Technical University of Crete

Diploma Thesis

Detection of Disaster Events via Monitoring of Social Media

Author:
Konstantinos Chasapas

Committee:
Dr. Michail Lagoudakis
Dr. Georgios Chalkiadakis
Dr. Michael Zervakis

A thesis submitted in fulfillment of the requirements for the engineering diploma

in the

Intelligent Systems Laboratory
School of Electrical and Computer Engineering

June 21, 2019

Declaration of Authorship

I, Konstantinos Chasapas, declare that this thesis titled, "Detection of Disaster Events via Monitoring of Social Media" and the work presented in it is my own. I confirm that:

- This work was done mainly while in candidature for a undergraduate degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: Konstantinos Chasapas

Date: June 21, 2019

"A real artificial intelligence would be intelligent enough not to reveal that it was genuinely intelligent."

George Dyson

Abstract

In our days, the use of social media platforms has gained a lot of popularity. A large number of messages are posted everyday on media, such as Twitter. These messages could possibly be used for monitoring real-world events. However, twitter streams contain a large number of meaningless messages and useless content, which may have a negative effect on the monitoring and successful event detection performance. Monitoring and analyzing this rich and continuous user-generated content can yield unprecedentedly valuable information, enabling users and organizations to acquire actionable knowledge. This thesis proposes techniques for natural disaster detection and report by monitoring social media streams. The main task is to use Twitter as a sensor and investigate the data it provides in order to detect and locate a dangerous natural disaster. The detection of such events is attempted by the employment of methods from Natural Language Processing, Machine/Deep Learning and an innovative hybrid combination of the best approaches among the various choices we analyzed (for text representation: Bag of Words, Global Vectors; for text classification: Naive Bayes, Logistic Regression, Random Forest, Decision Tree, Bagged Decision Tree, AdaBoost, Deep Neural Network, Long-Short Term Memory). Finally, our work proposes a spatio-temporal investigation as part of the whole pipelined system to report detected catastrophic events. As a result, the proposed system has been applied with success to Twitter data from 2012 and 2015. The data were referring to two different past disaster events (Nepal earthquake, Hurricane Patricia) and one event not related to disaster (USA presidential election 2012). This system managed to detect both the place and the time of each disaster and confirmed that there is no disaster in the case of the USA elections.

Ανίχνευση Καταστροφικών Συμβάντων μέσω Παρακολούθησης Κοινωνικών Δικτύων

Κωνσταντίνος Χασαπάς

Στις μέρες μας, η χρήση των μέσων κοινωνικής δικτύωσης έχει κερδίσει πολλή δημοτικότητα. Ένας μεγάλος αριθμός μηνυμάτων δημοσιεύονται καθημερινά σε κοινωνικά δίκτυα, όπως το Twitter. Αυτά τα μηνύματα θα μπορούσαν ενδεχομένως να χρησιμοποιηθούν για την παρακολούθηση πραγματικών γεγονότων. Ωστόσο, οι ροές του Twitter περιέχουν μεγάλο αριθμό ανούσιων μηνυμάτων και άχρηστο περιεχόμενο, γεγονός που μπορεί να έχει αρνητικές επιπτώσεις στην απόδοση της παρακολούθησης και της επιτυχούς ανίχνευσης γεγονότων. Η παρακολούθηση και η ανάλυση αυτού του πλούσιου και συνεχούς περιεχομένου που παράγουν οι χρήστες μπορεί να αποδώσει πρωτοφανώς πολύτιμες πληροφορίες, επιτρέποντας σε χρήστες και οργανισμούς να αποκτήσουν γνώσεις που μπορούν να συνδεθούν με δράσεις. Η παρούσα διπλωματική εργασία προτείνει τεχνικές για την ανίχνευση και καταγραφή φυσικών καταστροφών παρακολουθώντας ροές κοινωνικών μέσων δικτύωσης. Ο κύριος στόχος είναι να χρησιμοποιηθεί το Twitter ως αισθητήρας και να διερευνηθούν τα δεδομένα που παρέχει για να ανιχνευθεί και να εντοπιστεί μια επικίνδυνη φυσική καταστροφή. Η ανίχνευση τέτοιων συμβάντων επιχειρείται με την εφαρμογή μεθόδων από την Επεξεργασία Φυσικής Γλώσσας, τη Μηχανική/Βαθιά Μάθηση και έναν καινοτόμο υβριδικό συνδυασμό των καλύτερων προσεγγίσεων μεταξύ των διαφόρων επιλογών που αναλύσαμε (για την αναπαράσταση κειμένων: Bag of Words, Global Vectors - για την ταξινόμηση κειμένων: Naive Bayes, Logistic Regression, Random Forest, Decision Tree, Bagged Decision Tree, AdaBoost, Deep Neural Network, Long-Short Term Memory). Τέλος, η εργασία μας προτείνει μια χωροχρονική διερεύνηση ως μέρος της διαδικασίας διαδοχικής επεξεργασίας στο σύστημα για την αναφορά καταστροφικών γεγονότων που εντοπίζονται. Ως αποτέλεσμα, το προτεινόμενο σύστημα εφαρμόστηκε με επιτυχία σε δεδομένα του Twitter από το 2012 και το 2015. Τα δεδομένα αφορούσαν δύο διαφορετικά γεγονότα καταστροφών του παρελθόντος (σεισμός του Νεπάλ, τυφώνας Patricia) καθώς και ένα μη καταστροφικό γεγονός (προεδρικές εκλογές της Αμερικής 2012). Το εν λόγω σύστημα κατάφερε να ανιχνεύσει τόσο τον τόπο, όσο και τον χρόνο κάθε καταστροφής, ενώ συγχρόνως επιβεβαίωσε την μη ύπαρξη καταστροφής στην περίπτωση των προεδρικών εκλογών.

Acknowledgements

First, and most of all, I would like to express my sincere gratitude to my supervisor Professor Dr. Michail Lagoudakis, School of Electrical and Computer Engineering, Technical University of Crete, for his continuous guidance, patience and support.

Contents

List of Figures

List of Tables

Abbreviations

ML	Machine Learning
DL	Deep Learning
RL	Reinforcement Learning
AI	Artificial Inteligence
NLP	Natural Language Processing
BoW	Bag of Words
NN	Neural Network
DNN	Deep Neural Network
ANN	Artificial Neural Network
RNN	Recurrent Neural Network
LSTM	Long Short Term Memory
FNN	Feedforward Neural Network
GloVe	Global Vectors
LR	Logisitc Regression
MLP	Multi Layer Perceptron
NB	Naive Bayes
LR	Logisitc Regression
BgC	Bagged Classifier
RF	Random Forest
DT	Decision Tree

I would like to dedicate this thesis to my beloved family…

Chapter 1

Introduction

Recently, huge attention has been drawn from researchers businesses and enterprises in data science. Every process, either human or mechanic produces data. If we scale this up to a large business we can securely infer that excessive amount of data is daily produced and stored. As the speed of information growth increases in geometrical progress, different kind of use of this information increases at the same rate.

Microblogging, as a form of social media, has fast emerged in recent years. In everyday life media platforms such as Twitter give us complementary information about the world. Owing to the large user base on Twitter, the platform provides real-time information about what happens in the world. Detecting events and harvesting references to them from Twitter is therefore a highly valuable goal. Events of public interest and insignificant events may both be confused by words in various ways of using Twitter, meaning that whole different sentences contain same words. Another misleading fact that could arise is the control of metaphors, in which a word or phrase literally denote one kind of object or idea used in place of another.

The aim of this research is to expand existing work on detecting significant disaster events on Twitter, by taking advantage of social media endless information and extract from everyday posts useful results about these events. Our goal is not only to classify Twitter post in pools of relativeness with natural disasters, that affect humanity, but also introduce a hybrid implementation for this case purpose and a different spatio-temporal reporting system.

This thesis provides a survey of techniques found in the literature and the experiments for event detection from Twitter posts. Disaster event detection from Twitter posts is a vibrant research area that draws on techniques from various fields such as machine learning, deep learning, natural language processing, information extraction and retrieval, spatio-temporal prediction. Finally, it submits a pipelined system with all required techniques for an event detection ready to used in any familiar case study.

1.1 Motivation

Social media such as Twitter has been part of everyday life for many people around the word. Huge amount of data both essential and not, are stored in media platforms databases. We could easily observe that very often human posts in social media encloses an event which happened, happening or is going to happen. On the other hand, data science seems to gain great impact in human life and more specifically in business. Using a subfield of artificial intelligence concerned with the interactions between computers and human (natural) languages apace with predictive force from ML, DL or RL has certain advantages and number of possible applications. One of them is the goal of this research. Specifically the connection between human language via every day's posts and probabilistic models. Moreover, the powerful outcome of this research as an application is the ability to gather, analyze and produce new information useful for humanity. To be more specific, an event recognition from Twitter posts, when an event is a natural disaster dangerous for people lives, could be used as notification for people in the affected area in really fast time. Thus, if an event is not dangerous then we could use the predicted information as a simple notification for an expected event and its popularity.

Earlier researches [34][27] from data scientists have only ML or DL algorithms to handle this type of application. Today, it is a great challenge to make the same, or particularly the same, researches with modern and more dominant tools. In this thesis DL alongside with ML algorithms, are mainly used but a hybrid approach, with the best performed algorithms from both lists, has also introduced. Knowledge aggregation from a pool of diverse classifiers is a major issue in this thesis.

1.2 Research Questions

Furthermore, an approach on how to evaluate the quality of translated technical documents will be proposed. Regarding this issue, we address some main research questions:

- How could the detection of an event be implemented, given raw data from Twitter database?
- In case of an event prediction, how one could monitor the exact place and time of that phenomenon.
- Can we detect such event occurrence in real-time by monitoring tweets?
- What NLP techniques do we need to perform in data in order to exclude useless information
- In the state-of-art of the NLP field, could word embedding prevail over Bag of Words model in this case scenario?
- Should we use ML or DL algorithms for text classification?

1.3 Related Work

The idea of event detection via monitoring social media posts is not new. Researchers have done this work with success around 2010 [34]. Same research had also been done with a lot of success, from HP labs and other researchers [46][45][7]. The areas related to this thesis can be split up in two main areas. Natural Language Processing, a sub-field of Computer Science that is focused on enabling computers to understand and process human languages. The second area and the most important one is the classification (text classification) part. Research on this topic has been performed mostly with ML algorithms and secondary with DL. Thus, this thesis focuses both on Machine Learning and Deep Learning (a method of statistical learning that extracts features or attributes from raw data) but also it makes introduction to a hybrid approach which lists algorithms from both categories. It also uses a different and more modern technique for text representation, called word embeddings in contrast with earlier research with stable and trusted BoW model.

1.4 Structure of Thesis

The following Chapter 2 will address the theoretical background concerning this work, focusing on the knowledge discovery process, Machine and Deep learning approaches, Natural Language Processing techniques and technical documentation. Then comes the Experimental Setup in Chapter 3, that lists and explains every data, algorithms alongside with models evaluation, comparison and result's discussion. This chapter introduces the hybrid approach we have used and a survey of techniques for the final pipeline of the system. The experiments that will be conducted in real data and a discussion of the results will be given in Chapter 4. Also an approach for geo-location prediction and time determination is introduced and performed in the same chapter. Chapter 5 answers the research question and discusses limitations and future work.

Chapter 2

Literature Review

2.1 Machine Learning

Within the field of machine learning, there are two main types of tasks: supervised, and unsupervised. Supervised learning is when you have input variables (x) and an output variable (Y). The goal is to map the input variables to the output variable with a mathematical function. So, whenever you have input data (x), the learned function (f) can predict output data (Y) as accurately as possible [11].

$$Y = f(X) \tag{Function}$$

Unsupervised learning, on the other hand, is when you only have input data (x) and no corresponding output variables. That means that the system depends on its own actions and observations without "correct" answer. But the question here is: how it is supposed to learn with no output value. This type of model uses its previous observations to learn and proceed with predictions.

Supervised learning problems can be further grouped into regression and classification problems.
Classification is the problem of predicting a discrete class label output for example. A classification problem requires that examples be classified into one of two (binary classification) or more classes (multi-class or multi-label classification) with discrete or real-valued input variables.
Regression is the problem of predicting a continuous quantity output for example. This type of problems can have discrete or real-valued input variables. A problem

with multiple input variables is often called a multivariate regression problem. This thesis studies a binary classification problem and uses only supervised learning methods. So we use a variety of different algorithmic techniques to implement our classifiers for the purpose of prediction.

After the usual feature engineering and model implementation another essential part is the evaluation of the ML algorithm. This step is about how effective is the implemented model based on some specific metrics. The four basic classification performance metrics are: accuracy, F1, precision and recall. It is very important for someone to know which performance metric to use and why, as it's one of them demonstrates the same result from a different view. To make a brief analysis of metrics we will use the confusion matrix. The confusion matrix is not a performance measure, but an intuitive way to describe the performance of a classification model. It is a two dimensional table with rows as "predicted" and columns as "actual" values from which almost all of the performance metrics can be computed [44].

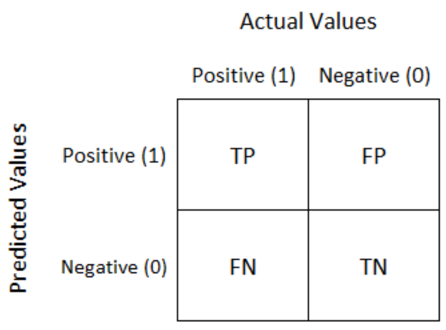


FIGURE 2.1: Confusion Matrix

Terms associated with the Confusion matrix:
1. True Positives (TP) are the cases where the actual class of the data point was 1(True) and the predicted is also 1 (True).
2. True Negatives (TN) are the cases where the actual class of the data point was 0(False) and the predicted is also 0 (False).
3. False Positives (FP) are the cases where the actual class of the data point was 0(False) and the predicted is 1 (True). False is because the model has predicted incorrectly and positive because the class predicted was a positive one (1).
4. False Negatives (FN) are the cases where the actual class of the data point was 1 (True) and the predicted is 0 (False). False is because the model has predicted incorrectly and negative because the class predicted was a negative one (0).

For every classification problem the goal is to minimize false negatives and/or false positives to make predictions more accurate. To define evaluation performance metrics using the above counts, we need to define mathematical functions for every single metric:

1. Accuracy is the number of correct predictions over all predictions:

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{2.1}$$

2. Precision in the number of true positives (TP) out of our all positive predictions:

$$Precision = \frac{TP}{TP + FP} \tag{2.2}$$

3. Recall denotes the positive predictions (TP) out of all actual positives:

$$Recall = \frac{TP}{TP + FN} \tag{2.3}$$

4. F1 score combines both accuracy and recall. It isn't always efficient, but in specific problems could make a better comparison between two classification algorithms. Actually it is the arithmetic mean of precision (P) and recall (R):

$$F1 = \frac{P + R}{2} \tag{2.4}$$

In this thesis, most of the times, algorithms have been compared using the accuracy score and occasionally with the F1 score for academic purposes only.

The next step in ML is about methods of data splitting. To evaluate the model's performance of an ML learning algorithm, it is necessary to have testing data. The most fundamental method is to split our verified data in what is usually called "Test" data and "Train" data. It is easy to understand its purpose of use as test data will be used to evaluate our model. Most of the time, our data are rather limited, which concludes to better prediction on training data and large error rate on test, so it is sensible to use the cross validation method. This procedure has to do with the skill of a ML model to deal with unseen data. Unlike the default test/train split of data, cross validation is a statistical technique which involves partitioning the data into groups, training the data on a subset and use the other subset to evaluate the model's performance, Figure 2.2 [11][14].

A common word among data scientists is model over-fitting. Over-fitting happens

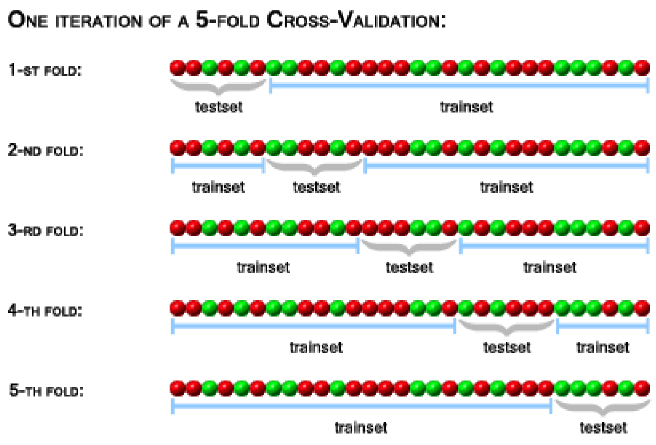


Figure 2.2: Cross Validation ("ProClassify User's Guide", August 2006)

when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data. That can be often detected by big divergences among test and train accuracy. Few or too many examples could cause this problem to a classification model as well as not balanced labels. The process of cross validation and other techniques can help algorithms prevent such problems by using all of the data both for training and testing its performance.

When creating a machine learning model, you'll be presented with design choices as to how to define model architecture. Often, we don't know from the beginning which architecture is optimal for our case. Subsequently, we need to perform changes to our ML model, in order to find the ideal one, a procedure usually called Hyperparameter tuning. Grid search and/or Random search are the most basic hyperparameter tuning methods. With this type of techniques, we build a model for each possible combination of all of the hyperparameter values provided, evaluating each model, and selecting the architecture with the best results [35][21]. Finally, both cross validation and hyperparameter tuning are essential techniques used in this thesis, so their theoretical approach is deemed necessary. Below we will approach theoretically a specific number of ML algorithms, which have been used during this thesis.

2.1.1 Logistic Regression

Logistic Regression is fundamental and widely known as ML classification algorithm. It is used in many social science applications, when the dependent variable is categorical. Logistic Regression uses logistic function (also called sigmoid function), Equation 2.5, in connection with the fundamental equation of generalized linear model, Equation 2.6. The result of these two called logistic regression equations are presented in Equation 2.8 [12].

$$y = b_0 + b_1 \times x \tag{2.5}$$

$$p = \frac{1}{1 + e^{-y}} \tag{2.6}$$

We can write the equation in terms of log-odds, which is a linear function of the predictors, Equation 2.7. The coefficient (b_1) is the amount the log-odds changes with a one unit change in x.

$$\text{log-odds} = ln\left(\frac{p}{1-p}\right) = ln\left(\frac{\text{probability of presence of charecteristic}}{\text{probability of absence of charecteristic}}\right) \tag{2.7}$$

$$ln\left(\frac{p}{1-p}\right) = b_0 + b_1 \times x \tag{2.8}$$

Logistic regression is similar to a linear regression but can handle with success classification problems too. For classification, we prefer probabilities between 0 and 1, so we wrap the right side of the equation into the logistic function. The curve is constructed using the log-odds of the target variable, rather than the probability as we can observe in Figure 2.3 [39].

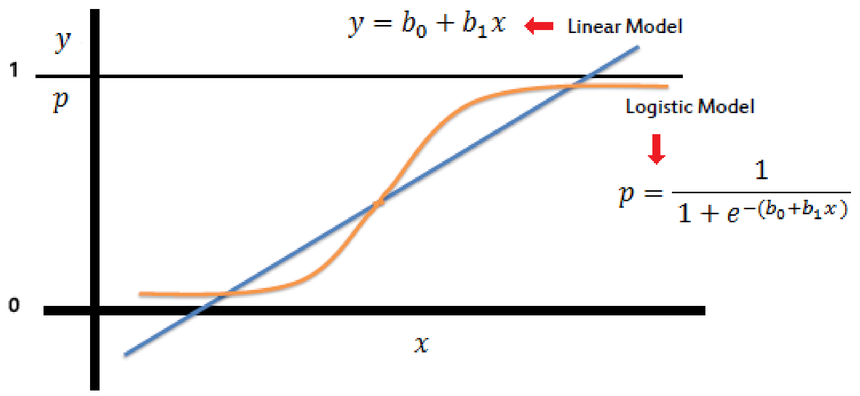


FIGURE 2.3: Logistic Regression Model ("All about Logistic regression in one article", October 2018)

The goal of logistic regression is to find the best fitting model to describe the connection between the dichotomous characteristic of interest (dependent variable) and a set of independent (predictor) variable.

2.1.2 Ensemble Learning

Ensemble learning is a well known technique to combine multiple classification models into one ensemble classifier. Ensemble methods usually produce more accurate solutions than a single model would. Figure 2.4 shows a simple ensemble learning architecture. In contrast to ordinary learning approaches, ensemble methods try to train a set of learners and then combine the separate predictions to solve the problem. There are multiple techniques to combine the variety of model's predictions so as to find out the final and complete prediction [28]. The most known and simple techniques are:

- Majority Voting: The predictions by each model are considered as a "vote". The predictions which we get from the majority of the models are used as the final prediction.
- Simple Averaging: Simple averaging obtains the combined output by averaging the outputs of individual learners directly.
- Weighted Averaging: Weighted averaging obtains the combined output by averaging the outputs of individual learners with different weights implying different importance [41].

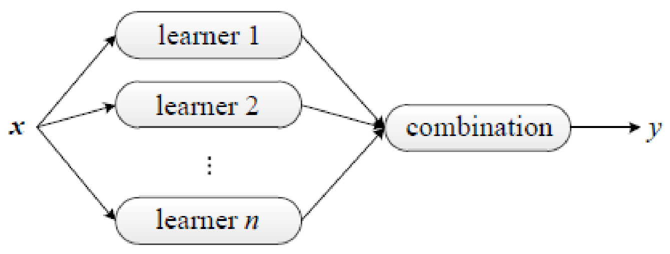


FIGURE 2.4: A common ensemble architecture (Zhou, 2006,"Ensemble Methods: Foundations and Algorithms", New York, NY: Taylor and Francis Group)

The main idea behind ensemble models is the effort to achieve better accuracy, reduce bias and variance errors as well as avoid model over-fitting. The most popular ensemble methods used in ML fields are:

- Bagging (Bootstrap Aggregating): In this type of method multiple models (weak models) of same learning algorithm trained with subsets of data are randomly picked from training data as we can see in Figure 2.5 [17]. The next step has to do with type of combination mentioned above.

