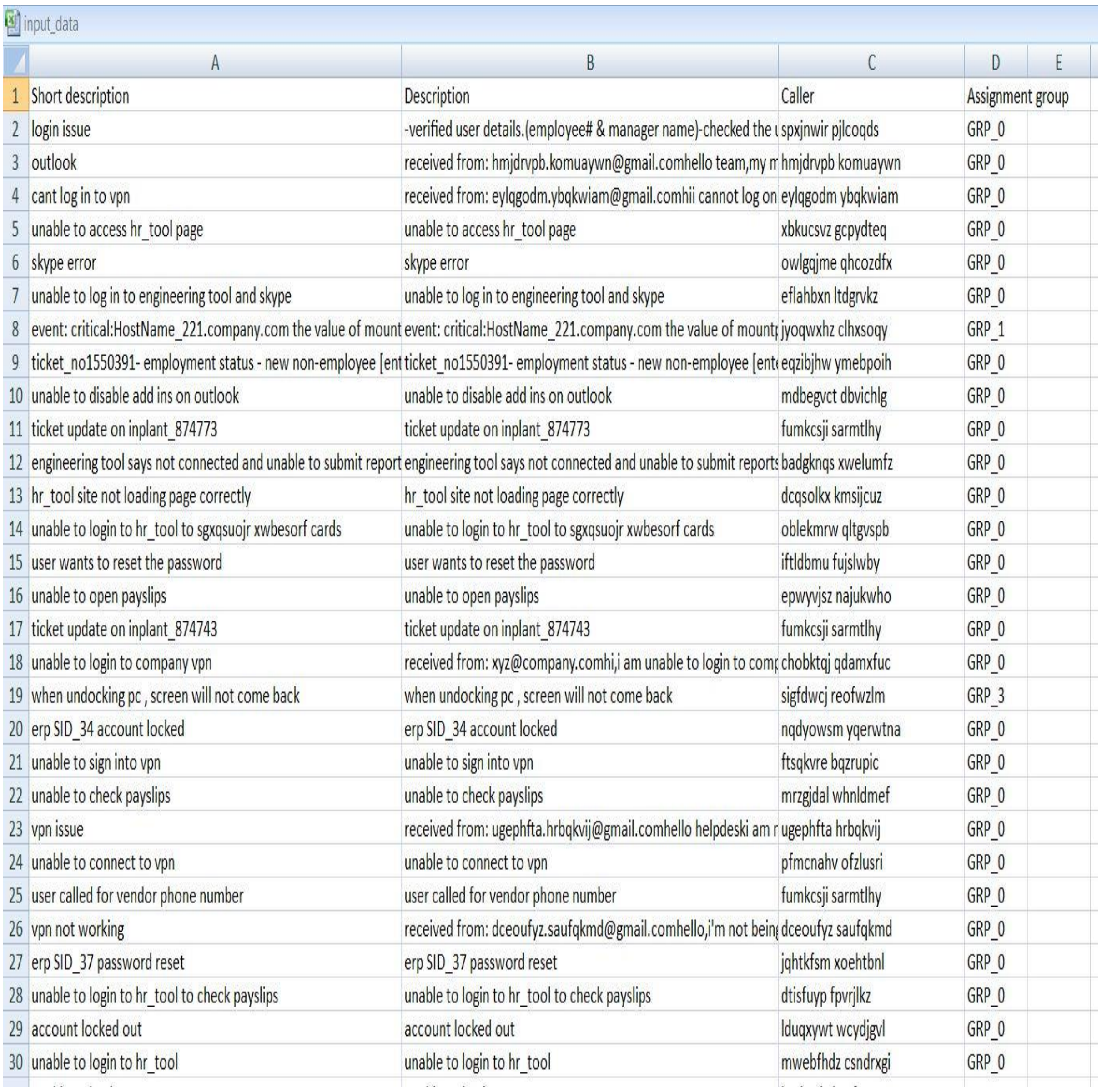
In these topics we will introduce the dataset used, the general structure of the proposed architecture, the strategy used when training and evaluating each of the Deep Learning models used and finally the metrics used.

3.1 Dataset

In this section we will focus on this dataset, which is the main dataset from which we will train and evaluate our implementation proposal.

The dataset used for this project can be found at dataset link. Here is some initial analysis:

- The dataset consists of 8500 rows and 4 columns. Each row represents information about an incident. The columns are Short description, Description, Caller and Assignment group.
- The Short description column summarizes the incident description and is short text.
- The Description column is the in detail description of the incident detailing the issue.
- The Caller column identifies the user with the firstname and lastname of the user.
- The Assignment group is the support group to which a particular incident was assigned to.
- Here is a snapshot of the dataset taken from excel



input_data

	A	B	C	D	E
1	Short description	Description	Caller	Assignment group	
2	login issue	-verified user details.(employee# & manager name)-checked the u	spxjnwir pjlcoqds	GRP_0	
3	outlook	received from: hmjdrvpb.komuaywn@gmail.comhello team,my m	hmjdrvpb komuaywn	GRP_0	
4	cant log in to vpn	received from: eylqgodm.ybqkwiam@gmail.comhii cannot log on	eylqgodm ybqkwiam	GRP_0	
5	unable to access hr_tool page	unable to access hr_tool page	xbkucsvz gcpydteq	GRP_0	
6	skype error	skype error	owlgqjme qhcozdfx	GRP_0	
7	unable to log in to engineering tool and skype	unable to log in to engineering tool and skype	eflahbxn ltdgrvkz	GRP_0	
8	event: critical:HostName_221.company.com the value of mount	event: critical:HostName_221.company.com the value of mount	jyoqwxhz clhxsoqy	GRP_1	
9	ticket_no1550391- employment status - new non-employee [ent	ticket_no1550391- employment status - new non-employee [ent	eqzibjhw ymebpoih	GRP_0	
10	unable to disable add ins on outlook	unable to disable add ins on outlook	mdbegvct dbvichlg	GRP_0	
11	ticket update on inplant_874773	ticket update on inplant_874773	fumkcsji sarmtlhy	GRP_0	
12	engineering tool says not connected and unable to submit report	engineering tool says not connected and unable to submit reports	badgknqs xwelumfz	GRP_0	
13	hr_tool site not loading page correctly	hr_tool site not loading page correctly	dcqsolkx kmsijcuz	GRP_0	
14	unable to login to hr_tool to sgxqsuojr xwbesorf cards	unable to login to hr_tool to sgxqsuojr xwbesorf cards	oblekmrw qltgvspb	GRP_0	
15	user wants to reset the password	user wants to reset the password	iftldbmu fujslwby	GRP_0	
16	unable to open payslips	unable to open payslips	epwyvjsz najukwho	GRP_0	
17	ticket update on inplant_874743	ticket update on inplant_874743	fumkcsji sarmtlhy	GRP_0	
18	unable to login to company vpn	received from: xyz@company.comhi,i am unable to login to comp	chobktqj qdamxfuc	GRP_0	
19	when undocking pc , screen will not come back	when undocking pc , screen will not come back	sigfdwcj reofwzlm	GRP_3	
20	erp SID_34 account locked	erp SID_34 account locked	nqdyowsm yqerwtna	GRP_0	
21	unable to sign into vpn	unable to sign into vpn	ftsqkvre bqzrupic	GRP_0	
22	unable to check payslips	unable to check payslips	mrzgjdal whnldmef	GRP_0	
23	vpn issue	received from: ugephfta.hrbqkvij@gmail.comhello helpdeski am r	ugephfta hrbqkvij	GRP_0	
24	unable to connect to vpn	unable to connect to vpn	pfmcnahv ofzlusri	GRP_0	
25	user called for vendor phone number	user called for vendor phone number	fumkcsji sarmtlhy	GRP_0	
26	vpn not working	received from: dceoufyz.saufqkmd@gmail.comhello,i'm not bein	dceoufyz saufqkmd	GRP_0	
27	erp SID_37 password reset	erp SID_37 password reset	jqhtkfsm xoehtbnl	GRP_0	
28	unable to login to hr_tool to check payslips	unable to login to hr_tool to check payslips	dtisfuyp fpvrjlkz	GRP_0	
29	account locked out	account locked out	lduqxywt wcydjgvl	GRP_0	
30	unable to login to hr_tool	unable to login to hr_tool	mwebfhdz csndrxgi	GRP_0	

Figure 3.1: Excel snapshot of dataset

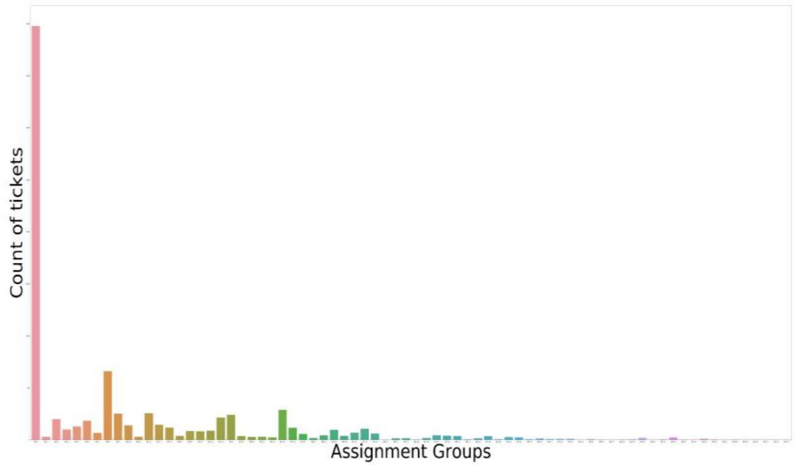
Findings from the data

Target column – The Assignment group is the target column of this problem.