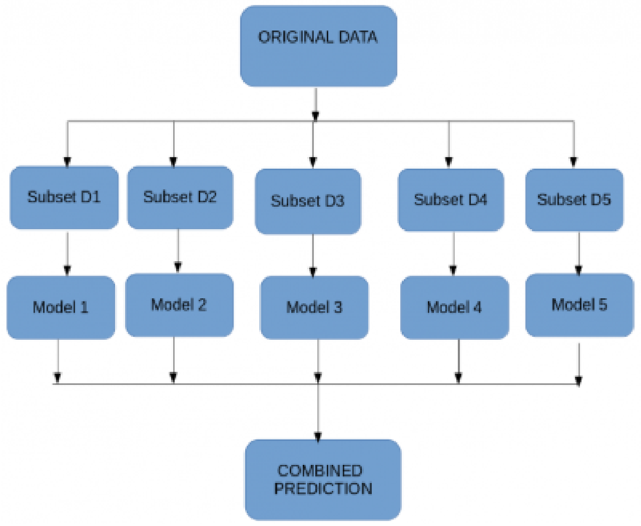


FIGURE 2.5: Bagging scheme("A Comprehensive Guide to Ensemble Learning", June, 2018)

- Boosting: Unlike Bagging, Boosting is a sequential procedure where weak learners turn to strong. In this process each subsequent model attempts to correct the errors of the previous model. So every model's success depends on its precursor. Boosting incrementally builds an ensemble by training each model with the same data but where the weights of instances are adjusted according to the error of the last prediction. The main idea is forcing the models to focus on the instances which are hard to be learned. A simple schema of boosting procedure can be found in Figure 2.6.

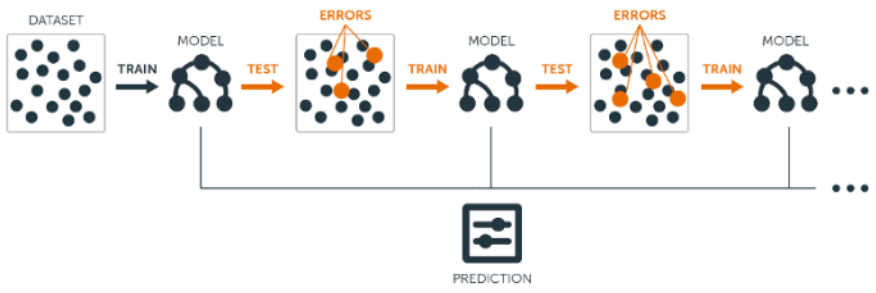


FIGURE 2.6: Bagging scheme ("Introduction to Boosted Tree", March, 2017)

There is a certain number of algorithms based on these two methods of ensemble learning. In Section 2.1.2.1 and 2.1.2.2 we will take a quick survey about two of them, which have been used in this thesis.

2.1.2.1 Adaboost

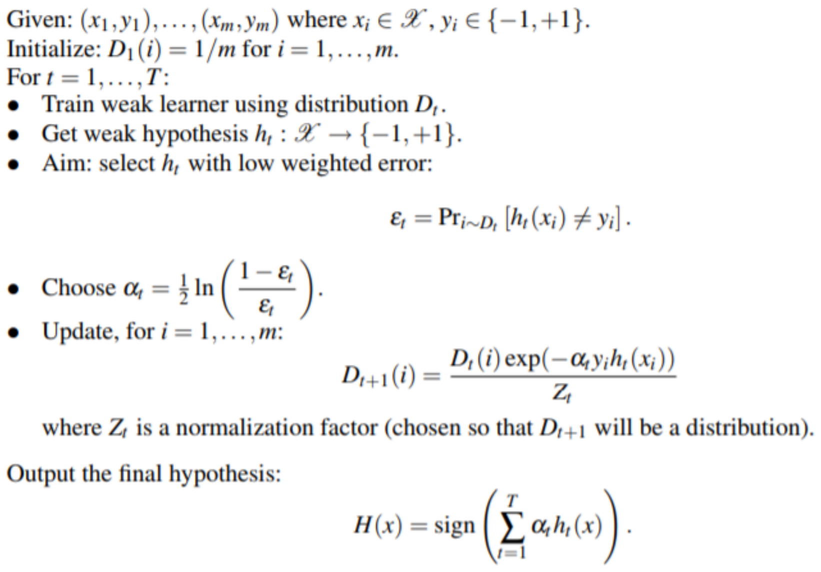
Adaptive Boosting or Adaboost was the first successful boosting algorithm developed for binary classification by Yoav Freund and Robert Schapire. AdaBoost can be used to boost the performance of any machine learning algorithm, but it is often used in conjunction with decision trees.

Adaboost as Boosting algorithm has the ability to improve faulty results of weak learners, with the hypothesis that the weak learner accepts weights in training data. The main contribution of this algorithm, is the choice of training a set of data for each new classifier based on the results of its precursor. Then adaboost determines the exact weight that should be applied to each base learner prediction, instead of a simple training of weak algorithms and averaging the results [26]. The drawback of AdaBoost is that it is easily defeated by noisy data.

“Pseudocode for AdaBoost is shown in Figure 2.7. Here we are given m labeled training examples $(x_1, y_1), \cdots, (x_m, y_m)$ where the x_i's are in some domain X, and the labels $y_i \in$ (-1, +1). On each round $t = 1, \cdots, T$, a distribution D_t is computed as in the figure over the m training examples, and a given weak learner or weak learning algorithm is applied to find a weak hypothesis $h_t : X \longrightarrow$ (-1, +1), where the aim of the weak learner is to find a weak hypothesis with low weighted error ϵ_t relative to D_t. The final or combined hypothesis H computes the sign of a weighted combination of weak hypotheses.

$$F(x) = \sum_{t=1}^{T} \alpha_t \times h_t(x) \tag{2.9}$$

This is equivalent to saying that H is computed as a weighted majority vote of the weak hypotheses h_t where each is assigned as weight α_t . In this chapter, we use the terms “hypothesis” and“classifier” interchangeably” [37].



Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in \mathcal{X}$, $y_i \in \{-1, +1\}$.
Initialize: $D_1(i) = 1/m$ for $i = 1, \ldots, m$.
For $t = 1, \ldots, T$:

- Train weak learner using distribution D_t.
- Get weak hypothesis $h_t : \mathcal{X} \to \{-1, +1\}$.
- Aim: select h_t with low weighted error:

$$\varepsilon_t = \Pr_{i \sim D_t}\left[h_t(x_i) \neq y_i\right].$$

- Choose $\alpha_t = \frac{1}{2} \ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$.
- Update, for $i = 1, \ldots, m$:

$$D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

Output the final hypothesis:

$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right).$$

FIGURE 2.7: The boosting algorithm AdaBoost(“Robert E. Schapire ”,n.d)

2.1.2.2 Bagged Decision Tree

Decision tree is one of the most commonly used ML algorithm for classification purposes. A decision tree is a tree where each node represents a feature of the original data. It’s link with the successor node (feature) represents a decision and finally each leaf of the tree corresponds to a possible outcome. A decision tree often starts with a single node,branches to a number of decision nodes and ends to a bunch of results as we you can see in Figure 2.8 .

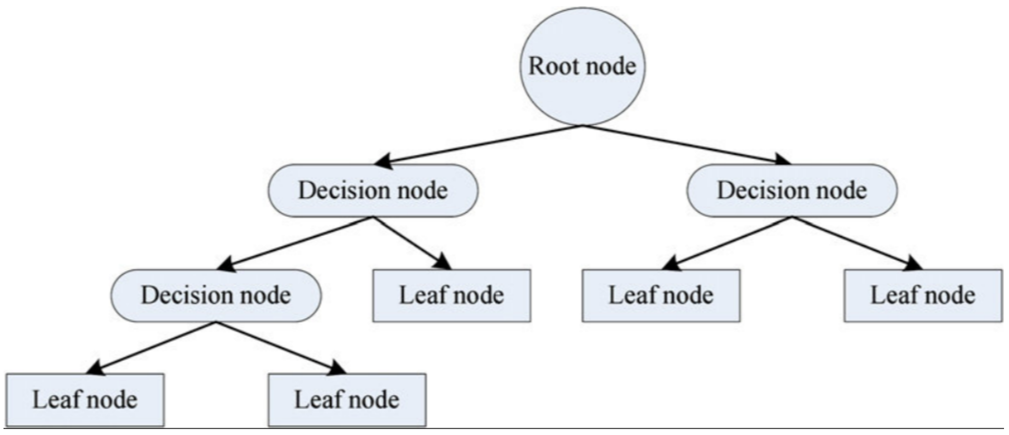


FIGURE 2.8: Concept of decision tree model(“Decision Tree–Based Deterioration Model for Buried Wastewater Pipelines”, October, 2013)

Bagging is a meta-algorithm usually used with decision trees to reduce models variance and improve stability of the weak algorithm. Ensemble bagging helps to improve bias-variance trade-off, given that every ML algorithm tries to achieve both low bias and variance (see Appendix A).

Suppose $C(S, x)$ is a decision tree classifier, based on the training data S, producing a predicted class label at input point x. To bag C we draw bootstrap samples $S^{*1}, \cdots S^{*\mathrm{B}}$ each of size N with replacement from the training data [20].

$$C_{\mathrm{bag}}(x) = MajorityVote\left\{C(S^{*\mathrm{b}}, x)\right\}_{\mathrm{b}=1}^{B} \tag{2.10}$$

2.1.3 Naïve Bayes

"Learning a Naïve Bayes classifier is just a matter of counting how many times each attribute co-occurs with each class."

-Pedro Domingos

A Naïve Bayes classifier is a probabilistic machine learning model that's used for classification problems.This algorithm is based on a mathematical intuition called Bayes Theorem. As presented in Equation 2.11

$$P(A \mid B) = \frac{P(B \mid A)\, P(A)}{P(B)} \tag{2.11}$$

$P(A \mid B)$ is the Posterior Probability, $P(B \mid A)$ is the Likelihood, $P(A)$ is the Prior Probability and $P(B)$ is the Marginal Likelihood. Using Bayes theorem, we can find the probability of A happening, given that B has occurred with the assumption that A, B are independent variables (that's the reason for Naïve). This assumption is called class conditional independence.

The NB classifier is the conditional model $P(B_1, B_2, \cdots, B_n \mid A)$ with the Naïve assumption that class A is independent. In simple terms, a Naïve Bayes classifier assumes that the presence of a particular feature of a class is unrelated to the presence of any other feature. That means that if we want to compute the probability of B_n samples given class A NB classifier assumes that are independent and work as follows $P(B_1, B_2, \cdots, B_n \mid A) = P(B_1 \mid A)P(B_2 \mid A) \cdots P(B_n)$. Using Bayes' theorem, as explained, using Marginal Likelihood in case of large n we can reform the above equation as follows

$$P(A \mid B_i) = P(A \mid B_1, B_2, \cdots, B_n) = \frac{P(B_1, B_2, \cdots, B_n \mid A) \times P(A)}{P(B_1, B_2, \cdots, B_n)} \tag{2.12}$$

$$P(A \mid B_1, B_2, \cdots, B_n) = \frac{P(B_1 \mid A)P(B_2 \mid A) \cdots P(B_n \mid A) \times P(A)}{P(B_1, B_2, \cdots, B_n)} \tag{2.13}$$

The Naïve Bayes classifier combines this model with a decision rule, that the hypothesis that is most probable commonly known as the maximum a posteriori or MAP decision rule. The corresponding classifier is the function defined as follows [49]:

$$\text{classify}(A) = \arg\max_{c} p(A) \prod_{i=1}^{n} p(B_i \mid A) \tag{2.14}$$

Finally it is important to understand the fact that NB classifier achieves better accuracy with independent variables and of equal importance as feed.

2.2 Deep Learning

Deep Learning or Deep Neural Learning or Deep Neural Network, is sub-field within Machine Learning, which has gained a lot of attraction in the last decades. The term Deep Learning was introduced to the machine learning community by Rina Dechter in 1986 and to artificial neural networks by Igor Aizenberg in 2000. Due to luck of both computational power and big data DL hadn't been used until 2006.

Neural network origin is from biology field as NN imitates the human brain workings, whose neuron cells are connected together by synapses. This artificial approach gave birth to a simple Artificial Neural Network by Widrow and Hoff. Below in Figure 2.9 is an example of a simple ANN. In practice an ANN is defined by its single hidden layer in contrast with DNN. More specific analysis of ANN in 2.2.1

Deep learning, despite of ML's linear algorithms, utilizes a hierarchical level of artificial neural networks of increasing complexity and abstraction. With deep

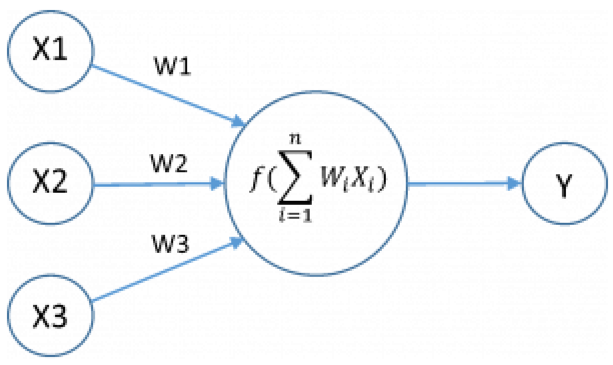


FIGURE 2.9: A single neuron

networks we can perform feature extraction and classification in one shot, which means we only have to design one model. The concept of DNNs demands more than one hidden layer from which features is extracted and feeding next layer. In the same way as we human brain learns from experience, DL algorithms perform same tasks repeatedly. In order to approximate the correct outcome DL algorithms make small changes on their learning process in each unique repetition [19][10].

2.2.1 Artificial Neural Network

An Artificial Neural Network is a mathematical model that tries to simulate the structure and functionalities of biological neural networks. The basic unit of every artificial neural network is an neuron, that is, a simple mathematical function. Moreover connections between two nodes (neurons), called synaptic weight, refers to the amplitude between those nodes. The relation between an input set x_i, output y_j and the synaptic weight can be represented mathematically in Equation 2.15.

$$y_j = \sum_i w_{\text{ij}} \times x_i + b \tag{2.15}$$

That means that every single input is multiplied by a different weight (synaptic weight from it's synapses). Next step has to do with summation in every artificial neuron, a function that sums all weighted inputs and bias (b). Outputs from every neuron in the network are passing through activation function, which is the last step of the procedure before the output. The whole process could be visualized in Figure 2.10.

Activation function is really important for a ANN so as to learn and make sense of something really complicated and non-linear and complex functional mappings

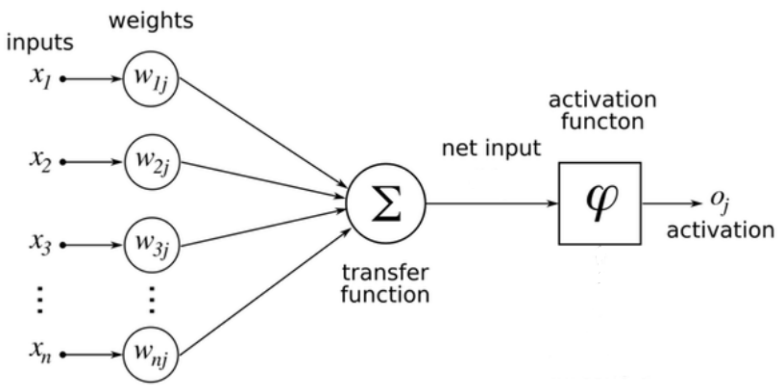


FIGURE 2.10: Artificial Neural Network, ("Artificial Intelligence and Neural Networks, Explained", Alex Castrounis, 2016)

between the inputs and output/s. It's main purpose is to map the results into a desired range, for example 0 to 1, in consideration of course with the type of the problem. There are many different activation functions. The most frequently used Activation functions, Figure 2.11, with their mathematical expression are:

- Sigmoid: $\sigma(z) = \frac{1}{1+e^{-z}}$
- ReLU: $R(z) = max(0, z)$
- Tanh: $tanh(x) = \frac{2}{1+e^{-2z}} - 1$

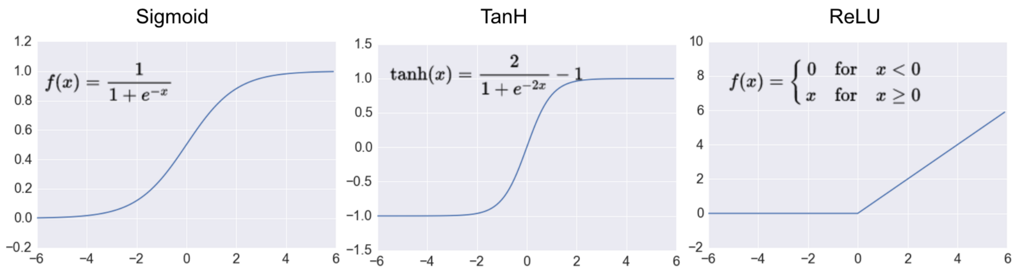


FIGURE 2.11: Basic Activation Functions, (What is meant by activation function?, Nawin Sharma, 2018)

Another important information on how ANN learns, is an algorithm deployed on the architecture, called Backpropagation. In practice the output values in comparison with the desired outputs have always a deviation. This amount of error is backpropagated through the ANN and the weights are adjusted accordingly. To conclude, when dealing with different kind of tasks, it is very efficient to know and hundle the ANN's topology or architecture. There are several kind of architectures all of them corresponding to the way that individual neurons are connected with others. We could discern to basic classes:

- Feed Forward: where information flows from inputs to outputs.
- Recurrent: where some of the information flows not only in one side (input to output as in FFN topology) but also from output to input.

Both topologies are represented in Figure 2.12

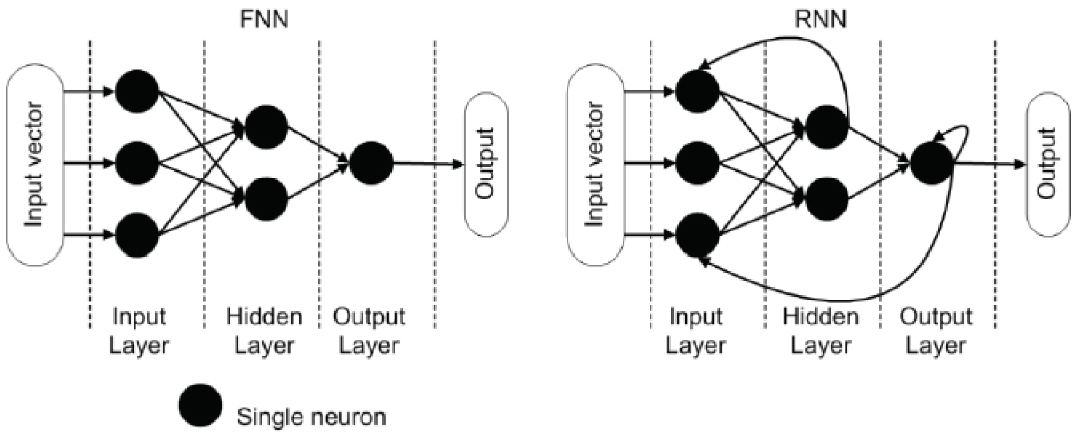


FIGURE 2.12: Feed-forward (FNN) and Recurrent (RNN) architecture, ("Introduction to the Artificial Neural Networks", Andrej Krenker, 2011)

2.2.2 Recurrent Neural Network

Recurrent Neural Network (RNN) is a robust type of neural network with recurrent architecture. Consequently information can transmitted from input to output and the opposite. This allows RNN to exhibit dynamic temporal behavior for a time sequence. RNNs, unlike feed-forward neural networks, can use their internal memory to process sequences of inputs. This is important because the sequence of data contains crucial information about the next input set. To understand how a Recurrent NN works, we could thought of as multiple copies of the same network, each one feed next layer. As presented in Figure 2.13, X_0 is the input for the first layer, then the output h_0 alongside with next input X_1 are feeding the next layer (green A box) and so goes on.

To summarize Recurrent NN are widely known and useful for purposes, such as: Speech Recognition, Machine Translation, Chatbots and many other text processing tasks. There are plenty types of RNN's with different characteristics for different cases. A bunch of them are: LSTM, Bi-directional, Hopfield [8][22].

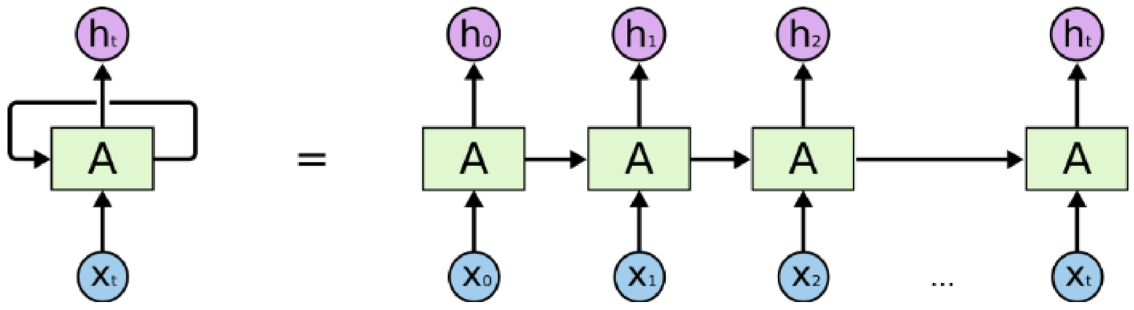


FIGURE 2.13: An unrolled recurrent neural network, ("Understanding LSTM Networks", Christopher Olah, 2015)

2.2.2.1 Long short-term memory

Long Short-Term Memory, usually just called LSTM networks are an extension of RNN's, which basically extends their memory. Having many loops of information in RNN's usually leads to very large updates in model's weighting system. This behavior tends to cause many problems, so it was essential to approach memory, of these NN's, in a different way. LSTM's are capable of learning long-term dependencies, as well as forgetting not necessary information based on the data. The long short-term memory block is a complex unit with various components (gates), Figure 2.14. This unit is called a long short-term memory block because the program is using a structure founded on short-term memory processes to create longer-term memory. Those gates, such as as weighted inputs, activation functions, inputs from previous blocks and eventual outputs act on the received signal and they block or let information pass in order to apply changes on their weights. These blocks make decisions on what to store, and when to allow reading, writing and erasing, via gates that open and close.

The horizontal line running through the top of the diagram, is called cell state, and is connected directly to the next unit, with some gates interacting with it. The first gate on the block called forget gate is responsible for removing information from the cell state, Figure 2.15. It is controlled by a single basic neural network layer with sigmoid as activation function, multiplied with the memory of the previous block. On each block we have three of these gates.

Next step, called memory gate has to do with what information from previous block we'll store in the cell state. In practice it's just a memory update (from $C_{\text{t-1}}$ to C_t). It is composed from two parts, a sigmoid NN multiplied with tanh layer. The inputs for these two layer are both the previous output and the current input.

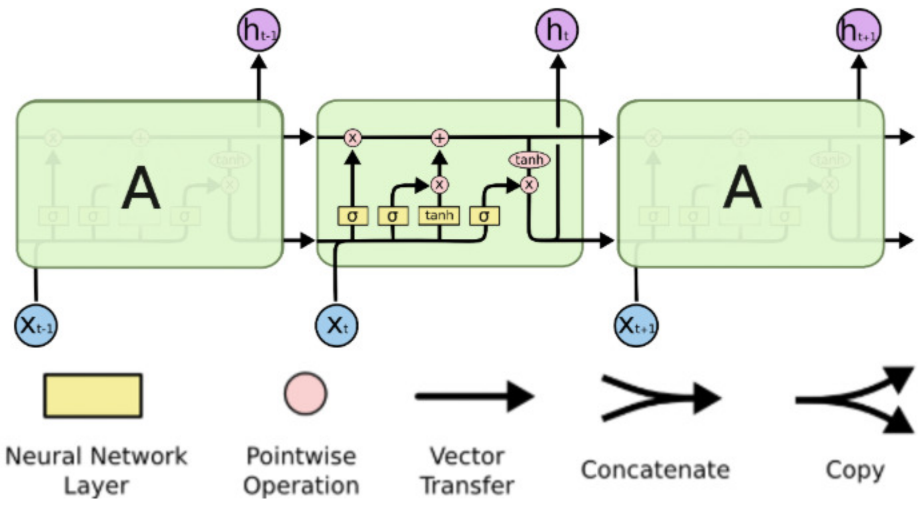


FIGURE 2.14: Three LSTM blocks, (“Understanding LSTM Networks”, Christopher Olah, 2015)

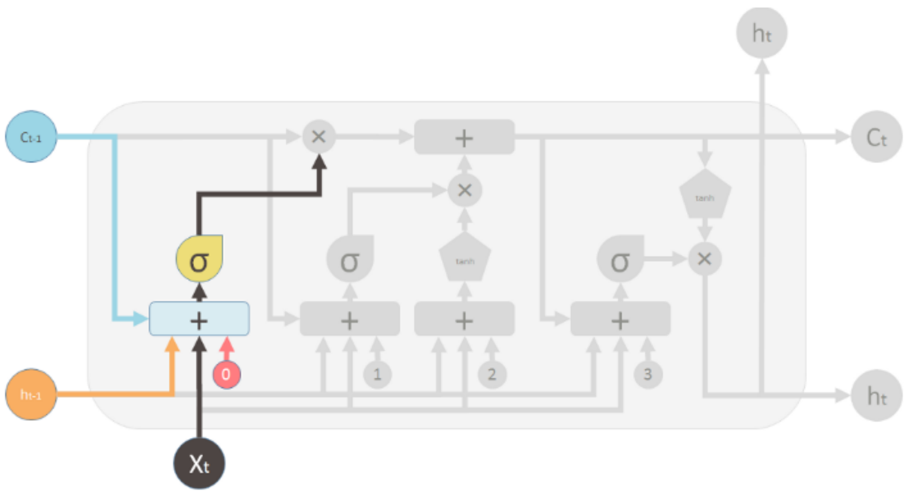


FIGURE 2.15: Forget Gate, (“Understanding LSTM Networks”, Christopher Olah, 2015)

Multiplier output is responsible for how much the new memory should influence the old one, Figure 2.16.

Finally, we need to generate the output, Figure 2.17. The output will be the cell state after tanh activation function (values between $(-1, 1)$) filtered (multiplied) with a sigmoid NN with current input, previous output and current memory as inputs [29][43][50].

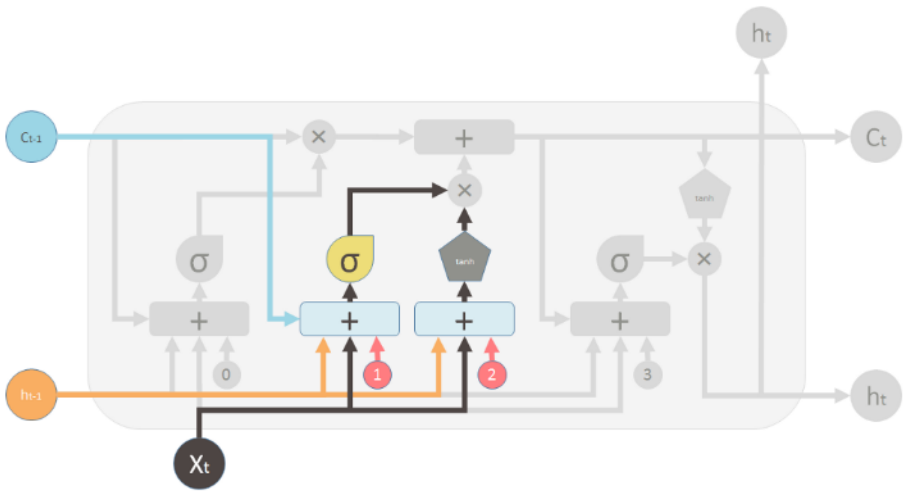


FIGURE 2.16: Memory Gate, ("Understanding LSTM Networks", Christopher Olah, 2015)

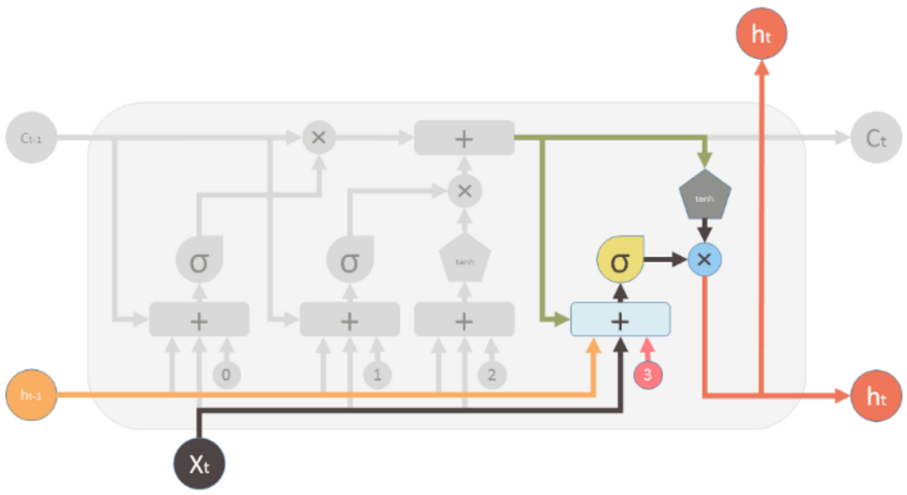


FIGURE 2.17: Output Gate, ("Understanding LSTM Networks", Christopher Olah, 2015)

2.3 Natural Language Processing

Natural Language Processing (NLP) is a sub-field of Data Science and Artificial Intelligence concerned with enabling computers to understand, interpret and manipulate human languages. The eventual objective of NLP is to read, understand and comprehend human languages in a manner that is valuable. Both ML and DL are used in NLP techniques with a lot of success. Commonly, feeding statistics and models have been the method of choice for interpreting phrases. Recent advances and traditional applications, in this area include voice recognition, machine translation, spam detection, named entity recognition, question answering (chatbots), sentiment analysis and text classification. In this thesis NLP will be used both for named entity recognition and text classification processes.

NLP is applying algorithms to identify and extract the natural language rules such that the unstructured language data is converted into a form that computers can understand. When the text has been provided, the computer will utilize algorithms to extract meaning associated with every sentence and collect the essential data from them. Sometimes, the computer may fail to understand the meaning of a sentence well enough, leading to obscure results.

Natural language processing uses a bunch of different techniques in order to apply grammatical rules to words and extract useful information from text. Syntactic Analysis (Syntax) and Semantic Analysis (Semantic) are the two main techniques that lead to the understanding of natural language.

- Syntax refers to the arrangement of words in a sentence so that they make grammatical sense.
- Semantic analysis refers to a formal analysis of meaning for a sentence

Another essential part of NLP is handling raw data before using them. Informal type of speech can often lead to insufficient understanding by computers. This step called Pre-processing or cleansing of data as a process of removing data "noise".

2.3.1 Data Pre-processing

A big part of NLP is preprocessing of text data. There is a certain number of techniques could be used in this section in consideration of the problem. The most basic of them are:

- Tokenization is a step which splits longer strings of text into smaller pieces, or tokens. A token is a string of contiguous characters between two spaces, or between a space and punctuation marks.
- Normalization generally refers to a series of related tasks meant to put all text on a level playing field

  - Stemming algorithms work by cutting off the end or the beginning of the word, taking into account a list of common prefixes and suffixes that can be found in an inflected word
  - Lemmatisation is the algorithmic process of determining the lemma of a word based on its intended meaning. Unlike stemming, lemmatisation depends on correctly identifying the intended part of speech and meaning of a word in a sentence

- Noise Removal: This step has to do with the removal of non useful information from data and/or extractions of valuable data from other formats, such as JSON. Other modifications of this step could be: Manipulation of regular expressions, remove of markup and metadata and remove of text file headers, footers.
- Other important steps: Except the above techniques there are several other essential steps for the overall process, a sample of them are: Removing special characters, removing stopwords, expanding contractions, remove punctuation, remove numbers, set all characters to lowercase, remove other languages (if needed) [51].

2.3.2 Text Classification

Text classification is a supervised ML method in the form of NLP used to classify sentences or text documents into one or more predefined classes or categories. It's a widely used kind of classification, playing an important role in tasks, such as: spam filtering, sentiment analysis, categorization of news articles and event detection. Monitoring events via text classification is an crucial approach this thesis has based on.

Although, ML and DL algorithms cannot work with raw text directly, it is very important to convert the text into vectors of numbers. This procedure is called feature engineering(feature extraction) and maps raw data to feature vectors, a form that can easily be used from algorithms.

2.3.3 Bag of Words

"A very common feature extraction procedures for sentences and documents is the bag-of-words approach (BOW). In this approach, we look at the histogram of the words within the text, i.e. considering each word count as a feature".

Yoav Goldberg

The Bag of Words (BOW) model is a popular and simple feature extraction technique used when we work with text. It describes the occurrence of each word within a document, by converting text into numbers. Especially vector of numbers, 2.18. Any information about the order, syntax or structure of words is discarded. We could split this procedure into two different steps for the final outcome:

- The implementation of a vocabulary of known words (also called tokens): it is composed from all the unique words from the given data.
- A measure of the presence of known words: every word in a sentence is compared with all words in vocabulary in order to understand whether a known word occurs in a sentence or document and most important how many times this word is appears there (counts), no matter where it is located [15].

BOW results into a big vector of numbers, each number maps to a part of sentence and represents the count of this word in vocabulary. This type of text representation is the most basic and fundamental for feeding every kind of ML or DL algorithm. At the same time BOW suffer from limitations and computational reasons in case of big data. It can be claimed that, BOW introduced limitations such as large feature dimension and sparse representation. To start, the vocabulary requires careful design, most specifically in order to manage the size, which impacts the sparsity of the document representations. For a large document, the vector size can be huge resulting in a lot of computation and time. This means it is needed to ignore words, even relevant to your use case. In addition, the basic BOW approach does not consider the meaning of the word in the document. It completely ignores the context in which it's used. The same word can be used in multiple places based on the context or nearby words. Still this model can deal with a lot of today's cases with success, but it's needed firstly to consider carefully the NLP approach of your task [18][9]. On the other hand, most current state of the art approaches rely on a technique called text embedding.

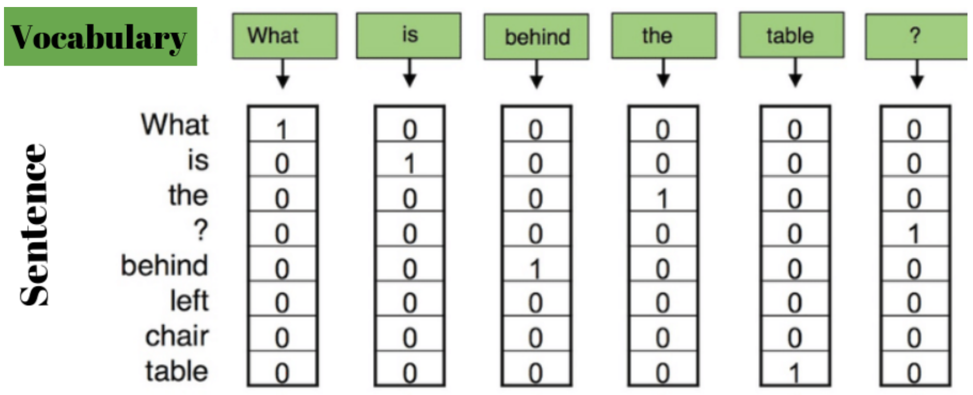


FIGURE 2.18: Bag-Of-Words (BOW) representation

2.3.4 Word Embedding

Distributed representations of words in a vector space help learning algorithms to achieve better performance in natural language processing tasks by grouping similar words. Word Embedding is an NLP technique, capable of capturing the context of a word in a document, semantic and syntactic similarity, relation with other words etc. It transforms text into a numerical representation in N-dimensional space. It allows for a word (or a document/sentence, depending on what embedding we use) to be expressed as a vector in this N-dimensional space so that semantically similar or semantically related words come closer depending on the training method. That is the basic advantage over Bag of Words model, because the last one neglects a lot of information like ordering and semantics of the words. For example, two sentences can have identical final representation but entirely different meanings. For example "I'm going to walk instead of sleeping" and "I'm going to sleep instead of walking". BOW method can't see different context those sentences but word embedding takes into consideration word relation with other words [13].

A word embedding is a representation of a word. They use numbers to represent words. For a neural network like GloVe, they may use $N = (50 - 500)$ numbers. Each of those numbers is in a dimension and each locates a word in N-dimensional space. Intuitively, we introduce some dependence of one word on the other words. The words in context of this word would get a greater share of this dependence. Once word embeddings have been trained, you can use them to derive similarities between words, as well as other relations. The typical example for word embeddings is that they allow you to perform math like this: King - Queen = Man -

Woman, because the difference between the word embeddings is similar and therefore King - Queen + Woman = Man. Another example presented in 2.19 has three dimensional space (wings, sky, engine) we can see how different words are located in space. The way they are located depends on embedding numbers in parenthesis. These numbers map to both semantic and syntactic relation of this word with the tree dimensions (which are our predefined words). As a result the word drone is placed in Sky-Engine half-plane. Another important fact of word embedding space is that when our embiddings are trained properly we can easily conclude to similarities between different words by using the mathematical equation of cosine (C). We could observe in Figure 2.19 that rocket-drone angle is smaller than rocket-helicopter, which means that rocket definition or usage in data, we have trained this model, is approaching drones and usually presented together. There

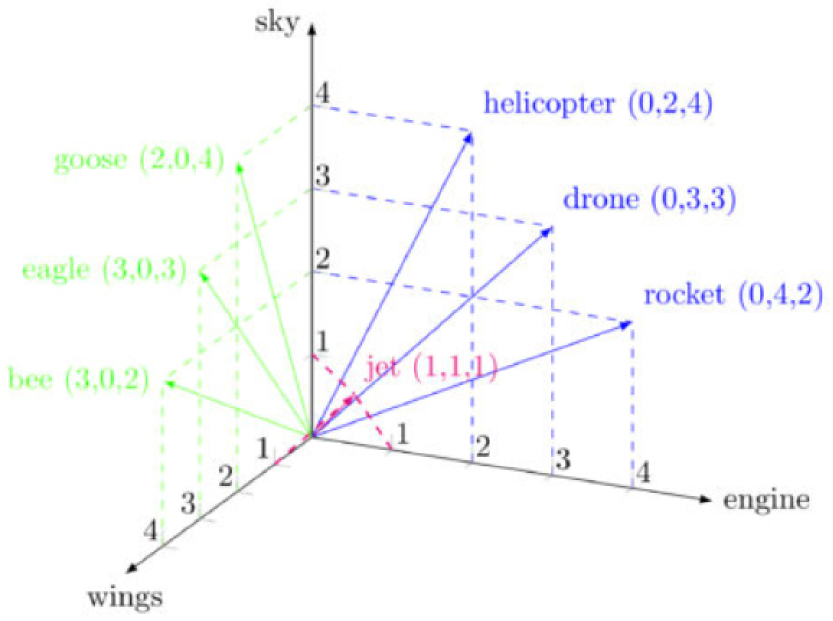


FIGURE 2.19: Three dimensional embedding space

are several predictive models using word embedding, such as:

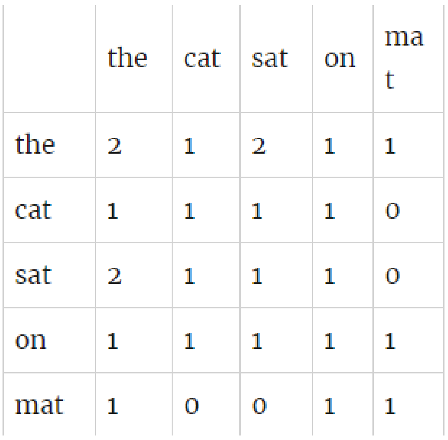
- Word2Vec Embedding: was created by a team of researchers led by Tomáš Mikolov at Google.
- FastText: was created by Facebook's AI Research lab.
- Global Vectors (GloVe): was created by Stanford NLP group.

Both GloVe and FastText have come afterwards and they are optimizations and/or different approaches. Other useful embedding representations are Sec2vec and Doc2vec for sentences and documents manipulation. We'll analyze later GloVe model as it mainly used in this thesis.

2.3.4.1 Global Vectors

To begin with, the predecessor of GloVe model is a neural predictive model, called Word2vec. Word2Vec stands for a sophisticated word embedding technique. This technique is based on the idea that words that occur in the same contexts tend to have similar meanings. In other words, this model tries to capture words co-occurrence with one "window" at time. The main principle of this method is to predict words based on their context by using one of two distinct neural models: CBOW and Skip-Gram. It does not take advantage of global count statistics. For example, "the" and "cat" might be used together often, but word2vec does not know if this is because "the" is a common word or if this is because the words "the" and "cat" have a strong linkage. This is how GloVe has appeared with the idea of using global count statistics.