3.2 Exploratory Data Analysis

Below are the observations when exploratory data analysis was performed on the given dataset –

- There are null values present for Short description and Description columns in the dataset. Specifically 8 incidents do not have values for Short description and 1 entry does not have a value for Description column.
- There are in total 74 unique support groups named from GRP_0 to GRP_73 present in the database. Analysis shows that there are 3976 (46%) incidents assigned to GRP_0 and the rest 4524 among the rest 73 groups. This makes the data highly disproportionate. The ticket distribution for top 10 assignment groups is shown below –



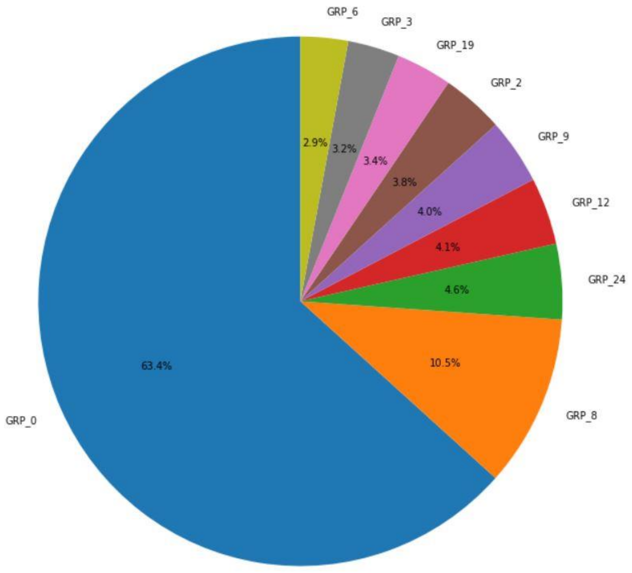
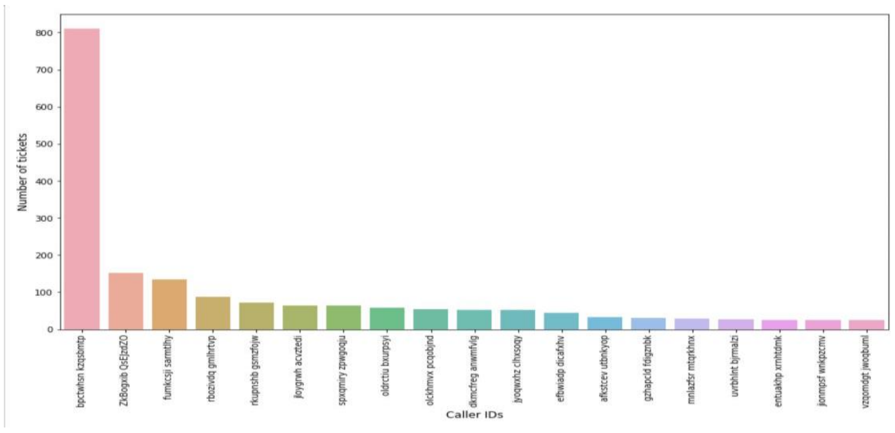


Figure 3.2. Assignment group wise ticket distribution

- There are 2950 unique users that have reported the incidents and the topmost user has 810 (27%) number of incidents raised. The ticket distribution for top 10 users is as shown below –



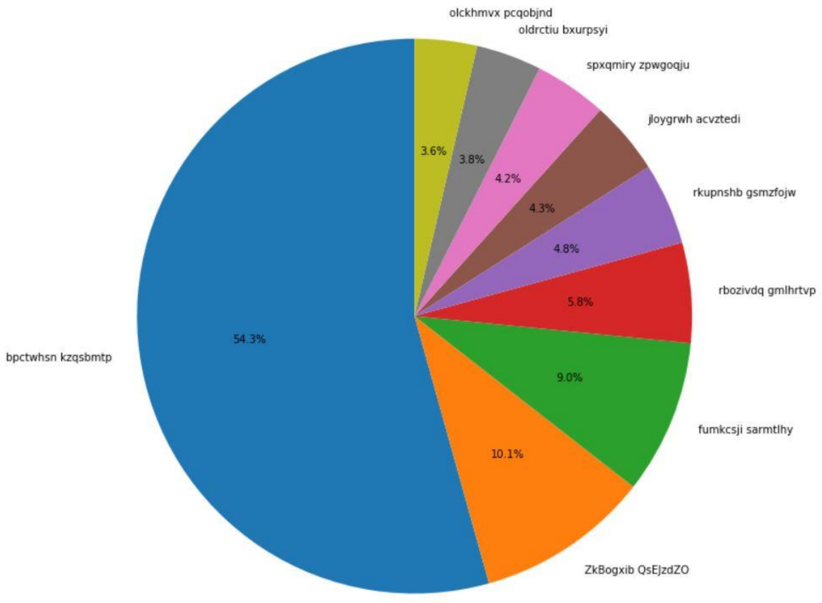


Figure 3.3: User wise ticket distribution

- There are many tickets that have been repeated in the Short Description column. The most common ticket raised by any caller is for 'password reset' .The following are the top20 tickets that has been repeated by the callers.

•

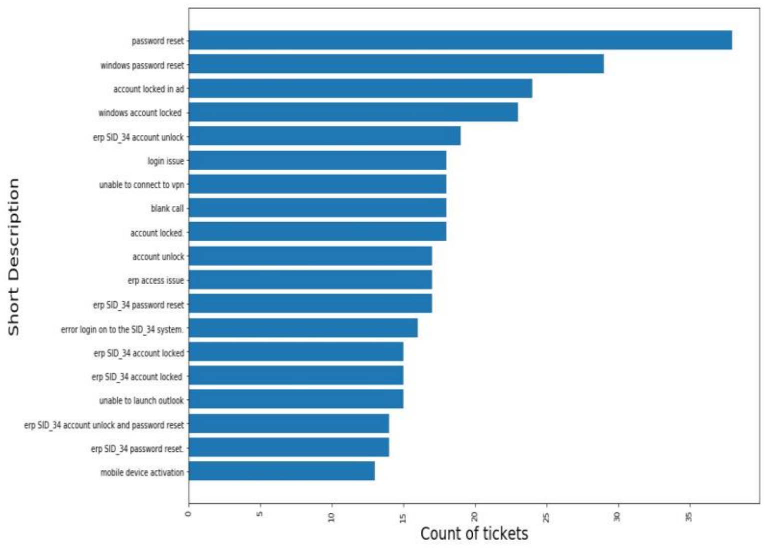


Figure 3.4: Short description wise ticket distribution

- See figure below for the word cloud data for Short description column –

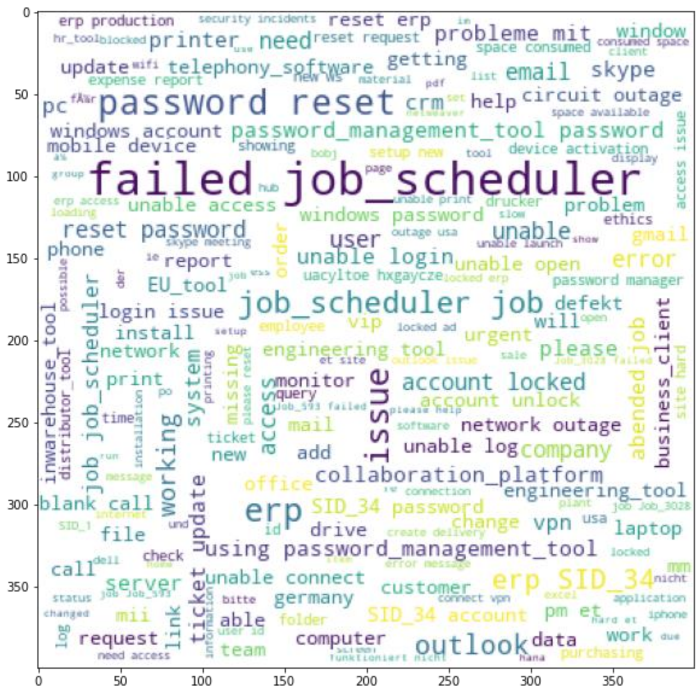


Figure 3.5: Word cloud for Short description column before preprocessing.

- See figure below for the word cloud data for Description column –

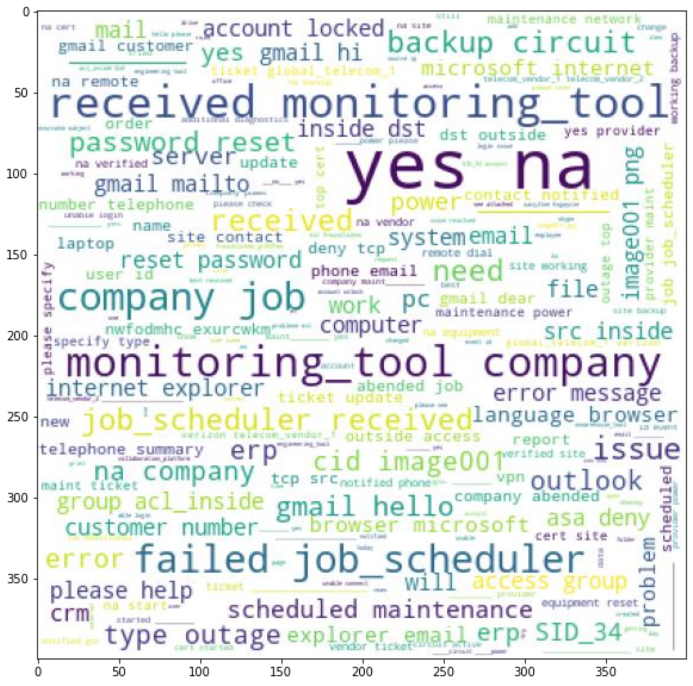


Figure 3.6: Word cloud for Description column before pre-processing.

The description column for most of the tickets also consists of text like “Reported by: emailid”. This can be removed as callers email id has no relationship with the target column.

Upon analysis of data we found that some of the phrases like disclaimers were present in some of the entries We created a list of such disclaimers which can be removed from the model training process as these do not affect the target column. Below is an example disclaimer -

```
this communication (including any accompanying documents) is inte
nded only for the sole use of the person(s) to whom it is address
ed and may contain information that is privileged,confidential an
d exempt from disclosure. any unauthorised reading,dissemination
,distribution,duplication of this communication by someone other
than the intended recipient is strictly prohibited. if your recei
pt of this communication is in error,please notify the sender and
destrtgoy the original communication immediately','please do not
print this email unless it is absolutely necessary. spread enviro
nmental awareness
```

Few words are repeated throughout the data and are visible in the word cloud as well. These common words can be treated as stop words and can be removed from the dataset. Few examples are –

```
'yes','no','na','mii','hello','regards','thanks','thankyou'
```

3.3 General structure of the proposed implementation

We have implemented below two modules in this project:

Preprocessing Module. This module includes the preprocessing actions that we will apply to each and every input sample with which we feed the models during the training and evaluation processes.

Classification Module. Module specialized exclusively in ticket classification.

An exclusive chapter will be dedicated to each of the modules mentioned above, where we will describe them in more detail.

3.4 Training and prediction strategies

In this section we will give some brushstrokes of the strategy followed when training the multiple models that are part of the classification module. There are two basic starting points:

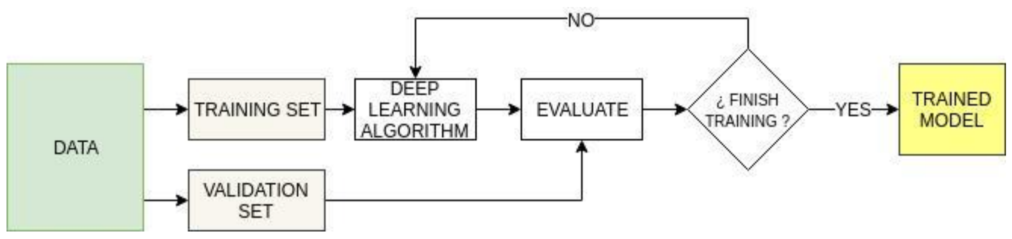


Figure 3.7: Model training process.

The models trained will be mainly LSTM neural networks which require supervised learning.

The dataset used is divided into 3 portions: train, validation and test. In the training process we will use only the training and validation portions. Test set will be reserved to evaluate the trained models and in the processes of combination of predictions and models.

Figure 3.7 shows the workflow of the training process. In this process only the training and validation subsets act. After each epoch the current model will be evaluated using the validation set looking for optimal values of bias and variance. In both the training and detection modules we will train multiple models.