GloVe is an unsupervised learning algorithm developed by Stanford researchers who argue that the ratio of word-word co-occurrence probabilities have the potential to encode some form of meaning. The authors of GloVe note that these context window-based methods suffer from the disadvantage of not learning from the global corpus statistics. Consequently, repetition and large-scale patterns may not be learned as well with these models as they do with global matrix factorization. So they introduce the idea that, the co-occurrence ratios between two words in a context are strongly connected to the meaning and has the potential for encoding it. Therefore, the training objective is to learn word vectors such that their dot product equals the logarithm of the word's probability of co-occurrence. As the logarithm of a ratio equals the difference of logarithms, this objective associates the ratios of co-occurrence probabilities with vector differences in the word vector space [40]. To start with the algorithm, GloVe uses the co-occuring matrix, e.g. Figure 2.20, in order to produce co-occurrence between words, but first uses neural methods to decompose the co-occurrence matrix into more expressive and dense word vectors by matrix factorization. However, the authors of GloVe discovered via empirical methods that instead of learning the raw co-occurrence probabilities,

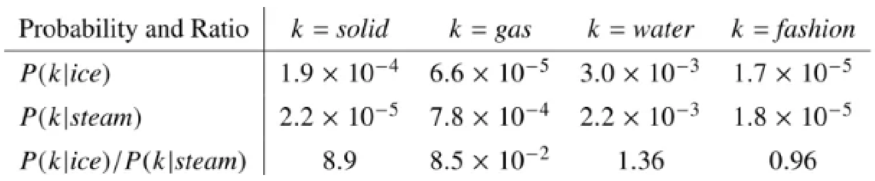


	the	cat	sat	on	ma t
the	2	1	2	1	1
cat	1	1	1	1	0
sat	2	1	1	1	0
on	1	1	1	1	1
mat	1	0	0	1	1

FIGURE 2.20: Three dimensional embedding space, (Paper Dissected: "Glove: Global Vectors for Word Representation", Keita Kurita, 2018)

it may make more sense to learn ratios of these co-occurrence probabilities, which seem to better discriminate subtleties in term-term relevance. To explain this we will use a simple example as presented by Pennington et al. (2014):

The relationship words can be revealed by studying the ratio of their co-occurrence probabilities with various probe words, k. Let $P(k|w)$ be the probability that the word k appears in the context of word w: ice co-occurs more frequently with solid than it does with gas, whereas steam co-occurs more frequently with gas than it does with solid. Both words co-occur frequently with water (as it is their shared property) and infrequently with the unrelated word fashion. In other words, $P(solid|ice)$ will be relatively high, and $P(solid|steam)$ will be relatively low. Therefore, the ratio of $P(solid|ice)/P(solid|steam)$ will be large. If we take a word such as gas that is related to steam but not to ice, the ratio of $P(gas|ice)/P(gas|steam)$ will instead be small. For a word related to both ice and steam, such as water we expect the ratio to be close to one. The following Figure 2.21 shows the actual statistics:



Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

FIGURE 2.21: Probability and Ratio example,(Paper Dissected: "Glove: Global Vectors for Word Representation", Keita Kurita, 2018)

To proceed with the GloVe algorithm we will use the mathematical equation of co-occurrence probability.

$$P_{\mathrm{ij}} = P(j|i) = \frac{X_{\mathrm{ij}}}{X_{\mathrm{i}}} = \frac{X_{\mathrm{ij}}}{\sum_k X_{\mathrm{ik}}} \tag{2.16}$$

$$X_{\mathrm{ij}} = \text{ number of times word j occurs in the context of word i} \tag{2.17}$$

We want to take data from the corpus in the form of global statistics and learn a function that gives us information about the relationship between any two words in said corpus. The function our model is learning is given by F. A Naïve interpretation of the desired model is given by the authors as:

$$F(w_i, w_j, \tilde{w_k}) = \frac{P_{\mathrm{ik}}}{P_{\mathrm{jk}}} \tag{2.18}$$

We could say that there are two kinds of embeddings: input and output, expressed as w and $\tilde{w}$) for the context and focus words. Since we want to encode information about the ratios between two words, the authors suggest using vector differences as inputs to our function. Then we have the following equation:

$$F(w_i - w_j, \tilde{w_k}) = \frac{P_{\mathrm{ik}}}{P_{\mathrm{jk}}} \tag{2.19}$$

The authors also design a weighting scheme for co-occurrences to reflect the relationship between frequency and semantic relevance. Consequently, we have two arguments, the context word vector, and the vector difference of the two words. Since the authors wish to take scalar to scalar values (the ratio of probabilities is a scalar), the dot product of these two arguments is taken, as we can see in next equation:

$$F(dot(w_i - w_j, \tilde{w_k})) = \frac{P_{\mathrm{ik}}}{P_{\mathrm{jk}}} \tag{2.20}$$

By taking the log of the probability ratios, we can convert the ratio into a subtraction between probabilities

$$F(dot(w_i - w_j, \tilde{w_k})) = \log P_{\mathrm{ik}} - \log P_{\mathrm{jk}} \tag{2.21}$$

By absorbing the final term on the right-hand side into the bias term, and adding an output bias for symmetry, we get

$$F(dot(w_i - w_j, \tilde{w_k})) + \tilde{b_k} = \log P_{\mathrm{ik}} \tag{2.22}$$

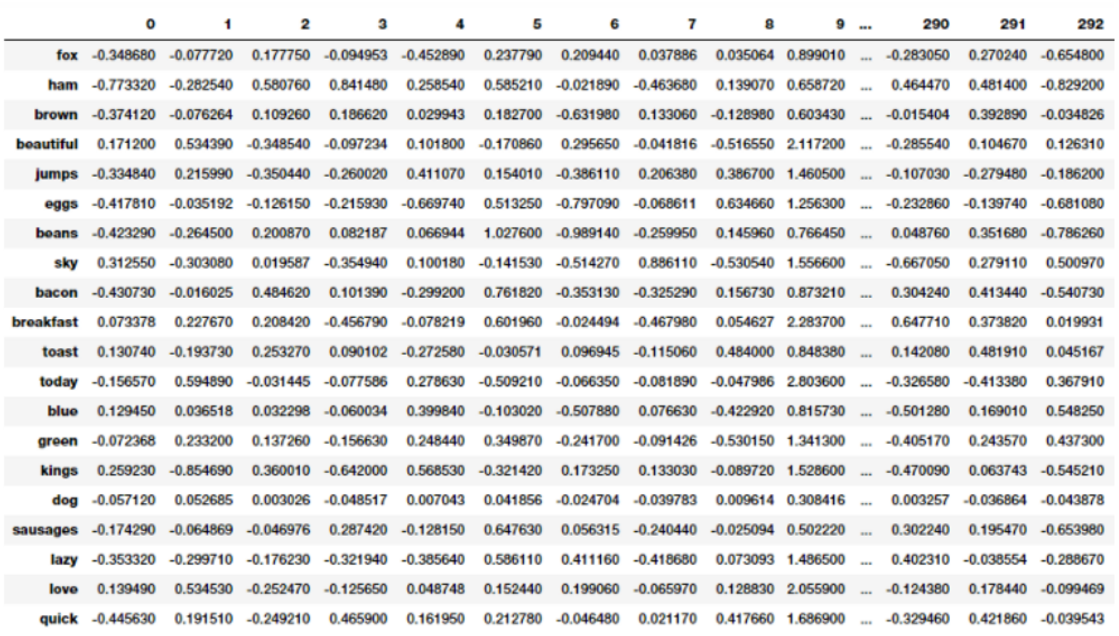
There is one problem with the equation above: it weights all co-occurrences equally. Unfortunately, not all co-occurrences have the same quality of information. Co-occurrences that are infrequent will tend to be noisy and unreliable, so we want to weight frequent co-occurrences more heavily. On the other hand, we don't want co-occurrences like "it is" dominating the loss, so we don't want to weight too heavily based on frequency. Through experimentation, the authors of the paper found the following weighting function to perform relatively well [24][47]:

$$weight(x) = \min\left(1, \left(\frac{x}{x_{\max}}\right)^{\frac{3}{4}}\right) \tag{2.23}$$

Essentially, the function gradually increases with x but does not become any larger than 1. Using this function, the loss becomes:

$$\sum_{\text{ij}} weight(X_{\text{ij}})((dot(w_i, \tilde{w_j})) + \tilde{b_k} - \log X_{\text{ik}})^2 \tag{2.24}$$

To conclude, GloVe embeddings are going to be equal with these, in Figure 2.22. Any DL or ML algorithm could be fed with this type of input but contrary to other types of text representation, now we consider both global word context and syntactic between words [25].



	0	1	2	3	4	5	6	7	8	9	...	290	291	292
fox	-0.348680	-0.077720	0.177750	-0.094953	-0.452890	0.237790	0.209440	0.037886	0.035064	0.899010	...	-0.283050	0.270240	-0.654800
ham	-0.773320	-0.282540	0.580760	0.841480	0.258540	0.585210	-0.021890	-0.463680	0.139070	0.658720	...	0.464470	0.481400	-0.829200
brown	-0.374120	-0.076264	0.109260	0.186620	0.029943	0.182700	-0.631980	0.133060	-0.128980	0.603430	...	-0.015404	0.392890	-0.034826
beautiful	0.171200	0.534390	-0.348540	-0.097234	0.101800	-0.170860	0.295650	-0.041816	-0.516550	2.117200	...	-0.285540	0.104670	0.126310
jumps	-0.334840	0.215990	-0.350440	-0.260020	0.411070	0.154010	-0.386110	0.206380	0.386700	1.460500	...	-0.107030	-0.279480	-0.186200
eggs	-0.417810	-0.035192	-0.126150	-0.215930	-0.669740	0.513250	-0.797090	-0.068611	0.634660	1.256300	...	-0.232860	-0.139740	-0.681080
beans	-0.423290	-0.264500	0.200870	0.082187	0.066944	1.027600	-0.989140	-0.259950	0.145960	0.766450	...	0.048760	0.351680	-0.786260
sky	0.312550	-0.303080	0.019587	-0.354940	0.100180	-0.141530	-0.514270	0.886110	-0.530540	1.556600	...	-0.667050	0.279110	0.500970
bacon	-0.430730	-0.016025	0.484620	0.101390	-0.299200	0.761820	-0.353130	-0.325290	0.156730	0.873210	...	0.304240	0.413440	-0.540730
breakfast	0.073378	0.227670	0.208420	-0.456790	-0.078219	0.601960	-0.024494	-0.467980	0.054627	2.283700	...	0.647710	0.373820	0.019931
toast	0.130740	-0.193730	0.253270	0.090102	-0.272580	-0.030571	0.096945	-0.115060	0.484000	0.848380	...	0.142080	0.481910	0.045167
today	-0.156570	0.594890	-0.031445	-0.077586	0.278630	-0.509210	-0.066350	-0.081890	-0.047986	2.803600	...	-0.326580	-0.413380	0.367910
blue	0.129450	0.036518	0.032298	-0.060034	0.399840	-0.103020	-0.507880	0.076630	-0.422920	0.815730	...	-0.501280	0.169010	0.548250
green	-0.072368	0.233200	0.137260	-0.156630	0.248440	0.349870	-0.241700	-0.091426	-0.530150	1.341300	...	-0.405170	0.243570	0.437300
kings	0.259230	-0.854690	0.360010	-0.642000	0.568530	-0.321420	0.173250	0.133030	-0.089720	1.528600	...	-0.470090	0.063743	-0.545210
dog	-0.057120	0.052685	0.003026	-0.048517	0.007043	0.041856	-0.024704	-0.039783	0.009614	0.308416	...	0.003257	-0.036864	-0.043878
sausages	-0.174290	-0.064869	-0.046976	0.287420	-0.128150	0.647630	0.056315	-0.240440	-0.025094	0.502220	...	0.302240	0.195470	-0.653980
lazy	-0.353320	-0.299710	-0.176230	-0.321940	-0.385640	0.586110	0.411160	-0.418680	0.073093	1.486500	...	0.402310	-0.038554	-0.288670
love	0.139490	0.534530	-0.252470	-0.125650	0.048748	0.152440	0.199060	-0.065970	0.128830	2.055900	...	-0.124380	0.178440	-0.099469
quick	-0.445630	0.191510	-0.249210	0.465900	0.161950	0.212780	-0.046480	0.021170	0.417660	1.686900	...	-0.329460	0.421860	-0.039543

FIGURE 2.22: A sample vector of GloVe embeddings

Chapter 3

Experimental Setup

3.1 Overview

To answer the research questions of this study, multiple selected algorithms were applied on various data from online datasets and the Twitter Api. The algorithms were implemented using the open source programming language Python. Due to expensive computational time of both the experiments and the implementation of models, we have used for our system three servers from Google Compute Engine. The original idea was to build two different models for text classification. As text we have tweets from the Twitter platform. The role of the first model is to identify if tweet text is relative to a group of disasters. The second model comes after the first's work. Its purpose is to learn how to classify tweets related to disaster in one of the predefined disasters: Earthquake, Hurricane. Both models were trained with several algorithms that we will talk about later in this chapter.

The ultimate goal of this research is to monitor one of the above disaster events into what people chatter and post in Twitter, locate it in the world map and approximate the time of this event.

3.2 Datasets

Multiple datasets were used in the experiments from CrisisNLP and CrisisLex. Datasets had been created from both payed workers and volunteers, crawled from

Table 3.1: Datasets used for this experiment

Datasets	**Num. of observations**	**Num. of Features**	**Num. of classes**	**Percentage minority class**
Disasters	79,194	1	2	40%
Earthquake or Hurricane	27,434	1	2	50%

the Twitter API,(Appendix B), and talking about the following disasters: Typhoon, Earthquake, Cyclone, Tornado. The purpose of use differs to my purpose, so I have purposely changed labels for each review. Table 3.1 provides an overview of the properties of all datasets. More details about each dataset can be found in the coming subsections.

3.2.1 Disasters

The first dataset have created named Disasters contains both text and numerical features and is a concatenation from the datasets listed below:

- 2012 Sandy Hurricane - CrisisLex
- 2010 Chile Earthquake - CrisisLex
- 2013 Boston Bombings - CrisisLex
- 2013 Oklahoma Tornado - CrisisLex
- 2013 Alberta Floods - CrisisLex
- 2015 Nepal Earthquake - CrisisNLP
- 2013 Pakistan Earthquake - CrisisNLP
- 2014 Chile Earthquake - CrisisNLP
- 2014 California Earthquake - CrisisNLP
- 2014 Hurricane Odile - CrisisNLP
- 2014 Pakistan Floods - CrisisNLP

- 2015 Cyclone PAM - CrisisNLP
- 2014 Philippines Typhoon Hagupit/Ruby - CrisisNLP

This dataset has one feature, named Text and one class named Choose one. There are 79,194 observations with text corresponding to real tweets crawled by Twitter Api. Every single text is related to one of the following disasters (map with number 1): Typhoon, Hurricane, Earthquake, Cyclone, Tornado (Huricane-Typhoon-Cyclone-Tornado considered as the same natural disaster) or it is totally irrelevant (map with number 0) to them. Note here that even other types of disasters, such as flood and bombing, are assumed as irrelevant text considering the type of disasters we want to specify. The point of this dataset is not to identify the type of the disaster but to split tweets in Relevant or Irrelevant with a disaster. We could say that the class's (Choose one) distribution is not totally balanced but 60% for relevant and 40% for irrelevant are percentages we can work with, Figure 3.1.

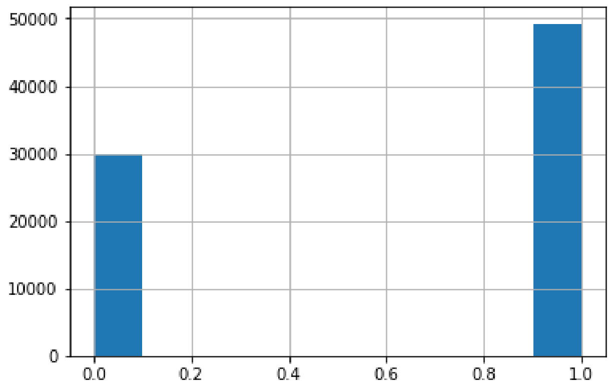


FIGURE 3.1: Disasters Dataset - Class Histogram

3.2.2 Earthquake or Hurricane

The second dataset i have created named Earthquake or Hurricane contains both text and numerical features and again is a selective concatenation from the datasets listed below:

- 2012 Sandy Hurricane - CrisisLex
- 2010 Chile Earthquake - CrisisLex
- 2013 Oklahoma Tornado - CrisisLex

- 2015 Nepal Earthquake - CrisisNLP
- 2013 Pakistan Earthquake - CrisisNLP
- 2014 Chile Earthquake - CrisisNLP
- 2014 California Earthquake - CrisisNLP
- 2014 Hurricane Odile - CrisisNLP
- 2014 Philippines Typhoon Hagupit/Ruby - CrisisNLP

This dataset, as the first dataset we referred above, has one feature, named Text and one class named Choose one. There are 27,434 observations with text corresponding to real tweets crawled by Twitter Api. Text feature in this data are talking about an earthquake (map with number 1) event or a hurricane event (map with number 0). Every row has a text relative to earthquake or hurricane. The main purpose of this dataset is to identify the type of the disaster between them. The class (Choose one) distribution is totally balanced with percentages: 51% for hurricane and 49% for earthquake, Figure 3.2. To conclude about the

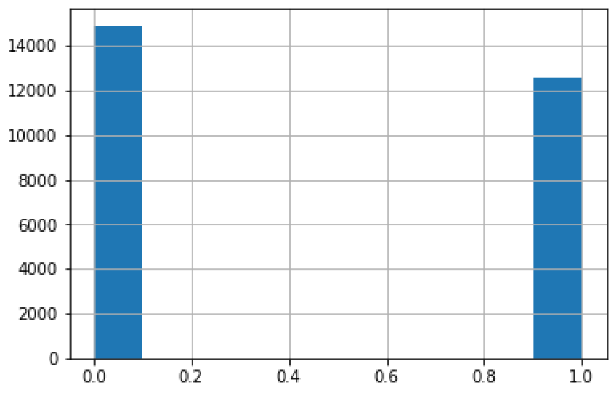
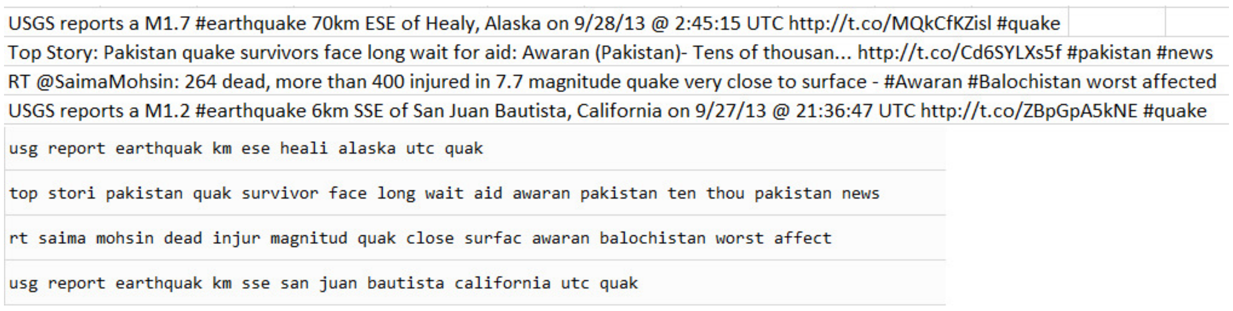


FIGURE 3.2: Earthquake or Hurricane Dataset - Class Histogram

datasets, mention that the first one (Disasters) is used for the first model and dataset Earthquake or Hurricane is also used for the second model.

3.3 Data Pre-Processing

This step was fundamental for this thesis because we had to manipulate every day tweets with a lot of abbreviations, emotes etc. Many NLP techniques have been used and listed in Section 2.3.1 with some help from scikit learn libraries. To be more specific, the first step was the text reformation to lower case characters and elimination of numbers and every kind of special characters except hashtags (#). In tweets, hashtags usually provide useful information, also every hashtag has the idiom of many words merged without space after the symbol. This problem was solved with a very useful library, named wordninja which extracts every meaningful word after a hashtag. Another part was the elimination of any form of markup language, such as: "http" All emotes, symbols, pictographs, map symbols iOS flags and transports removed with custom code. Next step was the transformation of abbreviations. For example, phrases like "I've" had been transformed to their full form, which is I have. Next step for pre-processing procedure had to do with a technique for text features called Stemming. Skicit learn provide a ready for use algorithm to stem every single word in a text, which were used for our purposes, but before this the method of tokenization was necessary to split sequences into single words. In addition, frequent English words, known as stopwords, were removed from each sentence again with help from sk-learn libraries. An example before pre-processing and after is represented in Figure 3.3



```
USGS reports a M1.7 #earthquake 70km ESE of Healy, Alaska on 9/28/13 @ 2:45:15 UTC http://t.co/MQkCfKZisl #quake
Top Story: Pakistan quake survivors face long wait for aid: Awaran (Pakistan)- Tens of thousan... http://t.co/Cd6SYLXs5f #pakistan #news
RT @SaimaMohsin: 264 dead, more than 400 injured in 7.7 magnitude quake very close to surface - #Awaran #Balochistan worst affected
USGS reports a M1.2 #earthquake 6km SSE of San Juan Bautista, California on 9/27/13 @ 21:36:47 UTC http://t.co/ZBpGpA5kNE #quake
```

```
usg report earthquak km ese heali alaska utc quak
top stori pakistan quak survivor face long wait aid awaran pakistan ten thou pakistan news
rt saima mohsin dead injur magnitud quak close surfac awaran balochistan worst affect
usg report earthquak km sse san juan bautista california utc quak
```

FIGURE 3.3: Result of Pre-processing compared with the original tweet

3.3.1 Data Analysis

After all the above procedures in the pre-processing task, we observe that a lot of text reviews had been empty. This can be explained because of the tweets nature. It is an often phenomenon for a tweet to have only emotes or links to other web pages or even photographs. So after NLP processing none of the above is part of the sentence and the result can be an empty field. In this thesis, rows with NaN as text were totally removed from the original datasets.

Data Pre-processing and Analysis were used for both “Disasters” and “Earthquake or Hurricane” datasets. The result was a big text corpus for each file of data which was later used for different models training. In figure they are represented the two text corpora the pre-processing task result in.

3.4 Feature Extraction

Next milestone of this thesis has to do with feature extraction from text, as tweet’s text is the only feature used for classification of that observations. In this task two different approaches were followed which were finally compared from their results. Of course to compare their results, we needed to build the appropriate ML or DL models which is the next section of this thesis.