3.5 Metrics for classification module

This section will describe the main metrics used to evaluate the performance of the classification module.
Into research community there is an extensive list of possible metrics, which may be used to measure the performances of an architecture or implementation. Although in the training process we compute the error and accuracy against the cost function, it is normal that in order to evaluate the performance obtained with guarantees, another more advanced metric is used. Depending on the application and the problem, a mistake or another can significantly impact your performance and using the most widely used metric for that type of problem allows you to compare with other implementations using the same metric.
We have used the accuracy as a metric score and have monitored loss in our model evaluation step.

4 Preprocessing module

In this chapter we will focus on the actions applied on the dataset samples as a previous step to feed the models of the classification modules both in the training and prediction process. In Figure 4.1 we group in a flowchart all the actions applied in this module and that we will explain in detail in the following sections.

4.1 Process Overview

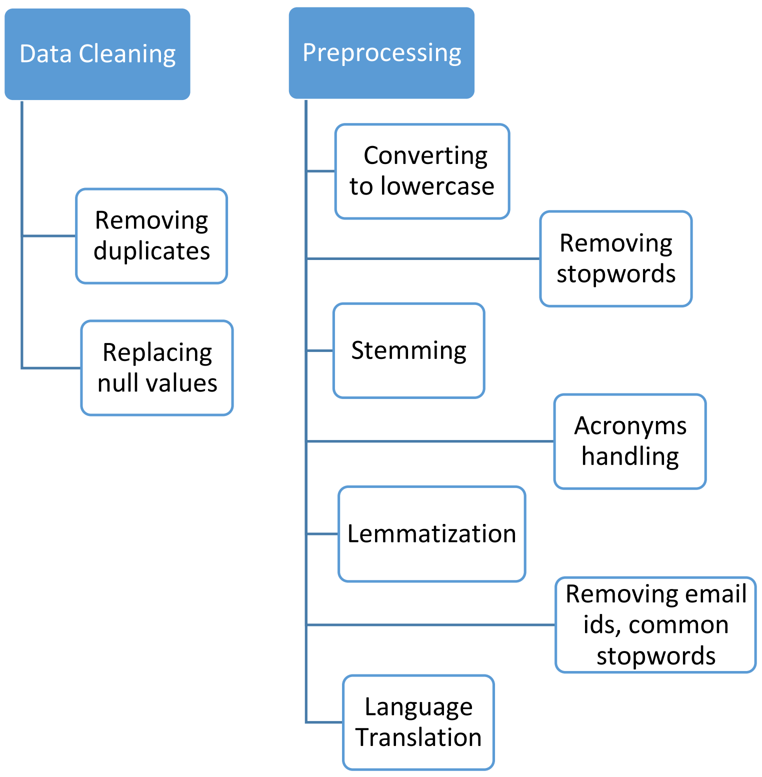


Figure 4.1 : Data cleaning process overview

The fundamental aspect of machine learning that determines how good the ML model can perform is Data. If data is insufficient or is not processed correctly to target the problem, we cannot expect the model to perform well.

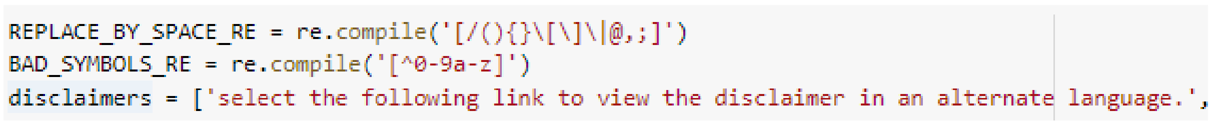
We have identified following preprocessing steps for the given dataset –

- The first step is to get rid of null values from the dataset. As noted earlier there are 9 rows that null values for either of Short Description and Description columns. We will replace the null values with stop words.
- There are 83 duplicate entries in the dataset, which can be safely removed.
- As noted earlier, we can get rid of text like “Reported by emailid” from the Description column. To build the email id for each user we can use the first name and last name information from caller column.
- The data is highly imbalanced as a lot of entries correspond to GRP_0. Also there are 6 groups for which only one entry is present. There are other groups with very few entries.

4.2 Clean_text function

We have consolidated all the remaining pre-processing texts into a single function called Clean_text. Here is a step by step walkthrough of the Clean_text function –

At the outset of the Clean_text function we have created different variables that hold characters/words that need to be replaced in the dataset.



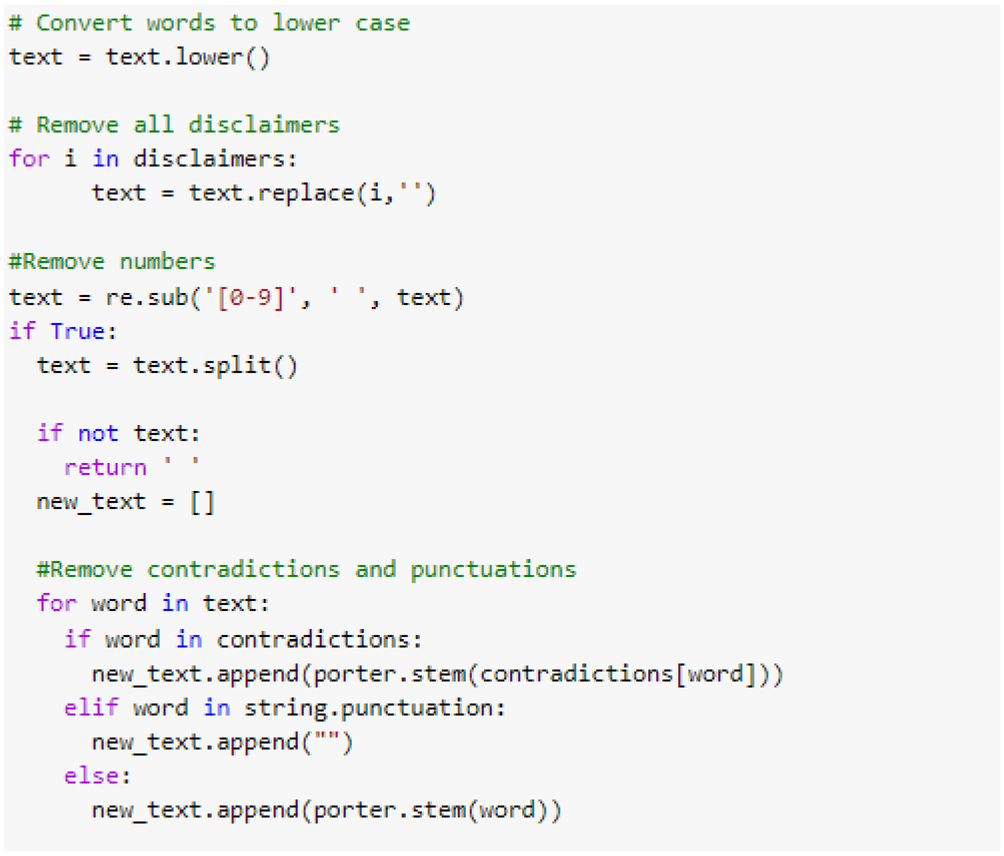
```
REPLACE_BY_SPACE_RE = re.compile('[/(){}\[\]\|@,;]')
BAD_SYMBOLS_RE = re.compile('[^0-9a-z]')
disclaimers = ['select the following link to view the disclaimer in an alternate language.',
```

- Newline and other special characters like ‘@’ are handled in REPLACE_BY_SPACE_RE variable. We will replace such characters by spaces in the Clean_text function.
- BAD_SYMBOLS_RE defines all the other non alphanumeric text. This text will also be replaced by spaces.
- Disclaimers defines all the disclaimers found in the dataset. These will be replaced by empty string.

We have covered following steps in this function

- Convert all text to lowercase.
- Remove all disclaimers identified in the data.
- Remove all numbers from the data.
- Tokenize the text to get words from the dataset.

- Replace all contradictions to a single key.
- Remove all punctuations from the data.
- Stem all words in the data using Porter Stemming - The Porter stemming algorithm (or ‘Porter stemmer’) is a process for removing the commoner morphological and in-flexional endings from words in English. Its main use is as part of a term normalization process that is usually done when setting up Information Retrieval systems.

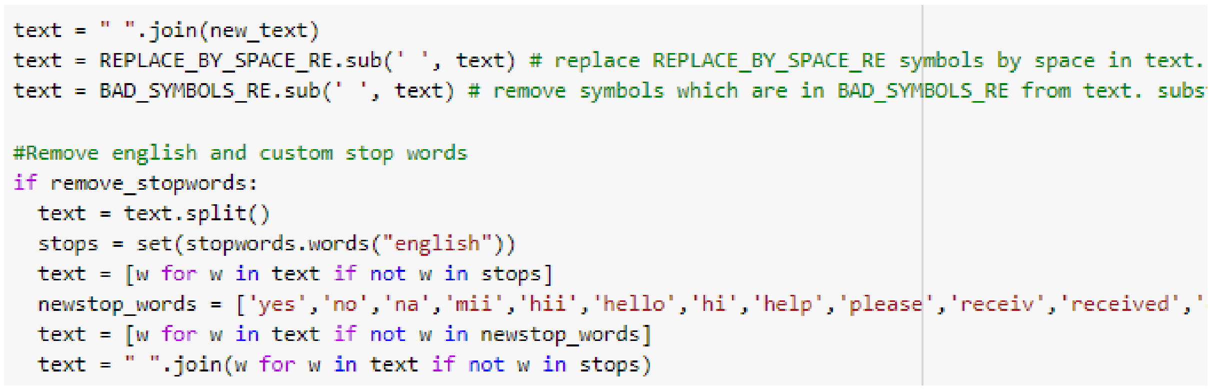


```
# Convert words to lower case
text = text.lower()

# Remove all disclaimers
for i in disclaimers:
      text = text.replace(i,'')

#Remove numbers
text = re.sub('[0-9]', ' ', text)
if True:
  text = text.split()

  if not text:
    return ' '
  new_text = []

  #Remove contradictions and punctuations
  for word in text:
    if word in contradictions:
      new_text.append(porter.stem(contradictions[word]))
    elif word in string.punctuation:
      new_text.append("")
    else:
      new_text.append(porter.stem(word))
```

The next steps are as follows –

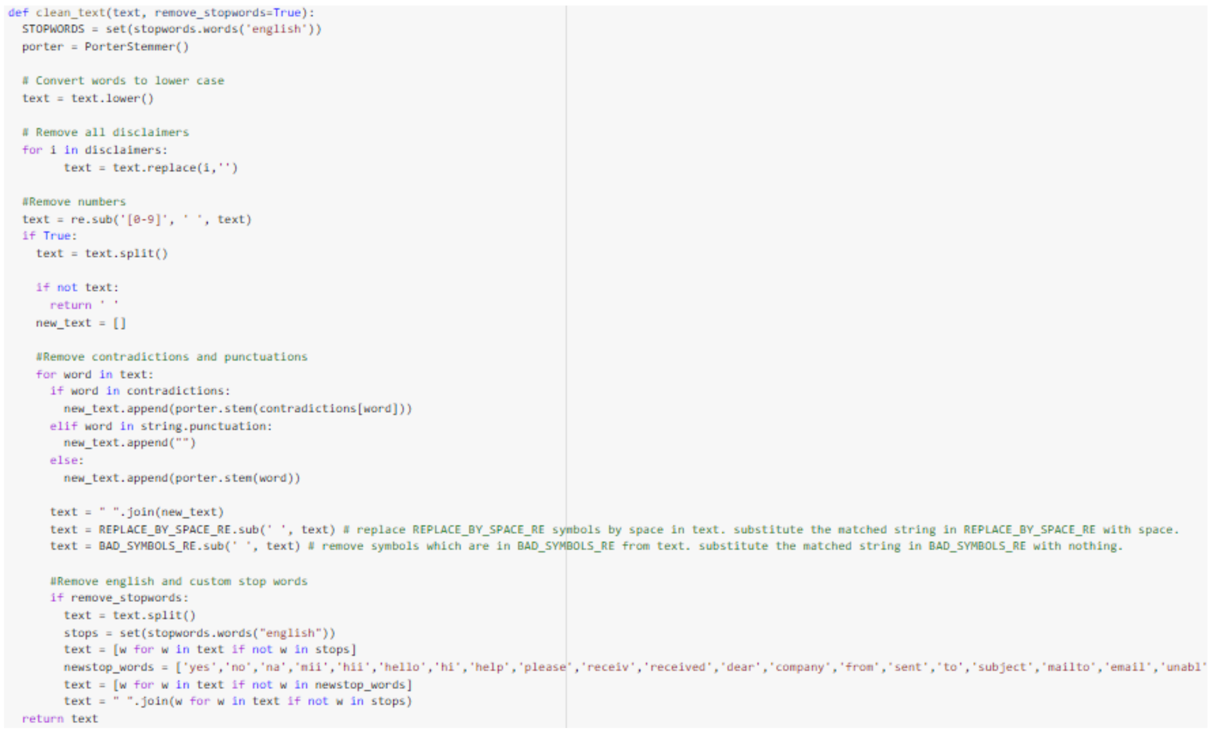
- Replace the REPLACE_BY_SPACE_RE and BAD_SYMBOLS_RE with spaces
- Remove stop words.

For stop words removal we have removed the NLTK stop words. Also we have removed the common words that are found in the dataset.



```
text = " ".join(new_text)
text = REPLACE_BY_SPACE_RE.sub(' ', text) # replace REPLACE_BY_SPACE_RE symbols by space in text.
text = BAD_SYMBOLS_RE.sub(' ', text) # remove symbols which are in BAD_SYMBOLS_RE from text. subs

#Remove english and custom stop words
if remove_stopwords:
  text = text.split()
  stops = set(stopwords.words("english"))
  text = [w for w in text if not w in stops]
  newstop_words = ['yes','no','na','mii','hii','hello','hi','help','please','receiv','received','
  text = [w for w in text if not w in newstop_words]
  text = " ".join(w for w in text if not w in stops)
```

Here is the complete clean_text function –



```
def clean_text(text, remove_stopwords=True):
  STOPWORDS = set(stopwords.words('english'))
  porter = PorterStemmer()

  # Convert words to lower case
  text = text.lower()

  # Remove all disclaimers
  for i in disclaimers:
        text = text.replace(i,'')

  #Remove numbers
  text = re.sub('[0-9]', ' ', text)
  if True:
    text = text.split()

    if not text:
      return ' '
    new_text = []

    #Remove contradictions and punctuations
    for word in text:
      if word in contradictions:
        new_text.append(porter.stem(contradictions[word]))
      elif word in string.punctuation:
        new_text.append("")
      else:
        new_text.append(porter.stem(word))

      text = " ".join(new_text)
      text = REPLACE_BY_SPACE_RE.sub(' ', text) # replace REPLACE_BY_SPACE_RE symbols by space in text. substitute the matched string in REPLACE_BY_SPACE_RE with space.
      text = BAD_SYMBOLS_RE.sub(' ', text) # remove symbols which are in BAD_SYMBOLS_RE from text. substitute the matched string in BAD_SYMBOLS_RE with nothing.

      #Remove english and custom stop words
      if remove_stopwords:
        text = text.split()
        stops = set(stopwords.words("english"))
        text = [w for w in text if not w in stops]
        newstop_words = ['yes','no','na','mii','hii','hello','hi','help','please','receiv','received','dear','company','from','sent','to','subject','mailto','email','unabl'
        text = [w for w in text if not w in newstop_words]
        text = " ".join(w for w in text if not w in stops)
  return text
```

After data cleansing below is the word cloud data for short description column –

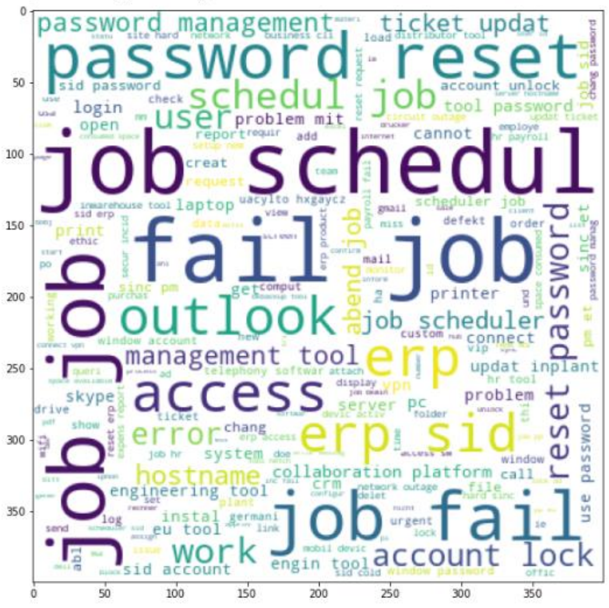


Figure 4.2: word cloud data for Short Description column after data cleansing.

See figure below for the word cloud for Description column –

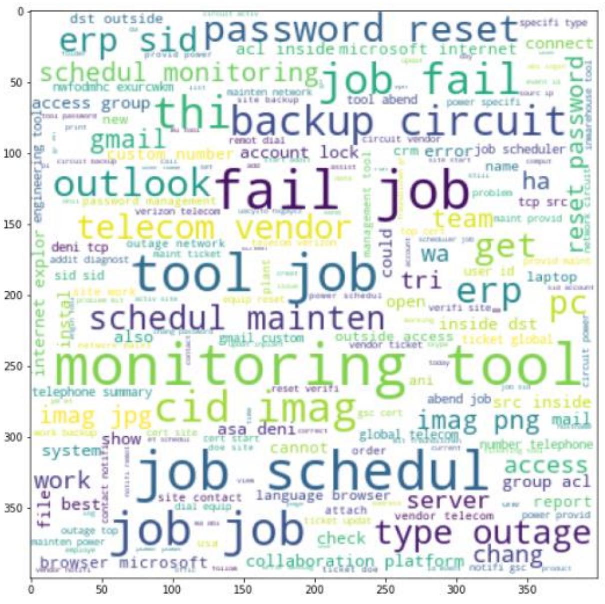


Figure 4.3: word cloud data for Description column.

4.3 Handling languages other than English

Upon manual analysis of the input excel, we found that some languages other than English were also present. We needed a way to detect and handle such records gracefully.

4.3.1 Fasttext library

We are building a model that supports only English text. This is done as there are a very small number of records that are from other languages
Non-English words are out of vocabulary to such model, and it does not handle it well. To cater to this need we found Fasttext library for language detection.

Fasttext is an open-source, free, lightweight library that allows users to learn text representations and text classifiers. It works on standard, generic hardware. Models can later be reduced in size to even fit on mobile devices.

Here is the **link** to the Fasttext model. The compressed version of the model is just a little shy of 1MB and supports 176 languages.
The output of the language detection model is a tuple of language label and prediction confidence. Language label is a string with "__lable__" followed by ISO 639 code of the language.

We used Fasttext library and used a pre-trained model to predict the language from the description of an incident. Further detailing revealed that the model made mistakes of predicting other languages due to presence of special characters in them.

We have used a threshold of 50% to indicate that we only consider the languages as other than English if the confidence output from model is over 50% value.

Below is the result -

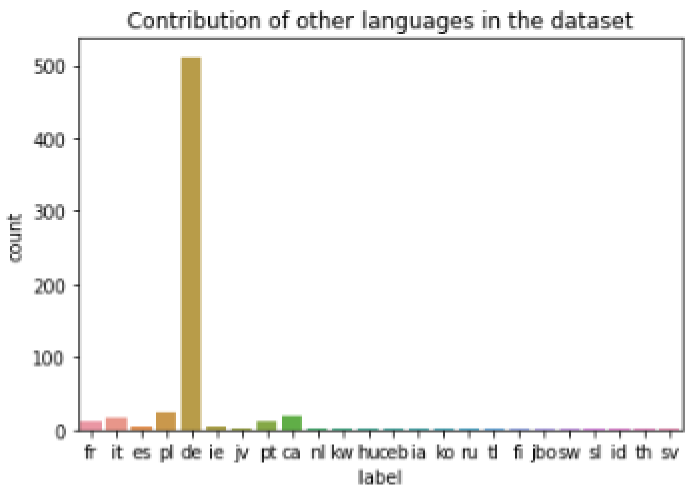


Figure 4.4: Contribution of languages other than English.

Upon analysis of data, other languages apart from English were found. A total of 631 records are present for other languages. It is observed that 512 records belong to German language and the rest of 119 belong to other languages. The incidents belonging to languages other than English is just 7.5% of whole dataset. Hence we will discard these incidents from our dataset.

4.4 Merging different features

To work on this problem we are going to use the transfer learning approach based on GloVe Embeddings. More on this in coming sections. In order to achieve this we need a single corpus on which our models could be trained to predict the assignment class. Hence we merge all the features leaving the target column, into a new column called as MergedColumn. This will be the single feature to our classification models.

```
data_df['MergedColumn'] = data_df[data_df.columns[0:3]].apply(
    lambda x: ' '.join(x.astype(str)),
    axis=1
)
data_df = data_df.drop(['Short description','Description','Caller'],axis=1)
data_df.head()
```

	Assignment group	MergedColumn
0	GRP_0	login verifi user details employee manag name ...
1	GRP_0	outlook team meetings skyp meet etc appear out...
2	GRP_0	cant log vpn cannot log vpn best eylqgodm ybqk...
3	GRP_0	access hr tool page access hr tool page xbkucs...
4	GRP_0	skype error skype error owlgqjme qhcozdfx

4.5 Splitting Method

As we have seen in the previous chapter the dataset consists of a total of 8500 samples.
Following the training strategy defined in section 3.4, we must group the samples with their corresponding annotations into 3 portions: training set, validation set and test set. Although this task may seem trivial, not performing a correct partition of the dataset will have negative effects that will affect continuously throughout the whole process of both training and evaluation. In order to reduce the effects of underfitting and overfitting, the partitioning method must ensure that each portion is a good representative of the whole, i.e. that each portion maintains a similar percentage of each class.
We used StratifiedShuffleSplit as splitting method but as few groups only have a single entry in them it makes it ipossible to use this method.
Hence we have used TrainTestSplit from scikit learn library.