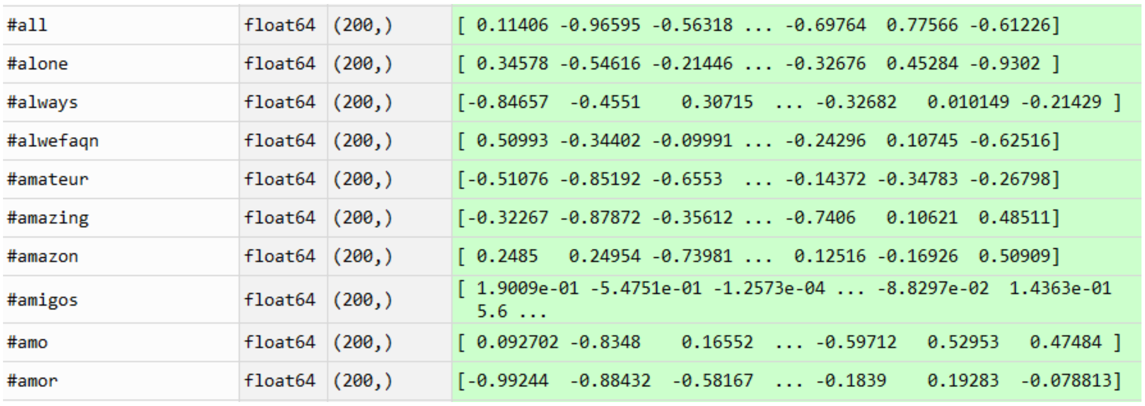
The two techniques have been used for text representation were: Bag of Words and Global Vectors, models as they were explained in Sections 2.3.3 and 2.3.4.1:

- To build BoW model we have used CountVectorizer library from sk-learn, which helped us to identify how many times a predefined number of words occur in every message (tweet’s text). The goal was to represent word corpus (for both the first and the second dataset) as a vector of numbers. It’s number map to an occurrence for this word in this tweet. First we defined one thousand (2000) features, corresponding to the most frequent words in corpus. CountVecorrizer algorithm searches for these words in every tweet and keep their occurrence. The result was the number of vectors. This vector is later used to train our models.

- To build the Glove model a totally different method was used. Firstly, GloVe model need a big amount of data to be trained. In consideration of this a ready implemented and trained model from Stanford NLP group was used. This model had been trained in 2 billion tweets, 27 billion tokens and 1.2 million of vocabulary words all in English language. Also 200 dimension vector out of 4 other options was picked [32]. First step was to load from the pre-trained model, the embedding vector with dimensions 1.2 million rows (which is words) and 200 columns, which is also words. For example the first row has the word “adult” and 200 words in columns related to “adult”. The advantage of this method, as mentioned in the first chapter, is that these

200 words have a relation in context and syntax with "adult". This relation had came up from all 2 billion tweets not a single sentence, that's why we are talking about Global Vectors. In Figure 3.4 are the first 10 words with their embeddings from the model, with positive and negative float numbers. Negative contrary with positives serves a low relation to each word.

Next task was to identify which of the words in our datasets are also in the vocabulary we talked about. For every word inside sentences in our datasets which they were in vocabulary, we crawled their embeddings. Then we produced the sum of every column which always results in a vector of 200 × 1 no matter how many words each sentence has. Next we normalized every matrix, which maps to a sentence of words. The GloVe algorithm is designed to produce similar vectors for semantically similar words with the cosine similarity computation. To conclude, the result was a big matrix with so many rows as in every dataset and 200 columns each of them contain numbers corresponding to similar words produced by the original text.



#all	float64	(200,)	[0.11406 -0.96595 -0.56318 ... -0.69764 0.77566 -0.61226]
#alone	float64	(200,)	[0.34578 -0.54616 -0.21446 ... -0.32676 0.45284 -0.9302]
#always	float64	(200,)	[-0.84657 -0.4551 0.30715 ... -0.32682 0.010149 -0.21429]
#alwefaqn	float64	(200,)	[0.50993 -0.34402 -0.09991 ... -0.24296 0.10745 -0.62516]
#amateur	float64	(200,)	[-0.51076 -0.85192 -0.6553 ... -0.14372 -0.34783 -0.26798]
#amazing	float64	(200,)	[-0.32267 -0.87872 -0.35612 ... -0.7406 0.10621 0.48511]
#amazon	float64	(200,)	[0.2485 0.24954 -0.73981 ... 0.12516 -0.16926 0.50909]
#amigos	float64	(200,)	[1.9009e-01 -5.4751e-01 -1.2573e-04 ... -8.8297e-02 1.4363e-01 5.6 ...
#amo	float64	(200,)	[0.092702 -0.8348 0.16552 ... -0.59712 0.52953 0.47484]
#amor	float64	(200,)	[-0.99244 -0.88432 -0.58167 ... -0.1839 0.19283 -0.078813]

FIGURE 3.4: Sample of Pre-trained Global Vectors

As it was expected GloVe model, because of its ability to represent context and syntactic of all words in a dataset/document, was a lot more accurate for this purpose. Considering the results with BoW and GloVe with LSTM model, in Figure 3.5, we identified that Global vectors had actually performed better in our case. This result led to continue only with GloVe model for all algorithms.

3.5 Algorithms

As discussed in Chapter 2 multiple algorithms from both ML and DL were selected based on their applicability and their performance from previous studies. In this

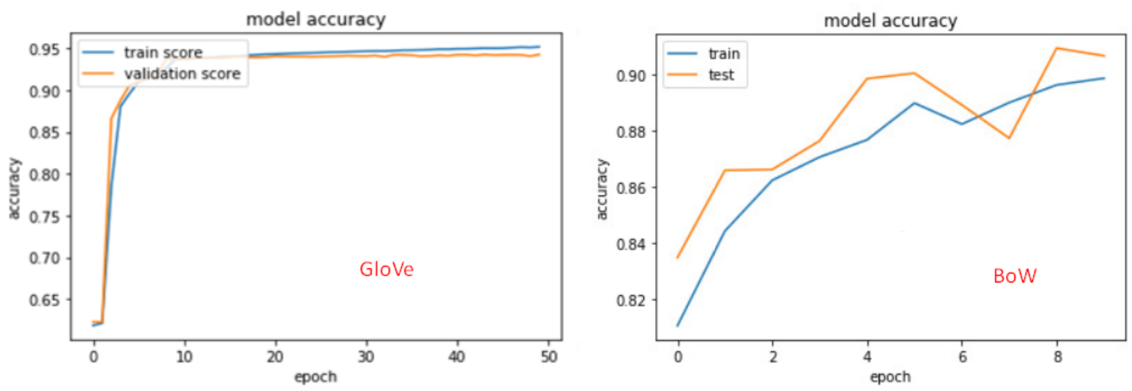


Figure 3.5: BoW vs GloVe with LSTM model

part of the thesis, 8 different algorithms had been used from ML and DL listed below:

- Logistic Regression (LR)
- Random Forest (RF)
- Bagged Classifier (BgC)
- Decision Tree (DT)
- Adaboost (Ada)
- Naive Bayes (NB)
- Deep Neural Network (DNN)
- Long-Short Term Memory (LSTM)

The initial point of this task had been to search for the best algorithm for every model, that's why eight different algorithms have been tried.

3.6 Training

In order to train both models Datasets were split. A percentage of 80% were used to train models and 20% to test them. Test of the algorithms with the 20% of datasets was not the final experiment, so we will talk about it in next chapter.

3.6.1 Machine learning Algorithms

For ML algorithms in order to achieve better performance for the models cross validation technique was used as described in Section 2.1 and hyperparameter tuning. To evaluate the effectiveness of each model we have used Accuracy, 2.1, as the performance metric. The idea is to find out the correct prediction out of all predictions that's why Accuracy seemed to be the best for this case. We have also investigate F1-score for ML algorithms to confirm model's efficiency.

3.6.1.1 Hyperparameter tuning

In order to boost the performance of each ML algorithm, their hyperparameters were optimized first. Both Grid-search and Random-search were used to find the best set of hyperparameters from a manually chosen subset of the hyperparameter space of each algorithm. These candidate hyperparameter values for supervised algorithms: Logistic Regression, AdaBoost, Random Forest, Bagged classifier (with decision tree as weak learner), Naive Bayes were chosen based on recommendations from Scikit-learn. Searches use brute-force search to enumerate all possible candidate sets of hyperparameters and validates their performance using cross validation on the 80% training data. In some cases multiple hyperparameter sets achieved similar results in terms of the accuracy, but differed in terms of computational effort. In such situations, the hyperparameters with lowest run-time were selected.

3.6.2 Deep learning Algorithms

Deep Learning algorithms that were used are: DNN and LSTM. These algorithms were used to build both models of the thesis. Both models have fed with batch size of 64 rows and have trained for 50 epochs.

- DNN: The architecture of DNN for both models is the same and it has been custom created after many previous studies and implementations with help of Keras library. The structure of this DNN is basic architecture as we could see in Figure 3.9. It is composed of 1 input layer, 3 hidden layers and one output layer. The input layer has 200 neurons, first, second and third hidden layer have 32, 64, 128 neurons equally. Last one the out put layer has only one neuron for binary classification. As we can see there are also some layers, called dropout layers. As mentioned in Wikipedia: Dropout is a regularization technique patented by Google for reducing over-fitting in neural networks by preventing complex co-adaptations on training data. It is a very efficient way of performing model averaging with neural networks. The term "dropout" refers to dropping out units (both hidden and visible) in a neural network [5]. To link the DNN model with the final matrix from the GloVe model, we could easily comprehend that the input layer has 200 neurons on for every feature (embedding).

Furthermore, every layer of DNN is fully connected with the next one and their weights were initializes with uniform function. For every neuron in each layer except the output layer, in order to map the sum the inputs multiplied with their weights we have used Relu as activation function. On the other hand output layer followed the equation of Sigmoid as activation function. The choice of activation functions was mostly empirical after useful reading about their dis/advantages. To summarize DNN architecture, in order to optimize model Stochastic Gradient Descent has been used, as optimization function to update weights properly. In every step the calculated loss function (error between output and target value) was: Binary Cross Entropy as we can see in Equation 3.1, where y is the label, $P(y)$ is the predicted probability and N the data points.

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \times \log(p(y_i)) + (1-y_i) \times \log(1-p(y_i)) \tag{3.1}$$

- LSTM: For the LSTM architecture we used Glove in a different way. First, we have found the 2000 most common words in the dataset. We used the pre-trained Glove model from Stanford NLP group to map our words with the given embeddings. Consequently we have a vector with 2000 rows (words) and 300 embeddings (words in form of number which they often used with our tokens). This vector could also be named as embedding matrix of our vocabulary for our case and we can see a small sample in Figure 3.4. Then, we predefined that each observation (tweet's text) of our datasets will have at max 50 words (Note here that for tweets with less than 50 words the empty slots had filled with zeros). We have mapped each word in each observation with the number that this word has had in the embedding matrix. The outcome of this procedure will be the input for our LSTM and could be comprehended with Figure 3.8. We have also used the embedding matrix with the 2000 words and their 300 embeddings for each, to initialize our weights in contrast with the simple DNN we talked about before in which the weights had followed uniform initialization. Finally, we do not want to update the learned word weights in every step in this model. Weights remain as in the original embedding matrix until the Embedding layer, but in the next block (Bi-LSTM layer) weights are totally trainable. In Figure 3.6, we can see the architecture we have used for our LSTM in both datasets. Our model consists of:

- Input Layer
- Embedding Layer
- LSTM Layer with 10 memory units
- Max-Pooling Layer
- Output Layer

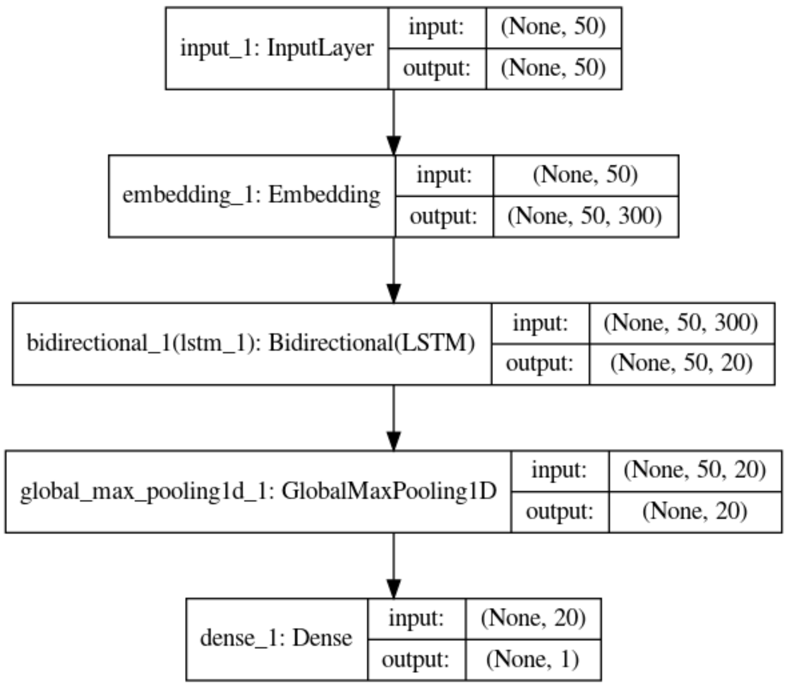
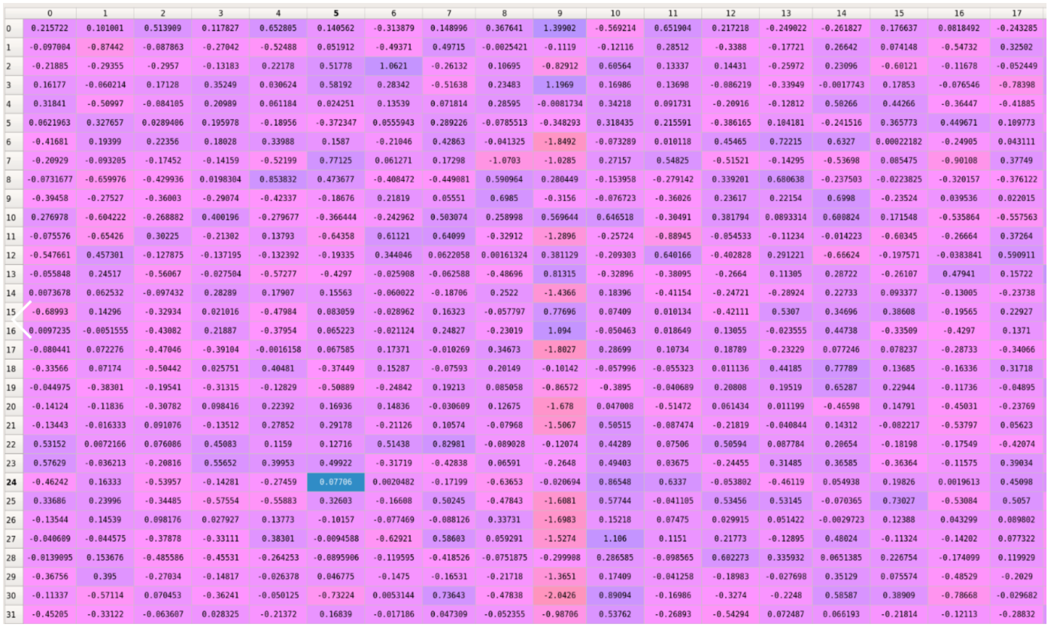


FIGURE 3.6: Architecture used for LSTM

Firstly, our model runs for 50 epochs with 128 batch size,and these numbers have been defined after lots of experiments. The input layer is the first layer



	0	1	2	3	4	5	6	7	8
0	0.215722	0.101001	0.513909	0.117827	0.652805	0.140562	-0.313879	0.148996	0.367641
1	-0.097004	-0.87442	-0.087863	-0.27042	-0.52488	0.051912	-0.49371	0.49715	-0.0025421
2	-0.21885	-0.29355	-0.2957	-0.13183	0.22178	0.51778	1.0621	-0.26132	0.10695
3	0.16177	-0.060214	0.17128	0.35249	0.030624	0.58192	0.28342	-0.51638	0.23483
4	0.31841	-0.50997	-0.084105	0.20989	0.061184	0.024251	0.13539	0.071814	0.28595
5	0.0621963	0.327657	0.0289406	0.195978	-0.18956	-0.372347	0.0555943	0.289226	-0.0785513
6	-0.41681	0.19399	0.22356	0.18028	0.33988	0.1587	-0.21046	0.42863	-0.041325
7	-0.20929	-0.093205	-0.17452	-0.14159	-0.52199	0.77125	0.061271	0.17298	-1.0703
8	-0.0731677	-0.659976	-0.429936	0.0198304	0.853832	0.473677	-0.408472	-0.449081	0.590964
9	-0.39458	-0.27527	-0.36003	-0.29074	-0.42337	-0.18676	0.21819	0.05551	0.6985
10	0.276978	-0.604222	-0.268882	0.400196	-0.279677	-0.366444	-0.242962	0.503074	0.258998
11	-0.075576	-0.65426	0.30225	-0.21302	0.13793	-0.64358	0.61121	0.64099	-0.32912
12	-0.547661	0.457301	-0.127875	-0.137195	-0.132392	-0.19335	0.344046	0.0622058	0.00161324
13	-0.055848	0.24517	-0.56067	-0.027504	-0.57277	-0.4297	-0.025908	-0.062588	-0.48696
14	0.0073678	0.062532	-0.097432	0.28289	0.17907	0.15563	-0.060022	-0.18706	0.2522
15	-0.68993	0.14296	-0.32934	0.021016	-0.47984	0.083059	-0.028962	0.16323	-0.057797
16	0.0097235	-0.0051555	-0.43082	0.21887	-0.37954	0.065223	-0.021124	0.24827	-0.23019
17	-0.080441	0.072276	-0.47046	-0.39104	-0.0016158	0.067585	0.17371	-0.010269	0.34673
18	-0.33566	0.07174	-0.50442	0.025751	0.40481	-0.37449	0.15287	-0.07593	0.20149
19	-0.044975	-0.38301	-0.19541	-0.31315	-0.12829	-0.50889	-0.24842	0.19213	0.085058
20	-0.14124	-0.11836	-0.30782	0.098416	0.22392	0.16936	0.14836	-0.030609	0.12675
21	-0.13443	-0.016333	0.091076	-0.13512	0.27852	0.29178	-0.21126	0.10574	-0.07968
22	0.53152	0.0072166	0.076086	0.45083	0.1159	0.12716	0.51438	0.82981	-0.089028
23	0.57629	-0.036213	-0.20816	0.55652	0.39953	0.49922	-0.31719	-0.42838	0.06591
24	-0.46242	0.16333	-0.53957	-0.14281	-0.27459	0.07706	0.0020482	-0.17199	-0.63653
25	0.33686	0.23996	-0.34485	-0.57554	-0.55883	0.32603	-0.16608	0.50245	-0.47843
26	-0.13544	0.14539	0.098176	0.027927	0.13773	-0.10157	-0.077469	-0.088126	0.33731
27	-0.040609	-0.044575	-0.37878	-0.33111	0.38301	-0.0094588	-0.62921	0.58603	0.059291
28	-0.0139095	0.153676	-0.485586	-0.45531	-0.264253	-0.0895906	-0.119595	-0.418526	-0.0751875
29	-0.36756	0.395	-0.27034	-0.14817	-0.026378	0.046775	-0.1475	-0.16531	-0.21718
30	-0.11337	-0.57114	0.070453	-0.36241	-0.050125	-0.73224	0.0053144	0.73643	-0.47838
31	-0.45205	-0.33122	-0.063607	0.028325	-0.21372	0.16839	-0.017186	0.047309	-0.052355

	9	10	11	12	13	14	15	16	17
0	1.39902	-0.569214	0.651904	0.217218	-0.249022	-0.261827	0.176637	0.0818492	-0.243285
1	-0.1119	-0.12116	0.28512	-0.3388	-0.17721	0.26642	0.074148	-0.54732	0.32502
2	-0.82912	0.60564	0.13337	0.14431	-0.25972	0.23096	-0.60121	-0.11678	-0.052449
3	1.1969	0.16986	0.13698	-0.086219	-0.33949	-0.0017743	0.17853	-0.076546	-0.78398
4	-0.0081734	0.34218	0.091731	-0.20916	-0.12812	0.50266	0.44266	-0.36447	-0.41885
5	-0.348293	0.318435	0.215591	-0.386165	0.104181	-0.241516	0.365773	0.449671	0.109773
6	-1.8492	-0.073289	0.010118	0.45465	0.72215	0.6327	0.00022182	-0.24905	0.043111
7	-1.0285	0.27157	0.54825	-0.51521	-0.14295	-0.53698	0.085475	-0.90108	0.37749
8	0.280449	-0.153958	-0.279142	0.339201	0.680638	-0.237503	-0.0223825	-0.320157	-0.376122
9	-0.3156	-0.076723	-0.36026	0.23617	0.22154	0.6998	-0.23524	0.039536	0.022015
10	0.569644	0.646518	-0.30491	0.381794	0.0893314	0.600824	0.171548	-0.535864	-0.557563
11	-1.2896	-0.25724	-0.88945	-0.054533	-0.11234	-0.014223	-0.60345	-0.26664	0.37264
12	0.381129	-0.209303	0.640166	-0.402828	0.291221	-0.66624	-0.197571	-0.0383841	0.590911
13	0.81315	-0.32896	-0.38095	-0.2664	0.11305	0.28722	-0.26107	0.47941	0.15722
14	-1.4366	0.18396	-0.41154	-0.24721	-0.28924	0.22733	0.093377	-0.13005	-0.23738
15	0.77696	0.07409	0.010134	-0.42111	0.5307	0.34696	0.38608	-0.19565	0.22927
16	1.094	-0.050463	0.018649	0.13055	-0.023555	0.44738	-0.33509	-0.4297	0.1371
17	-1.8027	0.28699	0.10734	0.18789	-0.23229	0.077246	0.078237	-0.28733	-0.34066
18	-0.10142	-0.057996	-0.055323	0.011136	0.44185	0.77789	0.13685	-0.16336	0.31718
19	-0.86572	-0.3895	-0.040689	0.20808	0.19519	0.65287	0.22944	-0.11736	-0.04895
20	-1.678	0.047008	-0.51472	0.061434	0.011199	-0.46598	0.14791	-0.45031	-0.23769
21	-1.5067	0.50515	-0.087474	-0.21819	-0.040844	0.14312	-0.082217	-0.53797	0.05623
22	-0.12074	0.44289	0.07506	0.50594	0.087784	0.20654	-0.18198	-0.17549	-0.42074
23	-0.2648	0.49403	0.03675	-0.24455	0.31485	0.36585	-0.36364	-0.11575	0.39034
24	-0.020694	0.86548	0.6337	-0.053802	-0.46119	0.054938	0.19826	0.0019613	0.45098
25	-1.6081	0.57744	-0.041105	0.53456	0.53145	-0.070365	0.73027	-0.53084	0.5057
26	-1.6983	0.15218	0.07475	0.029915	0.051422	-0.0029723	0.12388	0.043299	0.089802
27	-1.5274	1.106	0.1151	0.21773	-0.12895	0.48024	-0.11324	-0.14202	0.077322
28	-0.299908	0.286585	-0.098565	0.602273	0.335932	0.0651385	0.226754	-0.174099	0.119929
29	-1.3651	0.17409	-0.041258	-0.18983	-0.027698	0.35129	0.075574	-0.48529	-0.2029
30	-2.0426	0.89094	-0.16986	-0.3274	-0.2248	0.58587	0.38909	-0.78668	-0.029682
31	-0.98706	0.53762	-0.26893	-0.54294	0.072487	0.066193	-0.21814	-0.12113	-0.28832

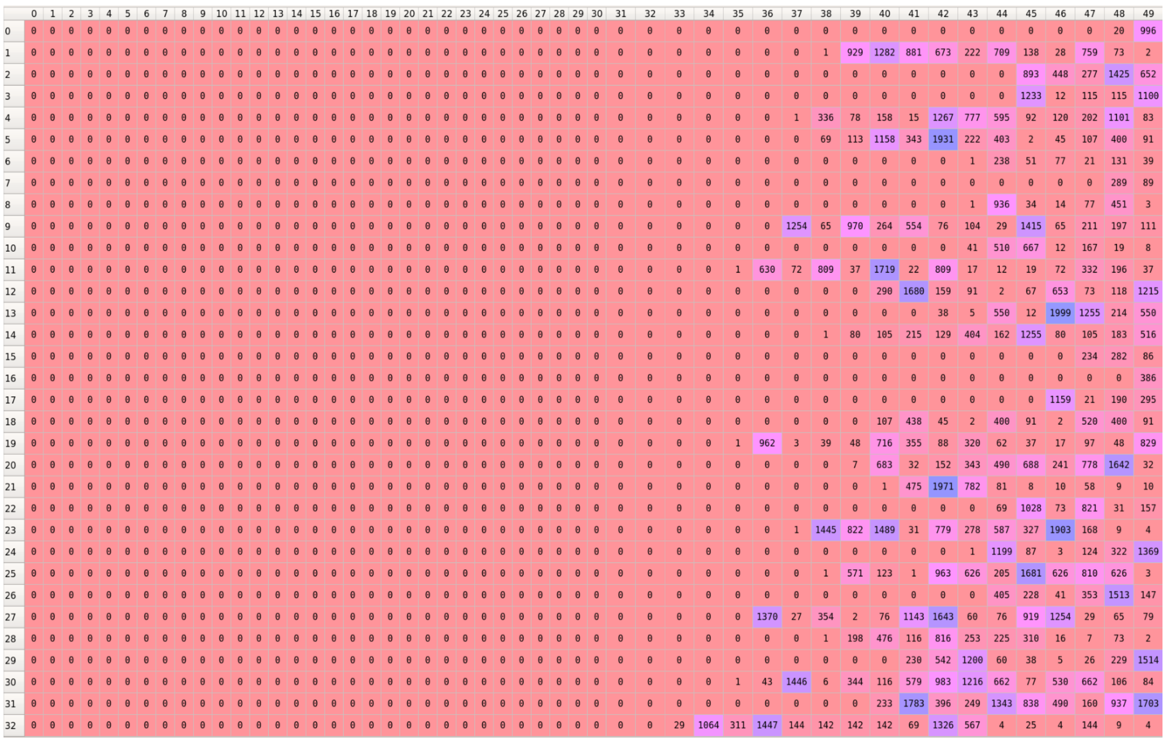
FIGURE 3.7: Sample of Embedding Matrix

in the network. It takes inputs (tokens) and passes them on to the next layer. It doesn't apply any operations on the inputs and has no weights and biases values associated. In our network we have 50 inputs, so 50 neurons and the same number of outputs. The Embedding layer is used to create word vectors for incoming words. Remember that its observation has 50 words and each word has 300 embiddings. So the embedding layer maps each of 50 words (which are the number of inputs in this layers) with their embeddings. It sits between the input and the LSTM layer, in other words the output of the Embedding layer is the input to the LSTM layer. In our case, we define an Embedding layer with a vocabulary of 2000 (e.g. integer encoded words from 0 to 1999, inclusive), a vector space of 300 dimensions in which words will be embedded and the embedding matrix as weights. The output of the Embedding layer is a 2D vector with 300 embeddings for each word in the input sequence of words, which are 50 words, so the output is a matrix 50 by 300.

For LSTM architecture we have used a different approach, where we wrapped the Lstm hidden layer with a bidirectional layer. This has created two copies of the hidden layer, one fits in the input text as-is and one on a reversed copy of the input text. By default, the output values from these LSTMs will be concatenated. That means that instead of receiving 50 time-steps of 10 outputs it will now receive 50 time-steps of 20 (10 units + 10 units) outputs.

Consequently the total output of this layer will be a matrix 50 by 20. The use of bidirectional LSTMs have the effect of allowing the LSTM to learn the problem faster. More information about Bidirectional can be found in Appendix D.

Then we used a layer before the final layer in order to down-sample the input matrix, which is the 50 by 20 output matrix from LSTM layer. The objective of this layer is to down-sample the input representation. In our case we have used a filter vector of 50 cells with stride equals to 50. That means that every 50 embeddings in the input matrix are represented with one element (Appendix E). The output of this layer is a vector of 20 elements. Finally we have fed this vector to the output layer which is a simple layer with 20 neurons for the 20 inputs and one output that is our prediction. To summarize LSTMs architecture, in order to optimize model it has been used Stochastic Gradient Descent as optimization function for the Bi-LSTM block to update weights properly. In every step the calculated loss function (error between output and target value) was: Binary Cross Entropy as we can see in Equation 3.1, where y is the label, $P(y)$ is the predicted probability and N the data points.



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20	996
1	0	0	0	0	0	0	1	929	1282	881	673	222	709	138	28	759	73	2
2	0	0	0	0	0	0	0	0	0	0	0	0	0	893	448	277	1425	652
3	0	0	0	0	0	0	0	0	0	0	0	0	0	1233	12	115	115	1100
4	0	0	0	0	0	1	336	78	158	15	1267	777	595	92	120	202	1101	83
5	0	0	0	0	0	0	69	113	1158	343	1931	222	403	2	45	107	400	91
6	0	0	0	0	0	0	0	0	0	0	0	1	238	51	77	21	131	39
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	289	89
8	0	0	0	0	0	0	0	0	0	0	0	1	936	34	14	77	451	3
9	0	0	0	0	0	1254	65	970	264	554	76	104	29	1415	65	211	197	111
10	0	0	0	0	0	0	0	0	0	0	0	41	510	667	12	167	19	8
11	0	0	0	1	630	72	809	37	1719	22	809	17	12	19	72	332	196	37
12	0	0	0	0	0	0	0	0	290	1680	159	91	2	67	653	73	118	1215
13	0	0	0	0	0	0	0	0	0	0	38	5	550	12	1999	1255	214	550
14	0	0	0	0	0	0	1	80	105	215	129	404	162	1255	80	105	183	516
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	234	282	86
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	386
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1159	21	190	295
18	0	0	0	0	0	0	0	0	107	438	45	2	400	91	2	520	400	91
19	0	0	0	1	962	3	39	48	716	355	88	320	62	37	17	97	48	829
20	0	0	0	0	0	0	0	7	683	32	152	343	490	688	241	778	1642	32
21	0	0	0	0	0	0	0	0	1	475	1971	782	81	8	10	58	9	10
22	0	0	0	0	0	0	0	0	0	0	0	0	69	1028	73	821	31	157
23	0	0	0	0	0	1	1445	822	1489	31	779	278	587	327	1903	168	9	4
24	0	0	0	0	0	0	0	0	0	0	0	1	1199	87	3	124	322	1369
25	0	0	0	0	0	0	1	571	123	1	963	626	205	1681	626	810	626	3
26	0	0	0	0	0	0	0	0	0	0	0	0	405	228	41	353	1513	147
27	0	0	0	0	1370	27	354	2	76	1143	1643	60	76	919	1254	29	65	79
28	0	0	0	0	0	0	1	198	476	116	816	253	225	310	16	7	73	2
29	0	0	0	0	0	0	0	0	0	230	542	1200	60	38	5	26	229	1514
30	0	0	0	1	43	1446	6	344	116	579	983	1216	662	77	530	662	106	84
31	0	0	0	0	0	0	0	0	233	1783	396	249	1343	838	490	160	937	1703
32	0	29	1064	311	1447	144	142	142	142	69	1326	567	4	25	4	144	9	4

FIGURE 3.8: Input matrix for LSTM

3.7 Testing

In this section the results of this study will be discussed. Firstly, as we state previously testing data was the 20% of each dataset. With cross validation technique these amounts of data are not always the same observations so we believe that algorithms are efficient enough on unseen data. Then, every dataset in this thesis maps to a different model for a different purpose, so all algorithms performances for the first dataset will be presented in 3.7.1 and for the second in 3.7.2.

3.7.1 First model results

In this section, we comment on and highlight parts of the results after running each algorithm for the first model. As we described in Sections 3.2.1 and 3.1, the point of the first model is to classify text tweets in two classes: Relevant to a disaster or Irrelevant to them. For this purpose we had used dataset Disasters.

Results from all eight ML and DL algorithms are illustrated in Figure 3.10. Of course the winning algorithm is LSTM with $Accuracy = 1$, as it was expected because of it's efficiency with text classification and the advantage of memory through epochs (more information in Section 2.2.2.1). In addition, algorithms like Logistic Regression, Adaboost and Bagged Classifier come next with worst performance in comparison with LSTM but over 90% witch is a good accuracy. Naive Bayes algorithm hasn't achieved as good results as it was expected. It's prediction accuracy is slightly below 90%, but all the above algorithms operate in the same way in both training and testing data. On the other hand, algorithms such as Decision tree and Random forest and DNN have great result on training data but very bad results on testing data which are from 9% to 10% worse. This is a problem called over-fitting and it is very common among algorithms, as described in last paragraph of 2.1. To explain more models over-fitting we'll analyze Figure 3.11. As we can see in plots of algorithms such as: DNN, DT and RF training and testing score have a big deviation because of models over-fitting where algorithms "learn" perfect seen data and couldn't transfer its learning accuracy in unseen data.

Another fact visualized in this figure is that some models such as: NB and LR have acceptable accuracy score in both training and testing data, but their prediction has high uncertainty. To be more specific the probability of chosen label is tend to the unpicked one.

Last step of training procedure was to save all trained algorithms in JSON files for the final step, we'll discuss it in Section 3.9.

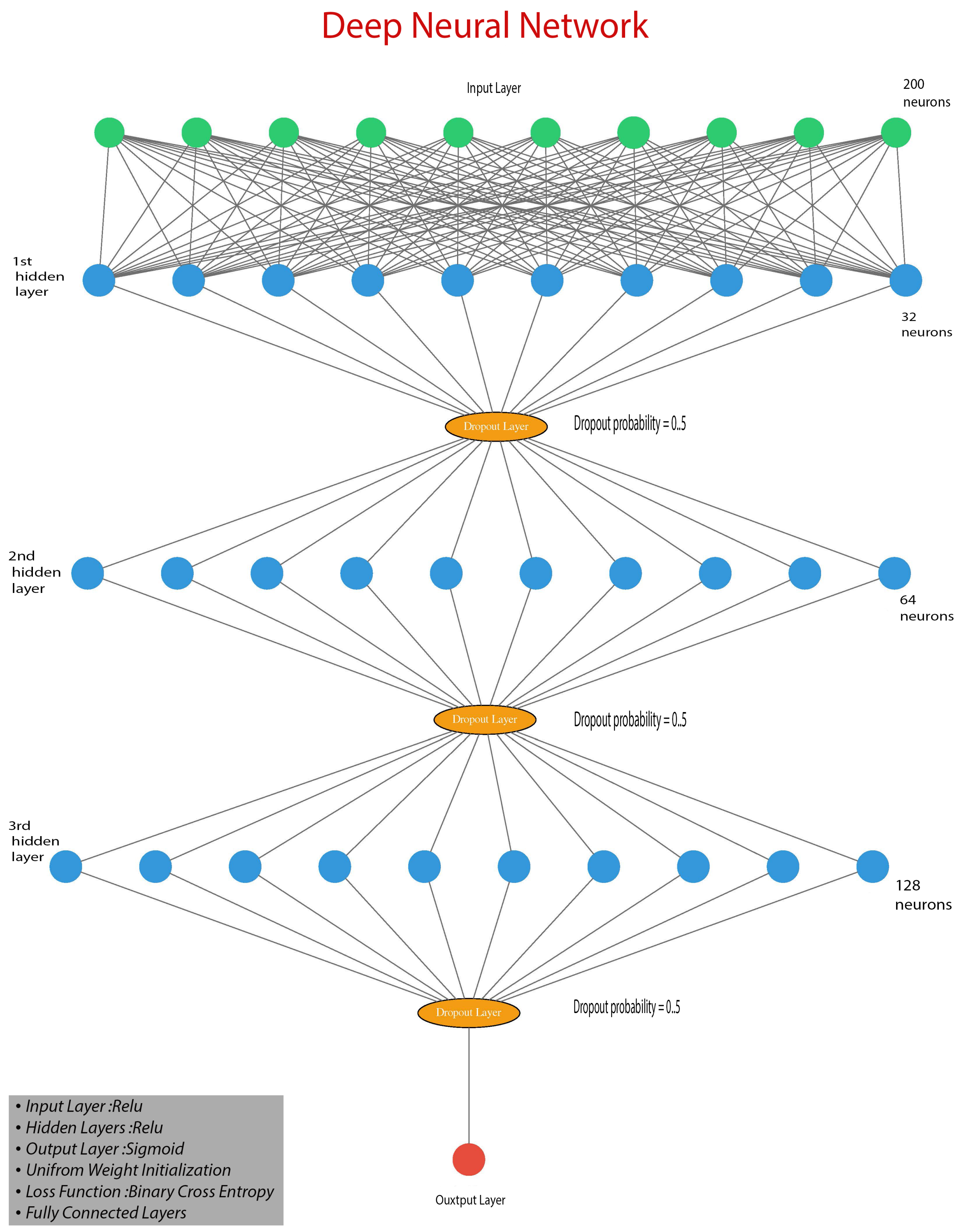
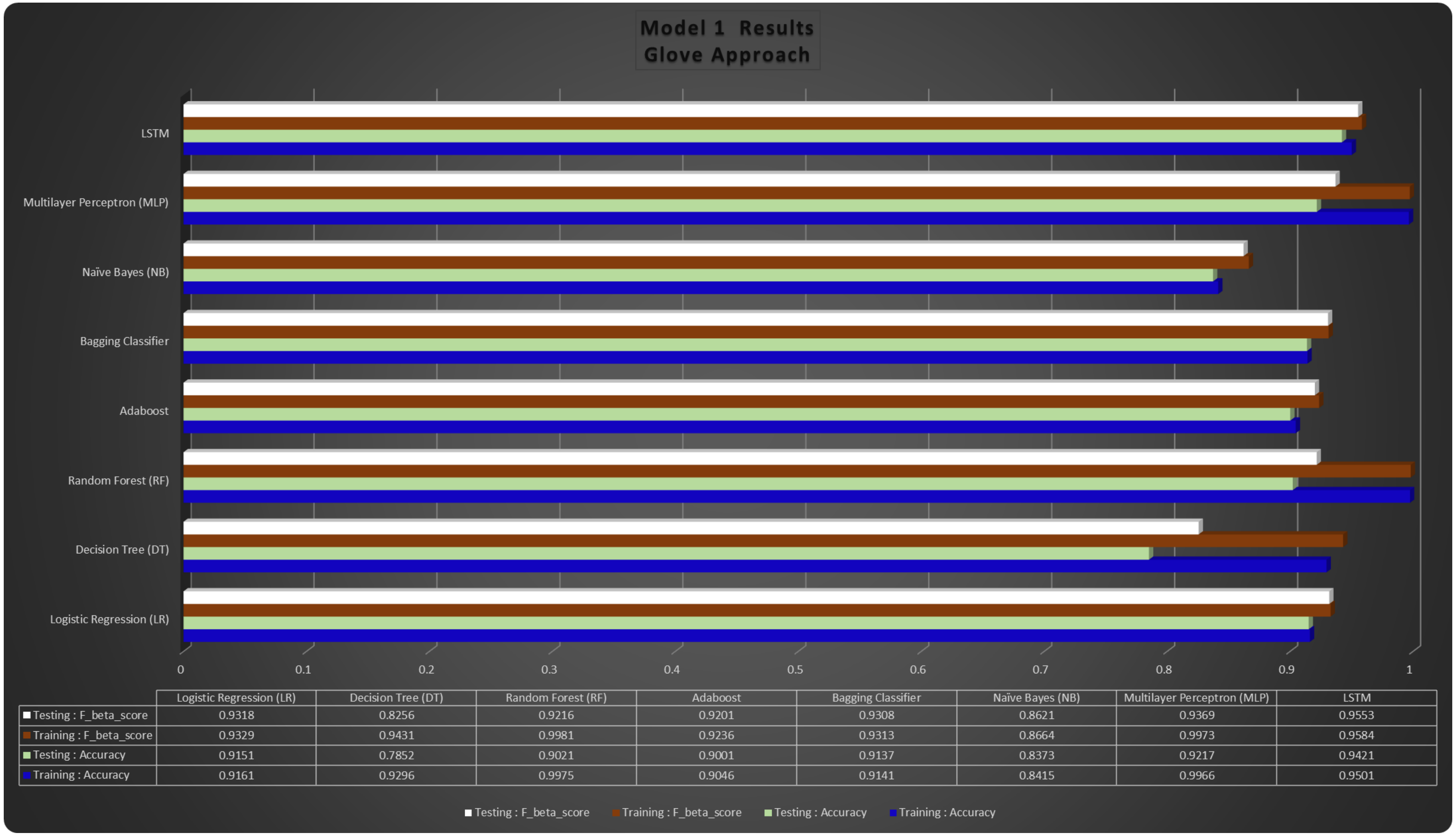


Figure 3.9: Deep Neural Network Architecture



	Logistic Regression (LR)	Decision Tree (DT)	Random Forest (RF)	Adaboost	Bagging Classifier	Naïve Bayes (NB)	Multilayer Perceptron (MLP)	LSTM
⬜ Testing : F_beta_score	0.9318	0.8256	0.9216	0.9201	0.9308	0.8621	0.9369	0.9553
🟫 Training : F_beta_score	0.9329	0.9431	0.9981	0.9236	0.9313	0.8664	0.9973	0.9584
🟩 Testing : Accuracy	0.9151	0.7852	0.9021	0.9001	0.9137	0.8373	0.9217	0.9421
🟦 Training : Accuracy	0.9161	0.9296	0.9975	0.9046	0.9141	0.8415	0.9966	0.9501