The percentage of the entire dataset in each serving has been established as follows:
Train Set: 80%.
Test Set : 20%.

While running the training process we further divide the train dataset in to validation set with 90/10 split.

5 Classification module

In this chapter we will focus on the development and evaluation of the performance of the classification module. This module as its name indicates specializes in the task of classification providing a global prediction of all the ticket assignment groups. Given a sample in ticket we will obtain the likely Assignment group. We will also validate the performance of traditional ML models in this section.

When designing this module we started from the following premises:

- The objective of the module is to provide a classification of assignment group to an incoming incident
- There is high imbalance of data with 43% entries for Group_0.

5.1 Multiclass-classification problem

In this first stage of the classification module we formulate the problem as a multi-class classification task, i.e. given an input sample x, it should be classified to one and just one from the predefined classes y.

In the following, we will describe the traditional ML models and an LSTM model. And finally, each of the models has been individually trained with the same data set (train set and validation set).

We will conclude with a brief on Binary classification aspect.

5.1.1 Logistic regression

Contrary to popular belief, Logistic regression is a traditional and classic statistical & regression model, which has been widely used in the academy and industry. The model builds a regression model to predict the probability that a given data entry belongs to the category numbered as “1”. Just like linear regression assumes that the data follows a linear function, Logistic regression models the data using the sigmoid function.

$$f(x) = \frac{1}{1 + e^{-x}}$$

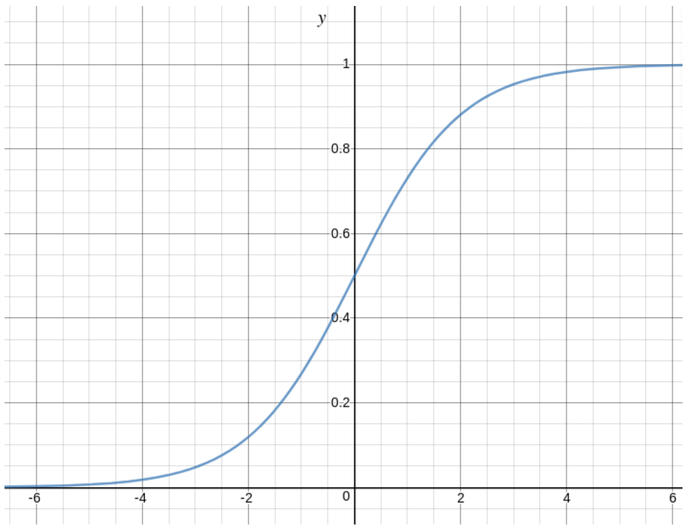
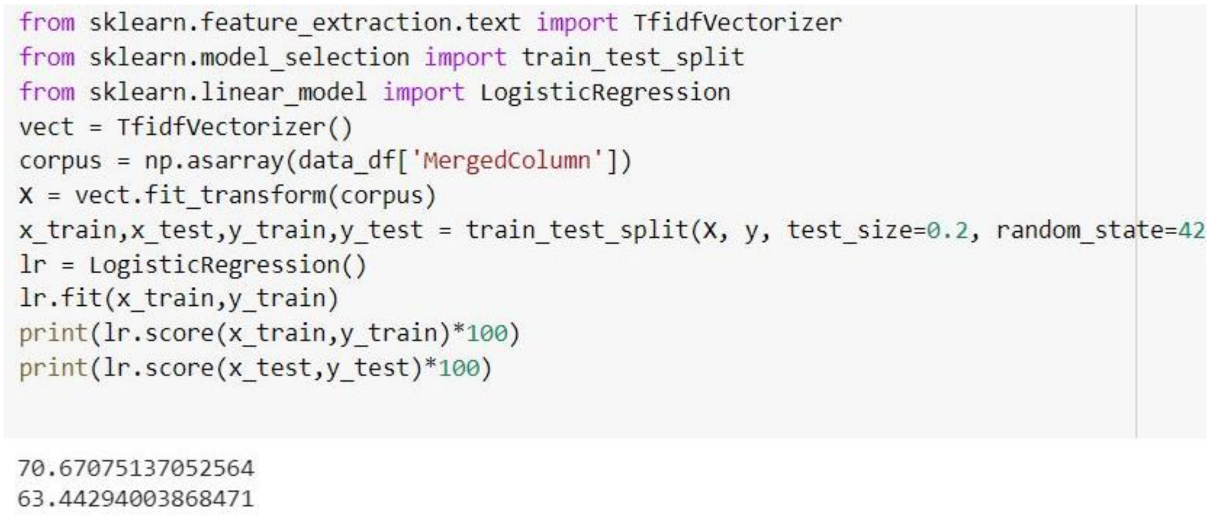


Figure 5.1: Logistic regression function

Logistic regression is basically a supervised classification algorithm. In a classification problem, the target variable (or output), y, can take only discrete values for given set of features (or inputs), X.

The sigmoid function gives an S-shaped curve and saturates when its argument is very positive or very negative. Take a moment to note down the formula. We will apply it later in the maximum likely estimation.

Logistic regression becomes a classification technique only when a decision threshold is brought into the picture. The setting of the threshold value is a very important aspect of Logistic regression and is dependent on the classification problem itself.



```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
vect = TfidfVectorizer()
corpus = np.asarray(data_df['MergedColumn'])
X = vect.fit_transform(corpus)
x_train,x_test,y_train,y_test = train_test_split(X, y, test_size=0.2, random_state=42
lr = LogisticRegression()
lr.fit(x_train,y_train)
print(lr.score(x_train,y_train)*100)
print(lr.score(x_test,y_test)*100)
```

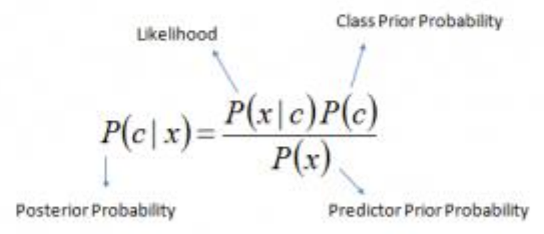
```
70.67075137052564
63.44294003868471
```

5.1.2 Naive Bayes

It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.

Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.

Bayes theorem provides a way of calculating posterior probability P(c|x) from P(c), P(x) and P(x|c).

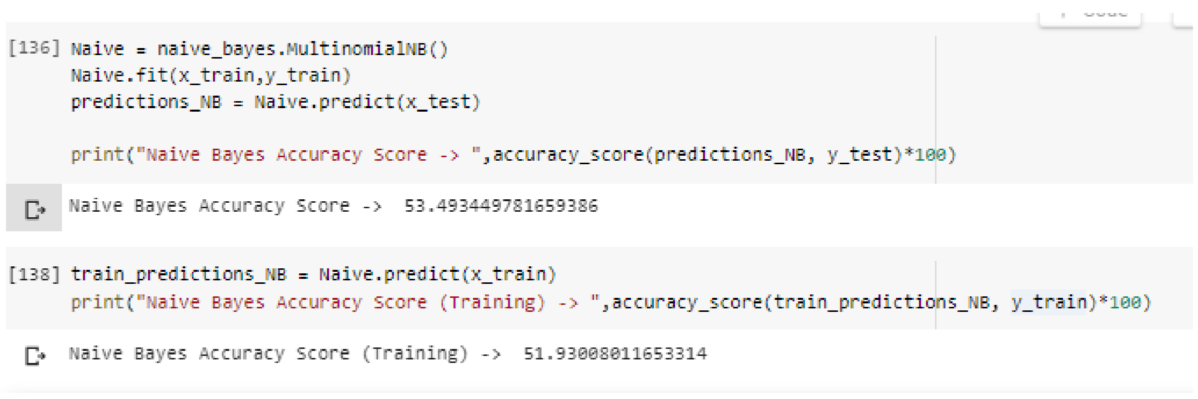

Likelihood
Class Prior Probability
$P(c \mid x) = \frac{P(x \mid c) P(c)}{P(x)}$
Posterior Probability
Predictor Prior Probability

$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

- P(c|x) is the posterior probability of class (c, target) given predictor (x, attributes).
- P(c) is the prior probability of class.
- P(x|c) is the likelihood which is the probability of predictor given class.
- P(x) is the prior probability of predictor.

Scikit learn (python library) will help here to build a Naive Bayes model in Python. There are three types of Naive Bayes model under the scikit-learn library.

- Gaussian
- Multinomial
- Bernoulli



```
[136] Naive = naive_bayes.MultinomialNB()
      Naive.fit(x_train,y_train)
      predictions_NB = Naive.predict(x_test)

      print("Naive Bayes Accuracy Score -> ",accuracy_score(predictions_NB, y_test)*100)

      Naive Bayes Accuracy Score ->  53.493449781659386

[138] train_predictions_NB = Naive.predict(x_train)
      print("Naive Bayes Accuracy Score (Training) -> ",accuracy_score(train_predictions_NB, y_train)*100)

      Naive Bayes Accuracy Score (Training) ->  51.93008011653314
```

5.1.3 K-Nearest Neighbours

K-Nearest Neighbours is one of the most basic yet essential classification algorithms in Machine Learning. It belongs to the supervised learning domain and finds intense application in pattern recognition, data mining and intrusion detection.

It is widely disposable in real-life scenarios since it is non-parametric, meaning, it does not make any underlying assumptions about the distribution of data
We are given some prior data (also called training data), which classifies coordinates into groups identified by an attribute. Below is the set of Training data.

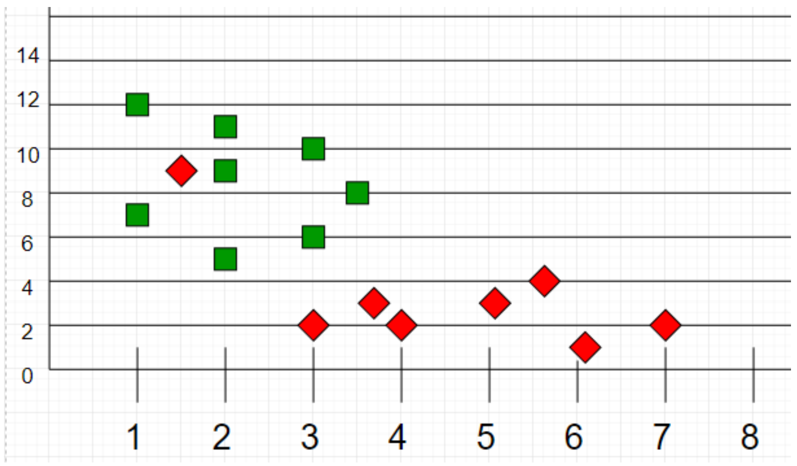


Figure 5.2: Training data for KNN

Now, given another set of data points (also called testing data), allocate these points a group by analysing the training set. Note that the unclassified points are marked as ‘White’.