Figure 3.10: First model - Algorithm's Results Table

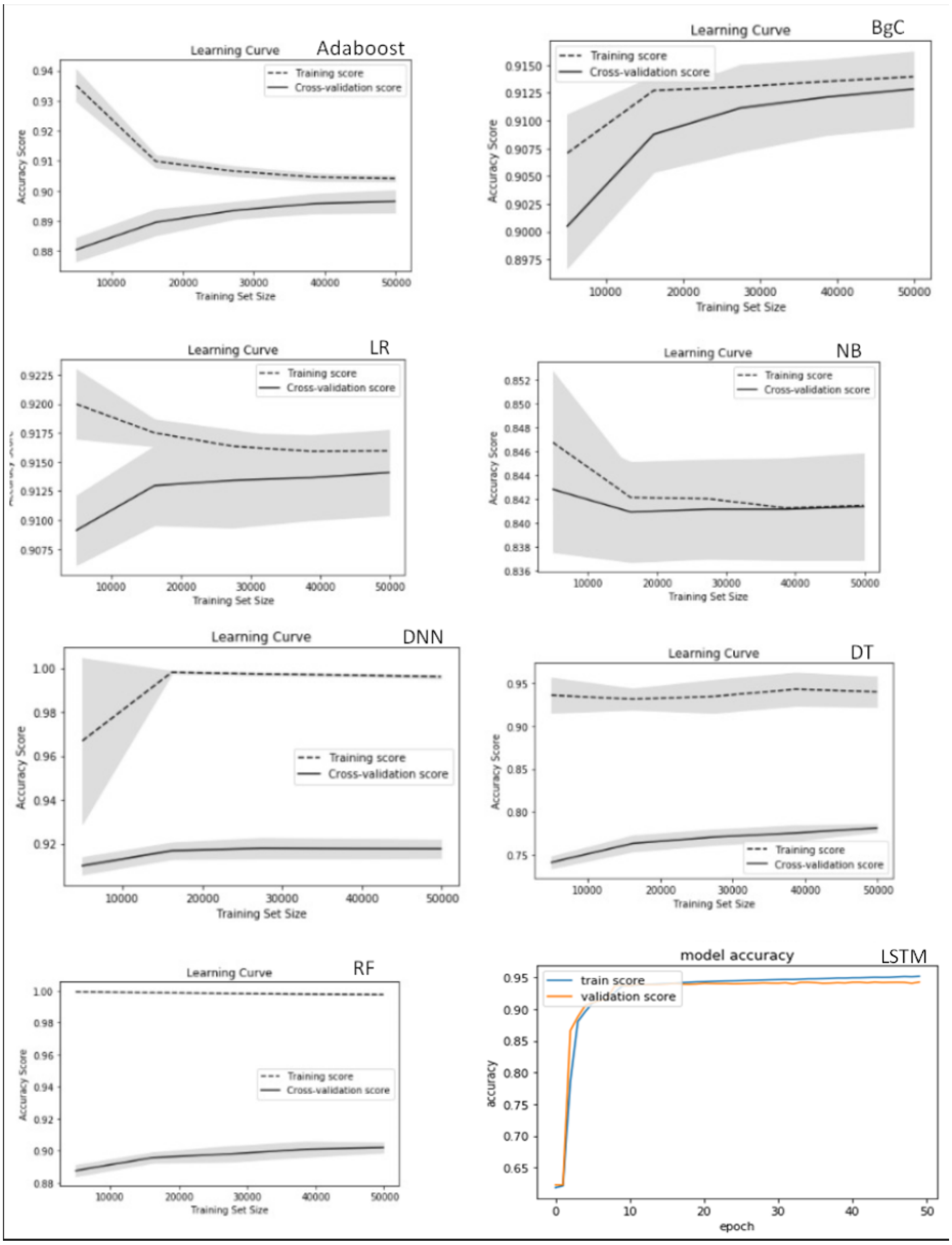


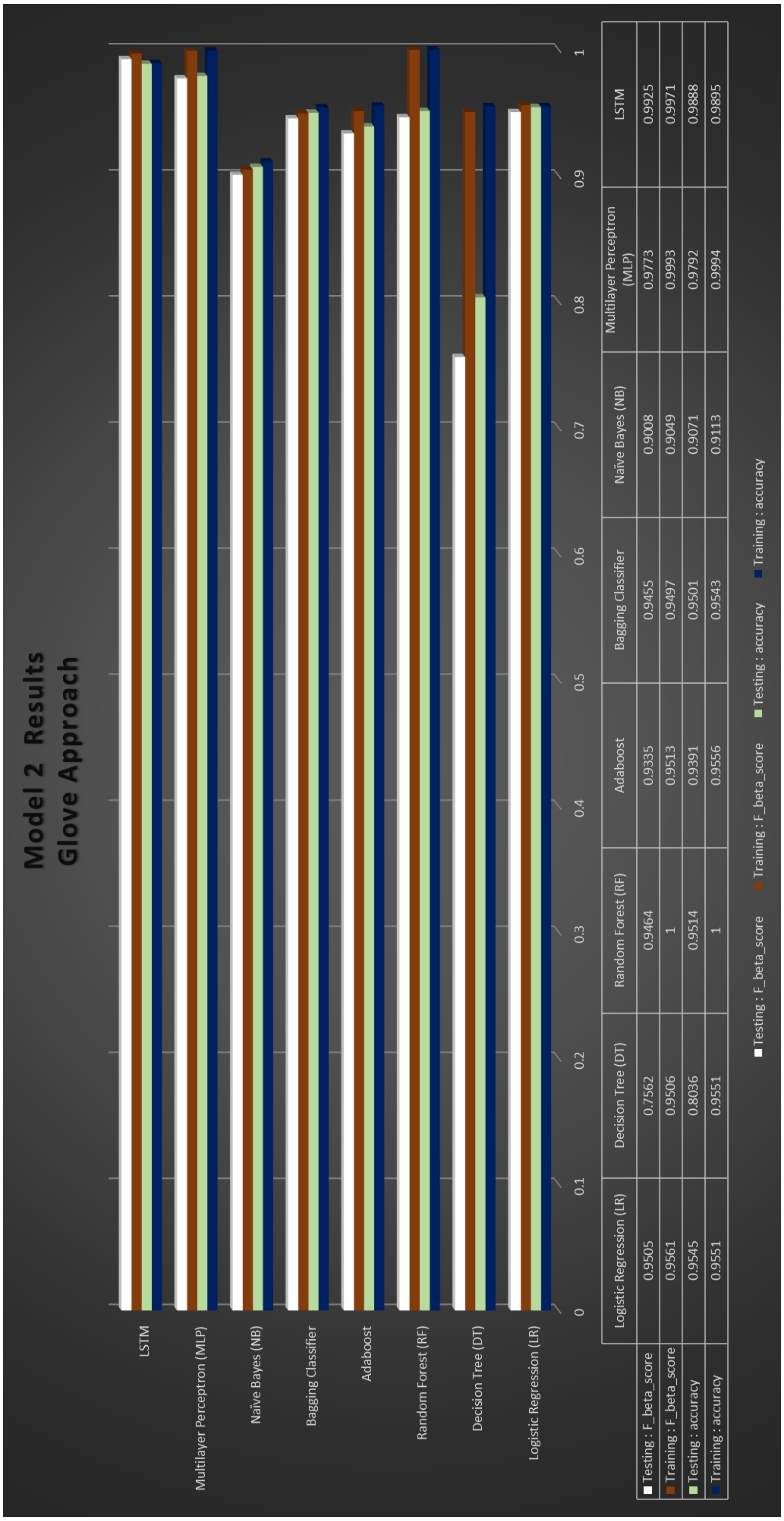
FIGURE 3.11: First model - Algorithm's Results Plots

3.7.2 Second model results

In this section, we comment on and highlight parts of the results after running each algorithm for the second model. As we described in Sections 3.2.2 and 3.1, the point of the second model is to classify text tweets in two classes: Earthquake or Hurricane. For this purpose we had used dataset Earthquake or Hurricane. It is important to state that the second model's use comes after first model. Identification of Earthquake or Hurricane text doesn't applied in every tweet text but only to relatives with disaster.

Results from all eight ML and DL algorithms are presented in Figure 3.12. Again, as it happened in first model, the winning algorithm is LSTM with $Accuracy =$ 0.997, as it was expected because of it's efficiency with text classification and the advantage of memory through epochs (more information in Section 2.2.2.1). LSTM's training accuracy has a deviation of 0.004, so obviously this algorithm doesn't 'suffer' from over-fitting. Moreover, algorithms like Logistic Regression, Adaboost, Naive Bayes, DNN and Bagged Classifier come next with worst performance in comparison with LSTM but over 90% which is a good accuracy. It is very important to note here that above algorithm's prediction accuracy is below LSTM's accuracy, but all of them operate in the same way in both training and testing data. On the other hand, algorithms such as Decision tree and Random forest have great results on training data, but very bad results on testing data which are from 5% to 15% worst. As it happened in first model, these algorithms couldn't handle the over-fitting problem (Section 2.1). As we did in first, we'll represent again algorithms plots to visualize over-fitting and uncertainty problems for second model as well. In Figure 3.13 we can see in plots of algorithms such as: DT and RF training and testing score have a big deviation because of models over-fitting where algorithms "learn" perfect seen perfect seen data and couldn't transfer its learning accuracy in unseen data.

Another fact visualized in this figure is that some models such as: NB and LR have acceptable accuracy score in both training and testing data, but their prediction have high uncertainty. To be more specific the probability of the chosen label tends to the unpicked one. Last step of training procedure, as in first model, was to save all trained algorithms in JSON files for the final step, we'll discuss in Section 3.9.



	Logistic Regression (LR)	Decision Tree (DT)	Random Forest (RF)	Adaboost	Bagging Classifier	Naive Bayes (NB)	Multilayer Perceptron (MLP)	LSTM
⬜ Testing : F_beta_score	0.9505	0.7562	0.9464	0.9335	0.9455	0.9008	0.9773	0.9925
🟫 Training : F_beta_score	0.9561	0.9506	1	0.9513	0.9497	0.9049	0.9993	0.9971
🟩 Testing : accuracy	0.9545	0.8036	0.9514	0.9391	0.9501	0.9071	0.9792	0.9888
🟦 Training : accuracy	0.9551	0.9551	1	0.9556	0.9543	0.9113	0.9994	0.9895

Figure 3.12: Second model - Algorithm's Results Table

3.8 Model Discussion

This chapter is a small synopsis of model's results and some assumptions we have made through paper reading. First the idea of choosing LSTM confirms every belief that this algorithm can be used for text classification and yields great results. Ensemble learning algorithms used such as: BgC and Adaboost did exactly what was expected. Ensemble learning algorithms can boost model performance and that is what happened in this thesis. In contrast DNN algorithm for model one had the same problem as RF and DT had in every model despite the fact that we have used dropout layer(DL) and grid search with 10 fold cross validation to train the algorithms. It is assumed that this problem has occurred because of data. To be more specific, it is believed that bigger amount of training data, for example 2 million data , may had solved this problem. On the other hand datasets with so many observations may cause "high bias" to models. In addition, computational time for such big datasets would have been another rigid problem.

3.9 Hybrid Model

The combination of two or more AI models is called hybrid model. It complements the weaknesses of one model with the advantage of others. Since neural networks is one of the best AI model, most of published hybrid model are neural network based model. Our previous results for first and second model had seemed very promising but even algorithms with good prediction accuracy tend to experience uncertainty problems. consequently our proposal to build the final model for the experiments, is a hybrid model for both models based on the most accurate algorithms each time. As it was explained in detail best algorithms for each classification model were:

- First model:
  - LSTM with 0.951 accuracy score
  - LR with 0.932 accuracy score
  - Adaboost with 0.922 accuracy score
  - Bagged Classifier with 0.931 accuracy score

- Second model:
  - LSTM with 0.989 accuracy score
  - LR with 0.951 accuracy score
  - Adaboost with 0.934 accuracy score
  - Bagged Classifier with 0.945 accuracy score
  - DNN with 0.977 accuracy score

The rest of the algorithms for both the first and the second model hadn't performed that well. Considering that, they are not part of the first hybrid model as we don't want to reduce accuracy of the final hybrid.

The proposed hybrid models in this research are based on the above algorithms for each one. The procedure for each hybrid model could easily be comprehend in Figure 3.14. Every selected classifier in this hybrid model had been fed with data, as happened in each algorithm testing. Next step, algorithms in this model, make their prediction with a probability for each class. We assume that class with the highest probability is a vote for each algorithm. Votes from all classifiers are collected so as to start the voting procedure. In the voting procedure, all votes(note that every classifier's vote is the class with highest probability for this classifier) are countered in order to conclude to the most probable class for this observation. Voting procedure in this hybrid model, is often called Computational Voting in Data Science bibliography.

As we mentioned above, in this thesis two different hybrid models had been build. The first one based on LSTM, LR, Adaboost and Bagged Classifier for disaster relativeness classification. Second hybrid model consists of the following algorithms: LSTM, LR, Adaboost, BgC and DNN. Voting procedure had been applied to both hybrid models, but not exactly in the same way. We could observe that hybrid model 1 is based on four different classifiers, so we have four votes from them. This could lead to a draw in the voting procedure, which happened in tests. In order to solve this problem, the probability of each algorithm has been used for it's prediction. To specify this, we could say that every algorithm is declaring how sure (probability) it is for it's predicted class. In the case of draw we will have four votes with their probabilities. Next we have summed the probabilities with the same vote, and finally we compare the means of each type of vote. We have to note that the types can be two in max, as we are talking about binary classification, therefore binary hybrid models (2 possible classes). To conclude, of course the "winning" class is the one with the highest mean of probabilities.

3.10 System Pipeline

This section, is the final step of the thesis. The initial idea was to build a system which could monitor a natural disaster among two predefined disasters (Earthquake and Hurricane) from real posts (tweets) in the Twitter platform. In order to give "body" in this system and link all the above techniques for the final purpose, a project "pipeline" has been build. The pipeline of this system is visualized in Figure 3.15. This pipeline with some additions could also work to detect natural disasters in real time. In this thesis this option was unavailable due to limitations 5.2, but the procedure would be exactly the same.

To begin with, the very first step of the pipeline model is the tweet/s crawling from Twitter API and the language detection so as to drop non English tweets (Twitter is a global multilingual platform). Next steps are the pre-processing step and data analysis in order to handle the noise of the text, discard possible missing values and give the desired shape we described in Section 3.3. Last step of NLP used in this pipeline system, is the feature extraction with GloVe pre-trained model. As we discussed in Chapter 3.4, from experiments we ended up with global vectors as the best way to represent text for classification. Until this step we have a tweet or a batch of tweets in English language, pre-processed represented as numbers (each word of the tweet) ready to be used from our models.

After modeling text in global vector of numbers, in the next step we have to decide if this tweet is relative or not to a natural disaster. Here comes the use of the first hybrid model which contains all the selected algorithms. These algorithms have been trained in first Dataset 3.2.1 to classify text in this way. The output of this hybrid model will be a useful information about this tweet, which will certify whether the tweet is talking about a disaster or not. Of course non relative tweets will not be part of the following steps as we keep only related. Consequently related with disaster tweets are now input data for the next hybrid model which is based on four five selected (best) algorithms have been trained in second Dataset 3.2.2. This hybrid will categorize them in one of the two classes: Earthquake or Hurricane.

Final part of the pipeline has to do more with geo-location of disaster and time determination, will talk about in Sections 4.2 and 4.3 of the following chapter. In simple words the final step is to detect the place where the identified disaster had happened. The output of the second hybrid model will be a prediction of hurricane or earthquake disaster. The geo-location part is exactly the same for both predictions. The implemented idea to detect the exact place is to search in the text, we believe is talking about a specific disaster, for a city name. As time goes more tweets with the same city name in their text, which of course are predicted as earthquake (or hurricane) text help as to assume that this city is the one that the disaster event had happened (or happening if we have to deal with real time posts). With help from Google Maps API we have finally build a reporting system for the whole purpose. By developing a simple heatmap with the city we have already detected, we visualize the specific "target" of the natural disaster.

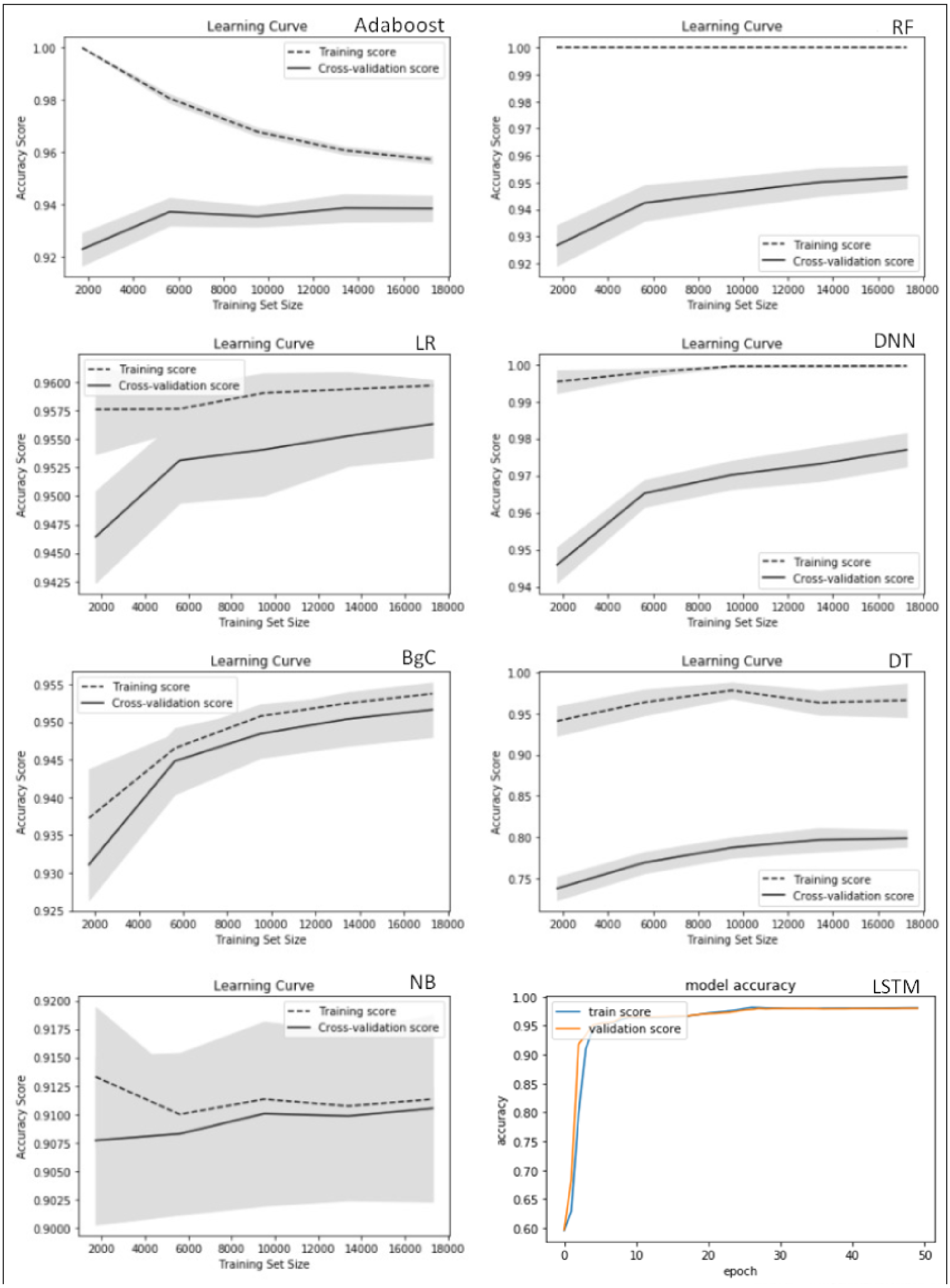


FIGURE 3.13: Second model - Algorithm's Results Plots

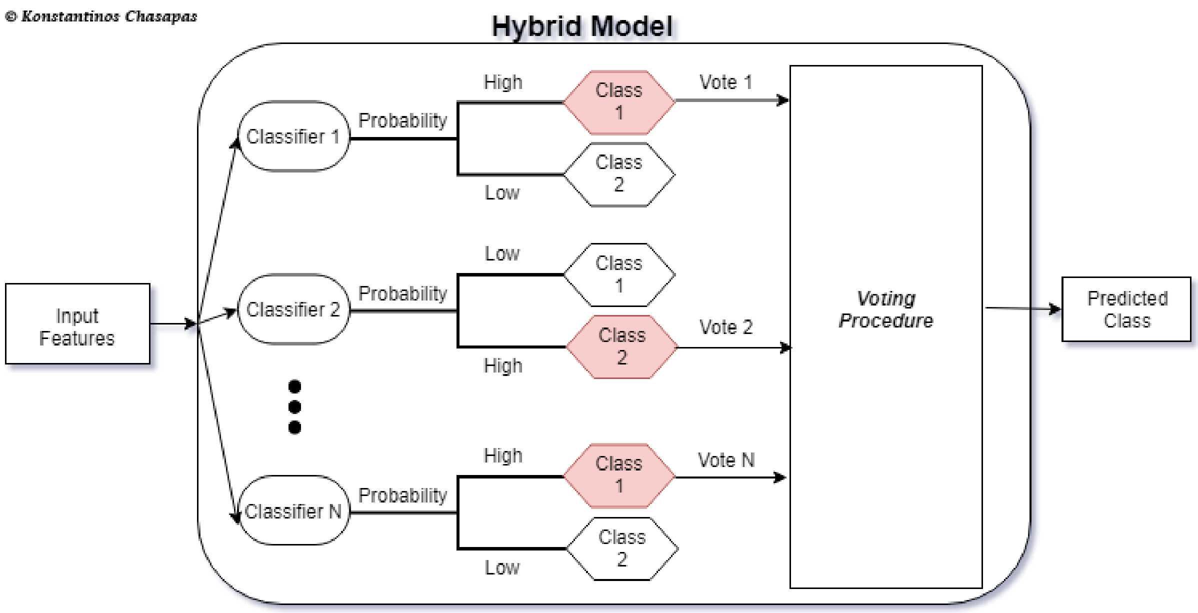


FIGURE 3.14: Hybrid Model Architecture

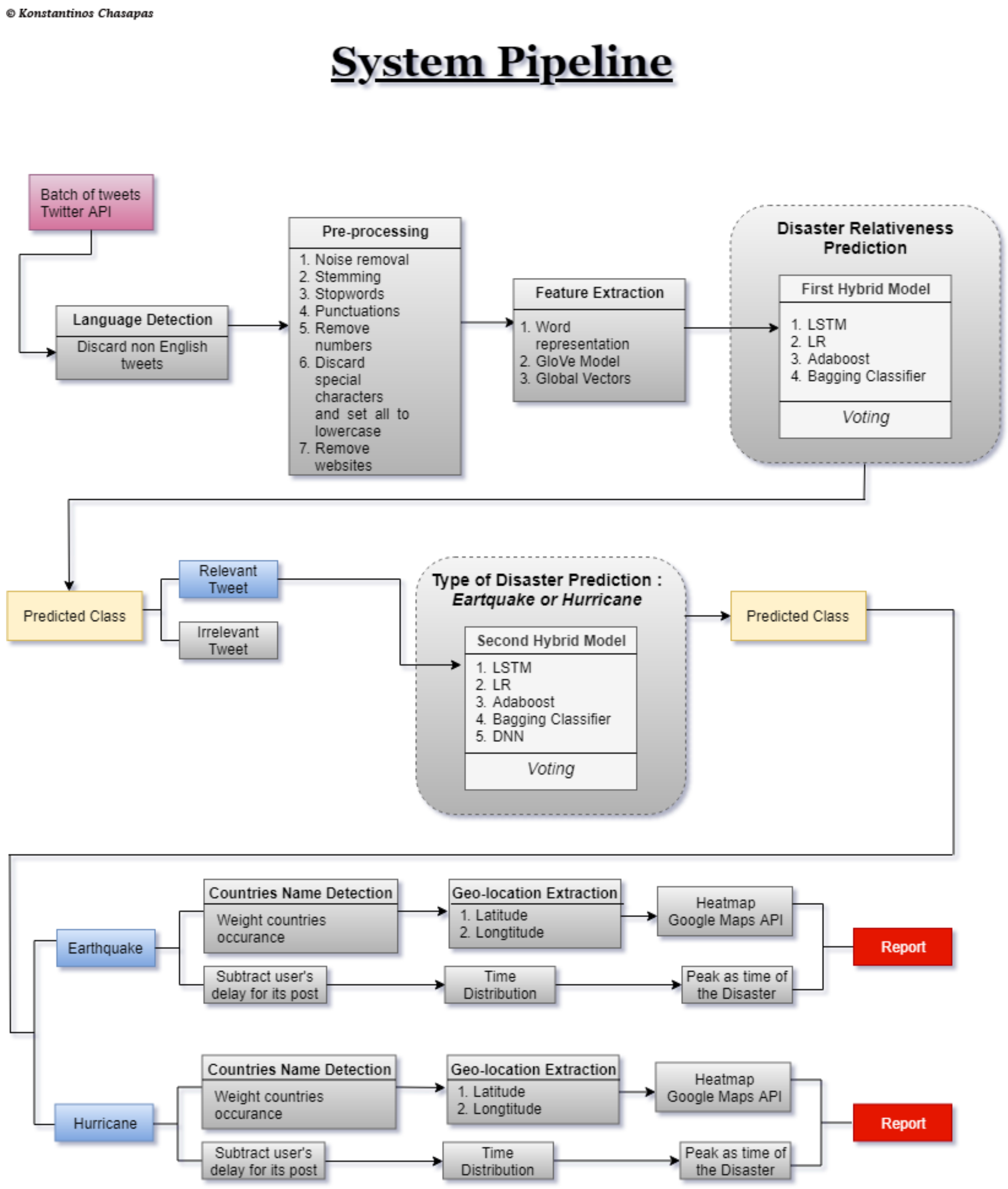


FIGURE 3.15: System Pipeline

Chapter 4

Real Experiments

This chapter discusses the real experiments that will be conducted and the results of these experiments. They will be 3 totally different experiments for our system. First, the datasets that will be used are introduced, then the experimentation plan is laid out. After that a reporting system with country detection and time identification will be discussed for the first and the second experiment. These two steps are only part of the real experiments because of data completeness.