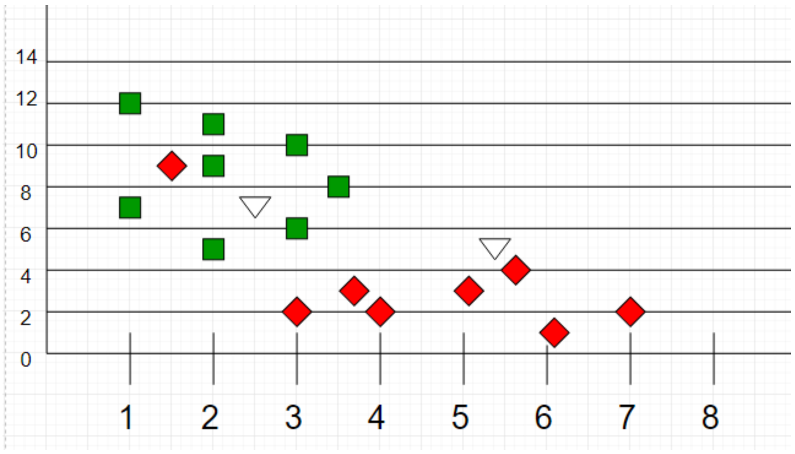
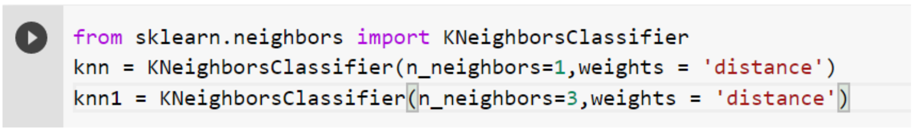


Figure 5.3: Result of KNN classification.

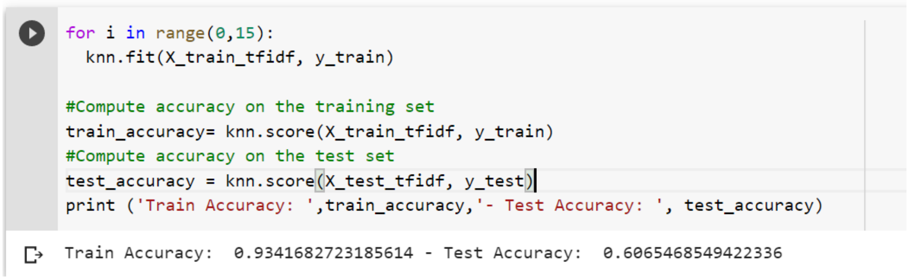
The k-NN classifier calculates the distances between the point and points in the training data set. Usually, the Euclidean distance is used as the distance metric. Then, it assigns the point to the class among its k nearest neighbours. Intuitively, we can see that the first point (2.5, 7) should be classified as ‘Green’ and the second point (5.5, 4.5) should be classified as ‘Red’.

With increasing K, we get smoother, more defined boundaries across different classifications. Also, the accuracy of the above classifier increases as we increase the number of data points.

Below are the details of model built, trained & checked performance:

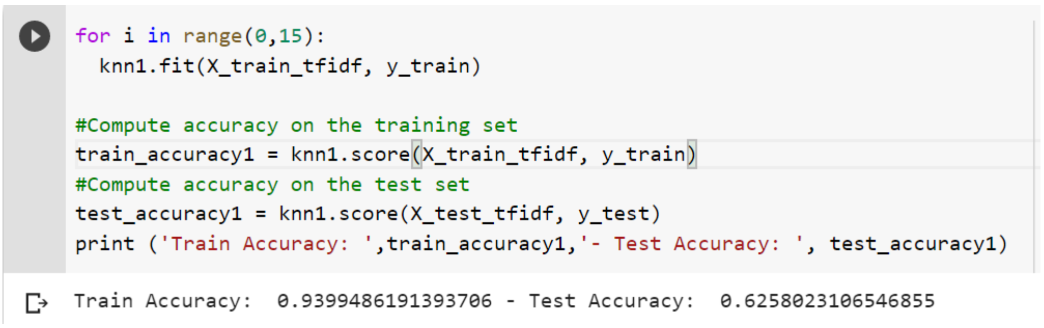


```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1,weights = 'distance')
knn1 = KNeighborsClassifier(n_neighbors=3,weights = 'distance')
```



```
for i in range(0,15):
  knn.fit(X_train_tfidf, y_train)

#Compute accuracy on the training set
train_accuracy= knn.score(X_train_tfidf, y_train)
#Compute accuracy on the test set
test_accuracy = knn.score(X_test_tfidf, y_test)
print ('Train Accuracy: ',train_accuracy,'- Test Accuracy: ', test_accuracy)
```

```
Train Accuracy:  0.9341682723185614 - Test Accuracy:  0.6065468549422336
```



```
for i in range(0,15):
  knn1.fit(X_train_tfidf, y_train)

#Compute accuracy on the training set
train_accuracy1 = knn1.score(X_train_tfidf, y_train)
#Compute accuracy on the test set
test_accuracy1 = knn1.score(X_test_tfidf, y_test)
print ('Train Accuracy: ',train_accuracy1,'- Test Accuracy: ', test_accuracy1)
```

```
Train Accuracy:  0.9399486191393706 - Test Accuracy:  0.6258023106546855
```

Test Accuracy for k=1 is 60.65% & k=3 is 62.58%.

5.1.4 Support Vector Machines (SVM)

In machine learning, Support Vector Machines (SVM) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. A SVM is a discriminative classifier formally defined by a separating hyper plane. The algorithm creates a line or a hyper plane which separates the data into classes

In other words, given labelled training data (supervised learning), the algorithm outputs an optimal hyper plane which categorizes new examples.

An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible.

Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier. Considering the dataset with 2 classes (Red & Blue),

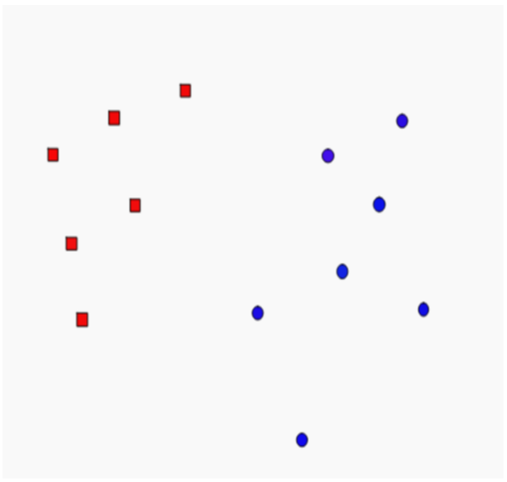


Figure 5.4: Training samples.

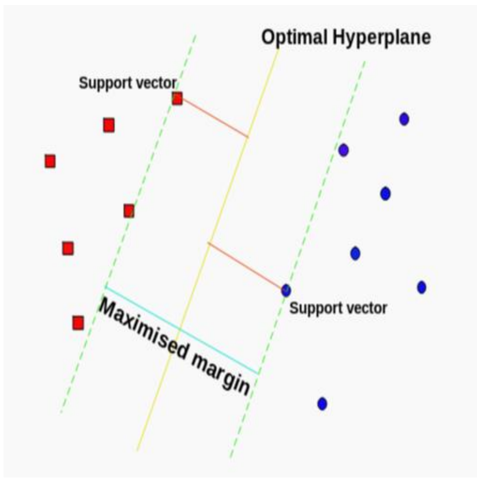


Figure 5.5: SVM classifier.

SVM tries to make a decision boundary in such a way that the separation between the two classes is as wide as possible.

```
[ ]  Encoder = LabelEncoder()
     Train_Y = Encoder.fit_transform(Train_Y)
     Test_Y = Encoder.fit_transform(Test_Y)
```

```
[ ]  Tfidf_vect = TfidfVectorizer(max_features=5000)
     Tfidf_vect.fit(new_data_df['MergedColumn'])
     Train_X_Tfidf = Tfidf_vect.transform(Train_X)
     Test_X_Tfidf = Tfidf_vect.transform(Test_X)
```

```
SVM = svm.SVC(C=1.0, kernel='linear', degree=3, gamma='auto')
SVM.fit(Train_X_Tfidf,Train_Y)
predictions_SVM = SVM.predict(Test_X_Tfidf)
prediction_SVM_train = SVM.predict(Train_X_Tfidf)
print("SVM Train Accuracy Score -> ",accuracy_score(prediction_SVM_train, Train_Y)*100)
print("SVM Test Accuracy Score -> ",accuracy_score(predictions_SVM, Test_Y)*100)
```

```
SVM Train Accuracy Score ->  81.15010104721661
SVM Test Accuracy Score ->  55.78406169665809
```

5.1.5 Long Short Term Memory (LSTM)

Long Short Term Memory networks are a special kind of RNN, capable of learning long-term dependencies. LSTM is the most powerful and well known type of artificial neural network designed to recognize patterns in sequences of data.

LSTM is explained in detail in Section 2. Here we have used a basic LSTM model and ran it for 10 epochs.

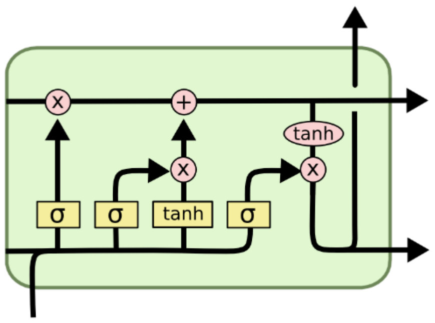
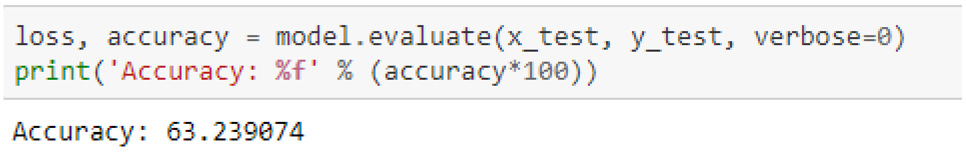


Figure 5.6: A basic LSTM cell

```
model = Sequential()
model.add(Embedding(num_words, embedding_size, weights = [embedding_matrix]))
model.add(SpatialDropout1D(0.2))
model.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(74, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```



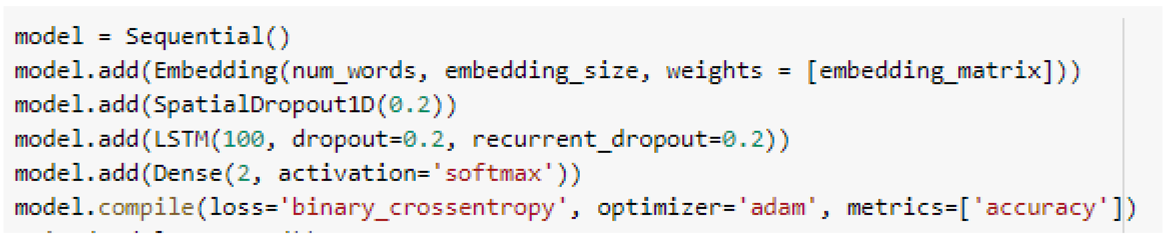
```
loss, accuracy = model.evaluate(x_test, y_test, verbose=0)
print('Accuracy: %f' % (accuracy*100))
```

```
Accuracy: 63.239074
```

The accuracy of the model on test set is 63.23%.

5.2 Binary classification problem

Since the dataset is highly imbalanced and almost 46% data is of group 0. We grouped all the other groups into one and ran a binary LSTM classifier with binary_crossentropy as loss metric.



```
model = Sequential()
model.add(Embedding(num_words, embedding_size, weights = [embedding_matrix]))
model.add(SpatialDropout1D(0.2))
model.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(2, activation='softmax'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Test accuracy = 81.22%
Train accuracy = 98.63%

6 Complete final architecture

Between chapters 4 and 5 we have gone through all the modules that compose the constructed diagnosis architecture, describing the design aspects and the most important actions that take place. In this itinerary we have also progressively evaluated each of the modules separately.
In this chapter we unify all the modules showing the complete implementation, doing a concise review of each module. And we will evaluate the possible improvement provided by this approach.
Finally, we will evaluate the final results obtained by the full implementation in context against similar works.

6.1 Architecture Design Summary

Next, we will make a brief review of each of the modules that compose the complete architecture.

1. Preprocessing Module. This module is the first contact of the architecture with the input sample. It processes the incoming data and sanitizes it for consumption by our classification module.
2. Classification Module. Specialized module to classify incoming incidents.

6.2 Approach

Here's how we will solve the classification problem:

- Convert all text samples in the dataset into sequences of word indices. A "word index" would simply be an integer ID for the word. We will only consider the top 6111 most commonly occurring words in the dataset, and we will truncate the sequences to a maximum length of 200 words.
- Prepare an "embedding matrix" which will contain at index i the embedding vector for the word of index i in our word index.

- Load this embedding matrix into a Keras Embedding layer
- Build on top of it a unidirectional LSTM network, ending in a softmax output over our 74 categories

Then we can format our text samples and labels into tensors that can be fed into a neural network. To do this, we will rely on Keras utilities keras.preprocessing.text.Tokenizer and keras.preprocessing.sequence.pad_sequences.

6.3 Model building

Once the data pre-processing steps are completed, the next step is to build the model itself. The model that we have used as the final model is a unidirectional LSTM model with GloVe embeddings.

6.3.1 Word indexing/ Vectorization

In this step, the text is vectorized, by turning each text into either a sequence of integers or into a vector. To do this we have used the text tokenization utility class provided by Keras, i.e. **Tokenizer.**

This class allows to vectorize a text corpus, by turning each text into either a sequence of integers (each integer being the index of a token in a dictionary) or into a vector where the coefficient for each token could be binary, based on word count, based on tf-idf.

It has a parameter for **num_words**, which implies the maximum number of words to keep, based on word frequency. Only the most common num_words-1 words are kept.

In our case we have kept num_words = 6111 which is the maximum length of feature text in X. Also the max number of words in each ticket is capped at 200 because a lot of tickets have shorter descriptions and very few have verbose text.

6.3.2 Word embedding

"Word embeddings" are a family of natural language processing techniques aiming at mapping semantic meaning into a geometric space. This is done by associating a numeric vector to every word in a dictionary, such that the distance (e.g. L2 distance or more commonly cosine distance) between any two vectors would capture part of the semantic relationship between the two associated words. The geometric space formed by these vectors is called an embedding space.

For instance, "coconut" and "polar bear" are words that are semantically quite different, so a reasonable embedding space would represent them as vectors that would be very far apart. But "kitchen" and "dinner" are related words, so they should be embedded close to each other.

Ideally, in a good embeddings space, the "path" (a vector) to go from "kitchen" and "dinner" would capture precisely the semantic relationship between these two concepts. In this case the relationship is "where x occurs", so you would expect the vector kitchen - dinner (difference of the two embedding vectors, i.e. path to go from dinner to kitchen) to capture this "where x occurs" relationship. Basically, we should have the vectorial identity: dinner + (where x occurs) = kitchen (at least approximately). If that's indeed the case, then we can use such a relationship vector to answer questions. For instance, starting from a new vector, e.g. "work", and applying this relationship vector, we should get sometime meaningful, e.g. work + (where x occurs) = office, answering "where does work occur?".

GloVe word embeddings

We will be using GloVe embeddings, which you can read about here. GloVe stands for "Global Vectors for Word Representation". It's a somewhat popular embedding technique based on factorizing a matrix of word co-occurence statistics. Specifically, we will use the 200-dimensional GloVe embeddings of 400k words computed on a 2014 dump of English Wikipedia. Refer figure 6.1 for using GloVe embeddings in a model. We will discuss these steps next.

```
max_features = max
maxlen = 200
embedding_size = 200
tokenizer = tf.keras.preprocessing.text.Tokenizer(num_words=max_features)
tokenizer.fit_on_texts(data_df['MergedColumn'])
```

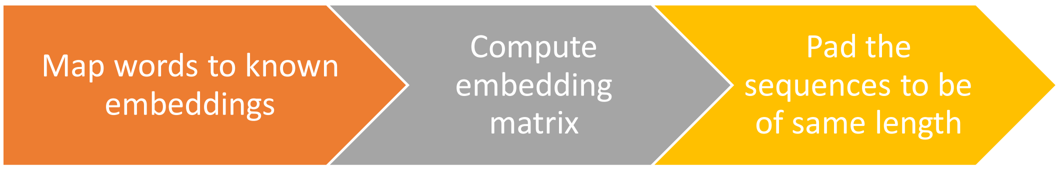

Map words to known embeddings
Compute embedding matrix
Pad the sequences to be of same length

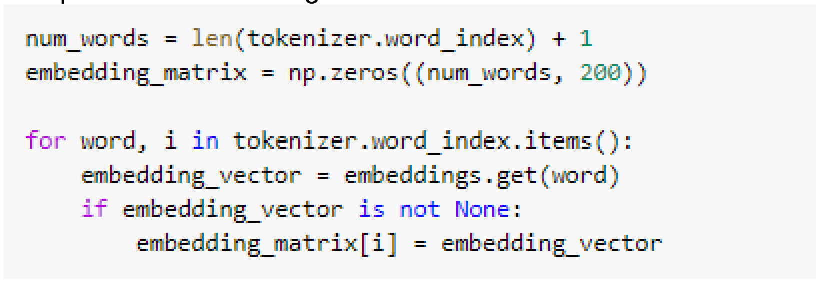
Figure 6.1: GloVe model building process

Preparing the Embedding layer

Next, we compute an index mapping words to known embeddings, by parsing the data dump of pre-trained embeddings:

```
EMBEDDING_FILE = './glove.6B.200d.txt'

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

At this point we can leverage our embedding_index dictionary and our word_index to compute our embedding matrix:



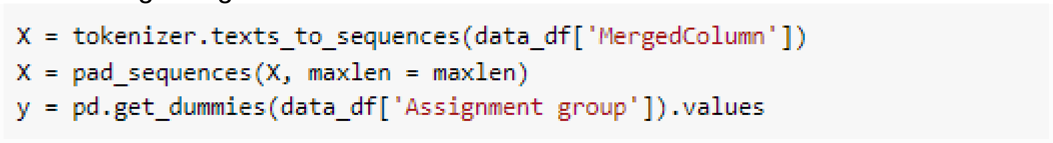
```
num_words = len(tokenizer.word_index) + 1
embedding_matrix = np.zeros((num_words, 200))

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

An Embedding layer should be fed sequences of integers, i.e. a 2D input of shape (samples, indices). These input sequences should be padded so that they all have the same length in a batch of input data (although an Embedding layer is capable of processing sequence of heterogenous length, if you don't pass an explicit input_length argument to the layer).

All that the Embedding layer does is to map the integer inputs to the vectors found at the corresponding index in the embedding matrix, i.e. the sequence [1, 2] would be converted to [embeddings[1], embeddings[2]]. This means that the output of the Embedding layer will be a 3D tensor of shape (samples, sequence_length, embedding_dim).

6.3.3 Padding

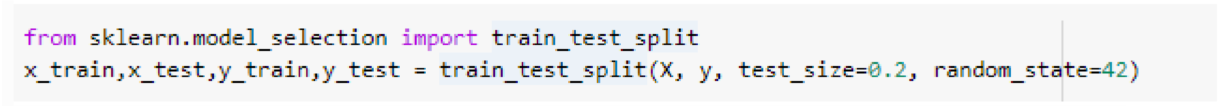
Truncate and pad the input sequences so that they are all in the same length for modeling.
Converting categorical labels to numbers.



```
X = tokenizer.texts_to_sequences(data_df['MergedColumn'])
X = pad_sequences(X, maxlen = maxlen)
y = pd.get_dummies(data_df['Assignment group']).values
```

6.3.4 Train test split

Truncate and pad the input sequences so that they are all in the same length for modeling As discussed in earlier sections, we divide our dataset into train,test and validation sets. Validation sets are created and specified for each epochs during model training stages. We have used 80-20 split for train and test sets.



```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

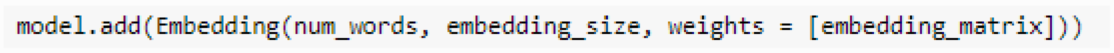
6.4 Architecture design

For this task we have built a LSTM network. The LSTM structure is one of the best-known base structures of the deep neural network theory that has already been introduced in this report in section 2.

In order to build a LSTM neural network from scratch we must fully define all the layers that make up the neural network as well as various aspects such as the initialization of the weights of all nodes, the activation function applied to each node and the implementation of various regularization techniques within the architecture.

The input layer.
The input layer is an embedding layer using GloVe embeddings.

We load this embedding matrix into an Embedding layer.



```
model.add(Embedding(num_words, embedding_size, weights = [embedding_matrix]))
```

.

The output layer.
The output layer depends on the approach of the problem and depends on the defined cost function. Since this is a multiclass classification problem, we have placed a single node at the end of the neural network with 74 output classes in which to use a **softmax** activation function. The softmax activation function is the activation function that works best with multiclass classification tasks in conjunction with **categorical cross-entropy loss**.

The hidden layers.
In order to determine the number of hidden layers and the number of nodes per layer, there is no guiding theoretical principle in the related literature that allows us to determine exactly what is the best possible composition and arrangement. It will depend a lot on the type of problem and the input data features.

In addition to defining the depth and width of the neural network, we must achieve with the neural network a correct generalization. As we have few input variables we could overfit the model very easily. For this reason, we have implemented the SpatialDropout1D regularization technique in each layer. SpatialDropout1D performs variational dropout in NLP models

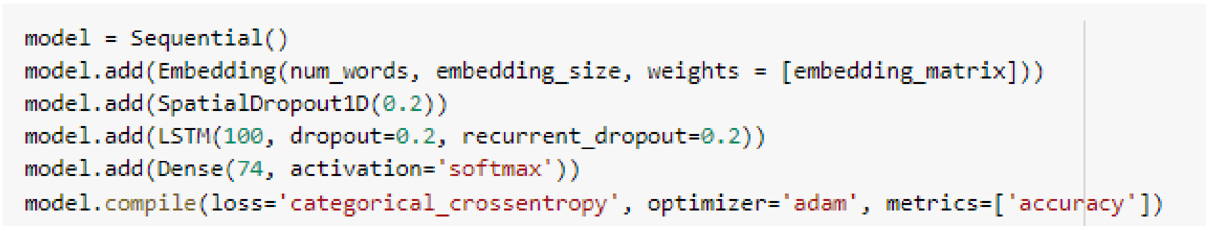
In short, the variables you have to configure when building such neural network are:

Select the optimal size (total number of nodes), depth (number of layers) and width (number of nodes per layer).

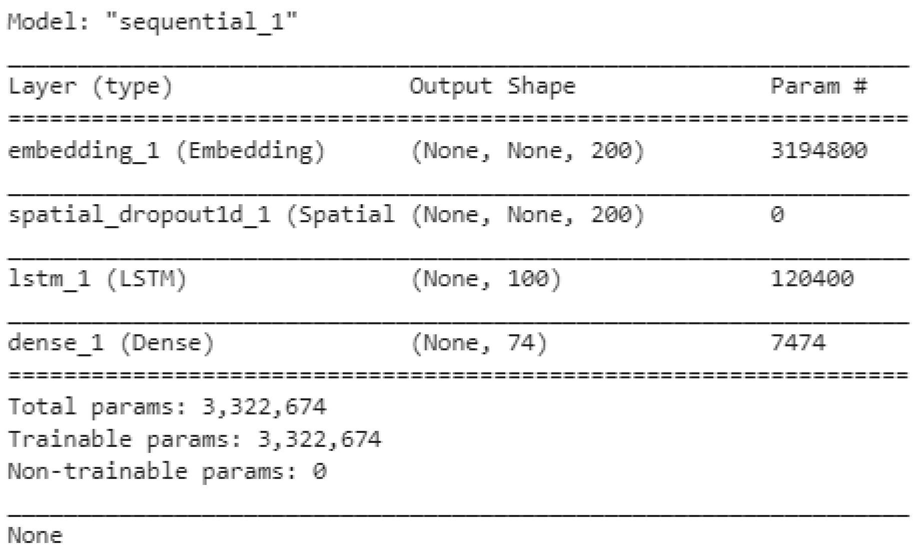
Select the dropped-out probability of the Dropout regulation technique.

Since there is no analytical formula for calculating model performance with each of the possible structures, we used a trial and error method calculating the performance of a finite number of possible combinations. We have used an LSTM with 100 memory units for this step.

Model summary –



```
model = Sequential()
model.add(Embedding(num_words, embedding_size, weights = [embedding_matrix]))
model.add(SpatialDropout1D(0.2))
model.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(74, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```



```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, None, 200)         3194800
_________________________________________________________________
spatial_dropout1d_1 (Spatial (None, None, 200)         0
_________________________________________________________________
lstm_1 (LSTM)                (None, 100)               120400
_________________________________________________________________
dense_1 (Dense)              (None, 74)                7474
=================================================================
Total params: 3,322,674
Trainable params: 3,322,674
Non-trainable params: 0
_________________________________________________________________
None
```

6.5 The training methodology

In the list of steps above, with each combination, the neural network is trained using the train and validation subset of the data.
In the optimization section, we have used the Adam optimizer and mini-batches of size 64.

We train the model for 30 epochs.

```
epochs = 30
batch_size = 64

history = model.fit(x_train, y_train, epochs=epochs, batch_size=batch_size,validation_split=0.1)
```

6.6 Model performance and results

We have used accuracy as the measure of performance. Below are the results –
On test set - **67.073%**
On training set – **93.73%**

The performance on test set and the plot of accuracy below suggests that the model has a large over fitting problem, more data may help, but more epochs will not help using the current data.

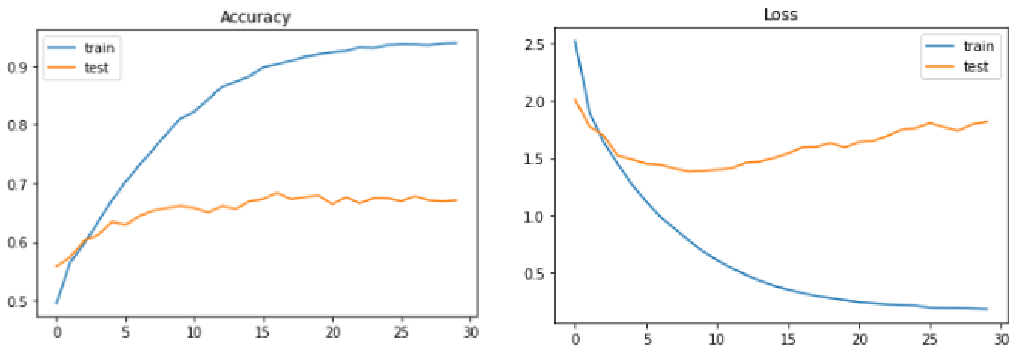


Figure 6.2: Accuracy and loss in train and test samples

6.7 Experiments & Results

Along with the LSTM model, we also used traditional ML models for this classification task. Below are the results. The final model results are in **bold**.

Sr.no	Model	Features used	Target classes	Training accuracy	Test accuracy	Comments
1	Logistic regression	Short description, Description, Caller	74	70.67	63.44	With train_test_split
2	Naïve Bayes	Short description, Description, Caller	74	51.93	53.49	With train_test_split
3	KNN (k=1)	Short description, Description, Caller	74	93.45	60.65	With train_test_split
4	KNN (k=3)	Short description, Description, Caller	74	93.99	62.58	With train_test_split
5	SVM	Short description, Description, Caller	74	81.15	55.78	With train_test_split
6	**LSTM + GloVe embeddings**	**Short description, Description, Caller**	**74**	**93.76**	**67.07**	**With train_test_split**
7	LSTM + GloVe embeddings	Short description, Description	34	92.41	65.78	With train_test_split
8	LSTM + GloVe embeddings	Short description, Description, Caller	2	98.63	81.22	With train_test_split
9	LSTM + GloVe embeddings	Short description, Description	2	99.08	79.37	With train_test_split
10	LSTM + GloVe embeddings	Short description, Description, Caller	2	98.20	81.60	With stratified_shuffle_split
11	LSTM + GloVe embeddings	Short description, Description, Caller	34	90.54	64.51	With stratified_shuffle_split

6.8 Comparison to benchmark

At the outset of the project during EDA phase, we found that the data was highly imbalanced. This is the biggest challenge in working with the dataset. Also there are a lot of classes which have less than 10 tickets in that group. This is very less data to train a model correctly.

When doing manual evaluation of data we found out that there is no clear boundary between how the tickets are distributed between groups. Hence it becomes very difficult to classify an incoming ticket correctly.

Multilingual dataset also poses its own problems as a single model may not be to predict correctly for different languages.

Given these many challenges with the dataset the performance we achieved is acceptable. We set ourselves a target initially to get more than 70% accuracy on test set. We achieved close to that using transfer learning approach.

7 Conclusions and future work

7.1 Conclusions

From the beginning of this project, our aim has been to develop an implementation that would allow us to accurately classify incoming tickets based on ticket description, providing a multiclass classification, of assignment groups.

Specifically, the implementation of this project has been designed around 2 main modules, in order to organize its development and achieve the corresponding objectives more effectively. These are: preprocessing module and classification module. This division is reflected in the structuring of this document in which we describe and evaluate each of these modules.

In the first place, using the preprocessing module we have extracted the information which is most important for classification and found common patterns from the data. And then, the classification module is proposed for classification of tickets. In the first step, 4 different traditional models are verified and analyzed and are not used further due to poor performance on the data set.

In the later steps we have introduced and implemented LSTM, whose structure has been entirely selected to maximize the quality of prediction in cases.

The main objective as well as the specific objectives have been achieved satisfactorily throughout the course of this master’s thesis. If we contextualize the results obtained we can appreciate the good performances achieved despite the difficulty of the challenge we have faced.

In addition, this project provides a valuable and extensive overview of the performance and limitations of the dataset used, in classification tasks, as well as in terms of the application of combination and regularization techniques. In any case, our top priority has been to maximize performance by implementing the state-of-art in LSTM networks, regularization and combination of predictions, with the certainty that these results can be an essential starting point for future related research, as detailed in the following section.

The project files are hosted on GitHub and can be found here.

7.2 Implications

This model will lower the Business cost invested for dispatching the incidents manually to the respective groups by a separate Team. As the tickets would be automatically dispatched to the respective groups (depending on the previous tickets dispatched), there would not be any need of a separate team for this task.
We recommend the Business to create a provision for the end user to raise their incident in a proper structured way. Ex:

- Options should be given to the end users for technical or non - technical issues.
- A dropdown to select in which application or functionality, the user is facing an issue.

This will significantly increase the accuracy of the model resulting in dispatching of tickets to the correct group most of the time.

7.3 Limitations

- The model supports only English language and no other languages are supported.
- The model performance is biased due to biased input data.

7.4 Closing Reflections

- EDA and preprocessing was the key in this project. Analyzing the data and finding useful insights helped us to architect the model correctly.
- LSTM and neural networks are powerful model architectures. If provided with sufficient datasets, a good performance is never difficult to achieve.
- The data is imbalanced, but a lot of data in real world is imbalanced as well. This project helped us to face the real world problem and how different strategies could be used to target such problems.

7.5 Future work

As mentioned in the previous section, this report covers the design and evaluation of a LSTM classification implementation, in the course of which a series of improvement aspects have been observed.

On the other hand, with regard to the dataset, we observe 2 evident aspects of improvement:

- To enhance the solution it is proposed to gather sufficient data for each class as classification on the basis of only one data point is difficult.
- There is no borderline between distributions of tickets to each group. A multi staged classifier, with the first stage as binary to classify between Group_0 and others and a second stage which classifies remaining 73 groups could be employed to achieve better performance.

On the other hand, we have used a variety of state-of-arts convolutional neural networks created for generic use, which we have adapted and trained for our project. It might be interesting to study the possibility of creating from scratch the Neural networks used in classification, optimizing them for our project and for this type of problems through NLP techniques.

Finally, given the implementation developed and the results obtained, an application could be created that integrates this implementation, providing it for use in professional IT support environments as a support tool to professionals.

Bibliography

[1] Francois Chollet, “Using pre trained word embeddings in a Keras model,” 2016. [Online]. Available: https://blog.keras.io/using-pre-trained-word-embeddings-in-a-keras-model.html

[2] Susan Li, “Multi-Class Text Classification with LSTM,” 2019. [Online]. Available: https://towardsdatascience.com/multi-class-text-classification-with-lstm-1590bee1bd17

[3] Logesh Umapathy, “A Handy pre-trained model for language Identification,” 2019, [Online]. Available: https://becominghuman.ai/a-handy-pre-trained-model-for-language-identification-cadd89db9db8

[4] Karthik Kandakunar,” IT Support Ticket Classification using Machine Learning and ServiceNow” 2019, [Online]. Available: https://medium.com/@karthikkumar_57917/it-support-ticket-classification-using-machine-learning-and-ml-model-deployment-ba694c01e416

[5] Keras documentation, [Online]. Available: https://keras.io/

[6] Kaggle notebooks, [Online]. Available: https://www.kaggle.com/

[7] Essentials of Deep Learning: Introduction to Long Short Term Memory, [Online]. Available: https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/

[8] GloVe Embeddings,[Online].Available: https://nlp.stanford.edu/projects/glove/