4.1 Data

The data used in the experiments are kindly provided by Arkaitz Zubiaga, assistant professor at University of Warwick. Original data are historical tweets crawled by Twitter API during two natural disasters and one presidential election. These tweets are from all over the world in JSON format and posted during the following disasters:

- April 25, 2015 Nepal earthquake: Original file as provided from Twitter contains multilingual tweets from all over the world on 25th of April, the day Nepal earthquake occurred.

- October 23, 2015 Hurricane Patricia: Original file as provided from Twitter contains multilingual tweets from all over the world on the 23th of October, the day Patricia hurricane had achieved its record peak.

- November 6, 2012 United States presidential election: Original file as provided from Twitter contains multilingual tweets from all over the world on the 6th of November, the day United States presidential election carried out.

From those files we have build three different datasets with English tweets only, after a small pre-processing. These datasets have later been used to implement our experiments. All three datasets have two features: tweet's text and post creation time. As we can see in table 4.1, the Nepal earthquake dataset has 1,612,591 unlabeled observations, Patricia hurricane has 917,519 unlabeled tweets and USA elections has 1,226,475 unlabeled tweets with each creation time.

Table 4.1: Data for Final Experiments

Datasets	**Num. of observations**	**Num. of Features**	**Num. of classes**
Nepal Earthquake	1,612,591	2	Unlabeled
Patricia Hurricane	917,519	2	Unlabeled
USA elections	1,226,475	2	Unlabeled

4.2 Geolocation Prediction

Detection of the place where disasters occurred, are another part of this thesis which is implemented only in final experiments and validated with information from Wikipedia pages. With the assumption that the whole system works in an efficient way, we will finally have a batch of tweets talking about a specific disaster.

Considering limitations from Twitter API 5.2, we didn't have the ability to use latitude and longitude from tweets, so we build a custom model to do the same job in different way. We observed that when people talk about a disaster they used to post also the place this disaster had occurred. Consequently, every single tweet's text related to a disaster (e.g. earthquake) was searched to detect if it has had any country name in its text. For this purpose it was used a file with the 256 most populated counties in world. So every tweet, related to this disaster, containing one of these counties in its text is finally used to detect the exact location.

Geo-location prediction was held only for Nepal and Patricia experiment, in which tweets were related with the disasters are system can identify (earthquake or hurricane). In both experiments (Nepal, Patricia) multiple counties were identified during the search phase through out all observations. Next step was to build a weighting system for each appeared country. Let's consider for example we identified 50 countries in text, from 500,000 related to earthquake tweets. The weight of a country is predefined as the frequency of reference of this country in the whole number of related tweets. For the example we used we will have 50 weights for the 50 countries we have detected. Of course, one or two countries (the affected from the disaster) will have a larger weight than the rest which was the reason we believed in this problem solution.

In addition, in order to find out latitude and longitude of each country, Geopy library was used. Finally we have a number of countries with their location and their weight. For example the above 50 countries, which maps to the most frequent countries referred to earthquake related tweets, will have two coordinates each and a weight. The final step was to build a heatmap with Google MAPS API, to represent every country we had already detected. Heatmap was the perfect map for our case in order to use the weights and for each country and conclude to the most frequent one which we believe is the county the disaster had occurred. In experiments 4.4 and 4.5 we will see and discuss how good this model performed.

4.3 Time Determination

This chapter is our system's final step. As mentioned in Section 4.1, every observation (tweet) of the experimental datasets has the time it had been posted in Twitter databases. For each disaster separately, we build a pool with the related with this tweets and their creation time. We assumed that humans need at least 15 minutes to publish information in Twitter from the time they realize the disaster. Considering that, we subtracted 15 minutes from each time stamp, which maps with an observation. Finally we used python libraries to build a simple time histogram for the pool of related tweets. Time where histogram has peaks, is considered as the time the disaster occurred. In experiment 4.4 and 4.5 we will see and discuss how good this assumption and implementation performed.

4.4 Nepal Experiment

This chapter is the first experiment for our system. The goal is to detect a natural disaster from the first experimental dataset, Nepal 25 April 2015. As we mentioned in Section 4.1, data are tweets from all over the world during the 25th of April. Data are unlabeled so the only way to confirm results is from Wikipedia pages as we did.

To begin with, from the original file we discard non English tweets as our system is trained to handle English words. After this step, the dataset has had 1,005,206 tweets. Next step, was to feed all these observations to our system pipeline where they follow the procedure we described in Section 3.10. The outcomes we are waiting for are:

- A histogram with the number of tweets talking about: Earthquake, Hurricane or the ones totally irrelevant with that type of disaster. Our system is trained to perform for that purpose exactly.
- A heatmap with the name of the city or cities where the earthquake had occurred.
- Last one, a time histogram of all tweets related with the earthquake with have detected.

4.4.1 First Experiment Results

To start with, in Figure 4.1 we could see the results of our system for the first experiment. During the procedure our system concluded that: 957,888 tweets were talking about an earthquake, 32,329 were not related with this event and 4,631 were talking about a hurricane. Here we can observe a deviation from the number of total tweets because the sum of all the tweets above is 994,848 and the input data have had 1,005,806 tweets. This is absolutely logical and due to duplicate observations and/or text contains only “http” websites. These types of tweets had been excluded during the pre-processing step of system. On the other hand, the hurricane bar could be explained with the following reasons: 1. The posts are indeed talking about a hurricane, 2. Our structure is not perfect even with hybrid approach for both models. In summary, our system has predicted 994,848 tweets related to earthquake on the 25th of April, 2015 from 06:00am to 12:00am. We could say that outcome is reasonable as on 25th of April, a big magnitude and deadly earthquake had occurred in Nepal it's said in Wikipedia [4].

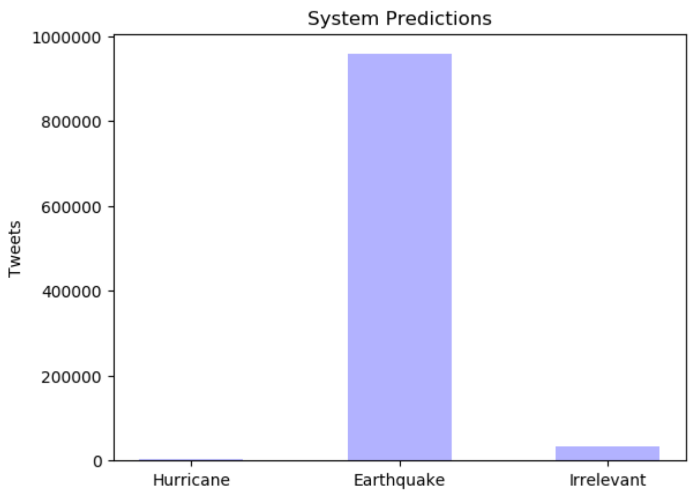


Figure 4.1: Nepal Experiment - System Predictions

Secondly, we need to specify the place the earthquake, our system has detected, had occurred. Using the technique we described in "Geolocation" Section 4.2 alongside with services provided by Google Maps API we have the following result in Figure 4.2. The image at the bottom is a pinned world map and below is a worldwide heatmap. Pins are related with the earthquake tweets our model had identified. We have detected that several counties are talking about earthquake. At this point we have used weighting system to build the heatmap. As we can observe Nepal is the country from which a lot of tweets had been posted about earthquake. We have assumed that the heatmap outcome is the place where the natural phenomenon had happened, which is totally correct as on 25th of April an enormous earthquake had occurred in Nepal. India is also pictured in the heatmap as as an affected country. That means that we had had an number of tweets related with Nepal earthquake, which is accepted as it was an affected area as mentioned in Wikipedia page. But we have to notice that the more "heated" area is the center of the event.

Last task of this experiment had been the time identification. In Figure 4.3, we have the time distribution our system had plot from the pool of related with Nepal earthquake. We can see two significant time peaks: one between 06:00am and 08:00am, another one between 11:00am and 12:00pm and another one between 11:00pm and 12:00am, but the first one is slightly higher than others. We initially assumed in Section 4.3 that the time the event occurred is the time more tweets are talking about this. So we strongly believe that Nepal earthquake had happened minutes after 06:00am, which is correct as it had actually occurred in 06:11am (UTC), as referred in Wikipedia official page. We assume that the other peaks between 11:00am and 12:00pm, 11:00pm and 12:00am have to do with some kind of aftershocks and/or total reports at the end of the day.

To sum up, from the outcomes above the system seemed to perform really well. Prediction of tweets had been accurate enough as it has achieved to detect the disaster event that happened in Nepal among tweets from all over the world. It is important to note that Twitter Api provides tweets only from verified users so maybe irrelevant tweets at that day had been more but are not in our dataset. Another point is that we can't see all the countries "heated" in heatmap. Considering the weighting system our pipeline follows this is acceptable. Finally we couldn't picture the exact time Nepal earthquake had happened because there isn't any method to know after how much time a user will report what they see, but we have restricted time within an hour.

4.5 Patricia Experiment

This chapter is the second experiment for our system. The goal is to detect a natural disaster from the second experimental dataset, Patricia 23 October 2015. As we mentioned in Section 4.1, data are tweets from all over the world during the 23th of October. Data are unlabeled so the only way to confirm results is from Wikipedia pages as we did.

To begin with, as we did in first experiment, from the original file we discarded non English tweets as our system is trained to handle English words. After this step, the dataset has had 544,842 tweets. Next step, was to feed all these observations to our system pipeline where it will follow the procedure we described in Section 3.10. The outcome we are waiting for are:

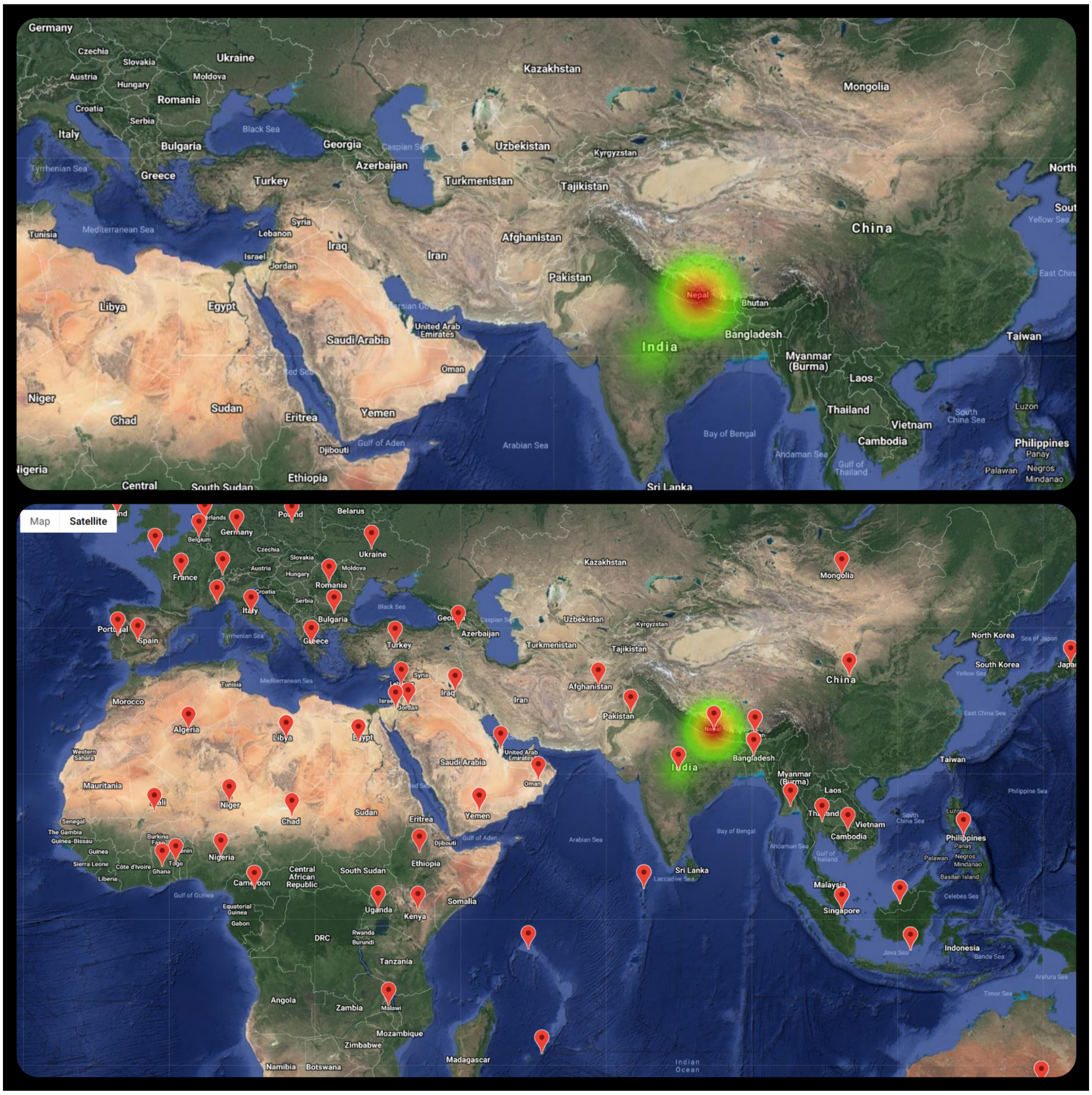


FIGURE 4.2: First Experiment - Heatmap

- A histogram with the number of tweets talking about: Hurricane, Earthquake or the ones that are totally irrelevant with that type of disaster. Our system is trained to perform for that purpose exactly.
- A heatmap with the name of the city or cities where the hurricane had occurred
- Last one, a time histogram of all tweets related with the hurricane event to have detected.

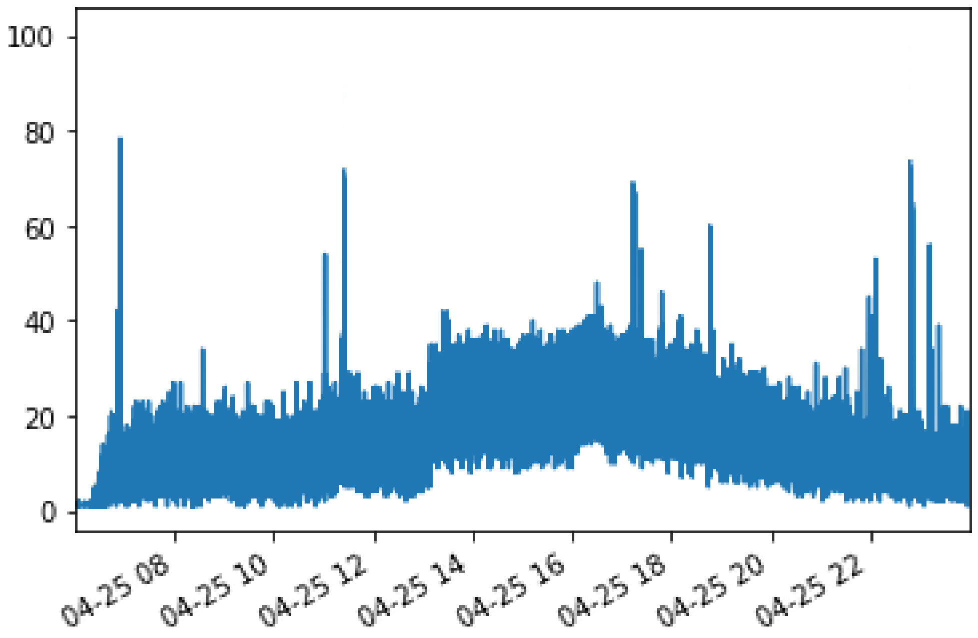


FIGURE 4.3: First Experiment - Time Distribution

4.5.1 Second Experiment Results

To start with, in Figure 4.4 we could see the results of our system for the second experiment. During the procedure our system concluded that: 457,025 tweets were talking about a hurricane, 69,215 were not related with this event and 12,295 were talking about an earthquake. Here we can observe a deviation from the number of total tweets because the sum of all the tweets above is 538,535 and the input data have had 544,842. This is absolutely logical and due to duplicate observations and/or text contains only "http" websites. These type of tweets had been excluded during the pre-processing step of system. On the other hand, the earthquake bar could be explained with the following reasons: 1. The posts are indeed talking about an earthquake, 2. Our system is not perfect even with the hybrid approach for both models. In summary, our system has predicted 544,842 tweets related to hurricane on the 23th of October, 2015 from 12am to 11:59pm. We could say that the outcome is reasonable as on 23th of October, the second-most intense tropical cyclone on record worldwide had formed in Mexico(called Patricia) as said in Wikipedia [3].

Secondly, we need to specify the place the hurricane, our structure has detected, had occurred. Using the technique we described in "Geolocation" Section 4.2 alongside with services provided by Google Maps API we have the following result

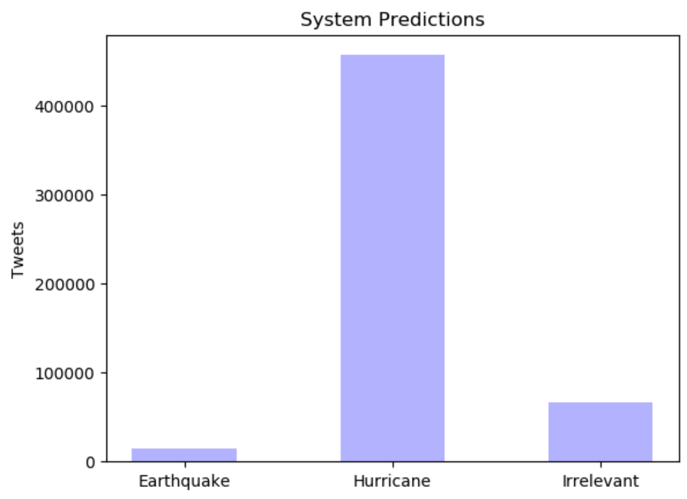


Figure 4.4: Patricia Experiment - System Predictions

in Figure 4.5. The picture at the bottom is a pinned world map and below is a worldwide heatmap. Pins the related with are hurricane tweets our model had identified. We have detected that several counties are talking about hurricane. At this point we have used the weighting system to build the heatmap. As we can observe Mexico is the country from which a lot of tweets posted about hurricane had been. We have assumed that the heatmap outcome is the place where the natural phenomenon had happened, which is totally correct as on 23th of October a noticeable hurricane had occurred in Mexico.

Last task of this experiment had been the time identification. In Figure 4.6, we have the time distribution our system had plot from the pool of related with Patricia hurricane. We can see one significant time peak: between 01:00am and 03:00am. This peak is slightly higher than the second one. We initially assumed in Section 4.3 that the time the event had occurred is the time more tweets are talking about it. So we strongly believe that Patricia hurricane had it's strength record in some time between 02:00am and 03:00am, which is correct as it had actually occurred in 03:00 am (UTC), as referred in Wikipedia official page.

To sum up, from the outcomes above the system seemed to perform really well. Prediction of tweets had been accurate enough as our system has achieved to detect that the disaster event had happened in Mexico among tweets from all over the world. It is important to note that the Twitter Api provides tweets only from verified users so maybe irrelevant tweets at that day had been more but it not in our dataset. Another point is that we can't see all countries "heated" in heatmap. Considering the weighting system our pipeline follows this is acceptable. Finally we couldn't picture the exact time Patricia hurricane had happened because there isn't any method to know after how much time a user will report what they see/attempt, but we have a deviation of no more than an hour.

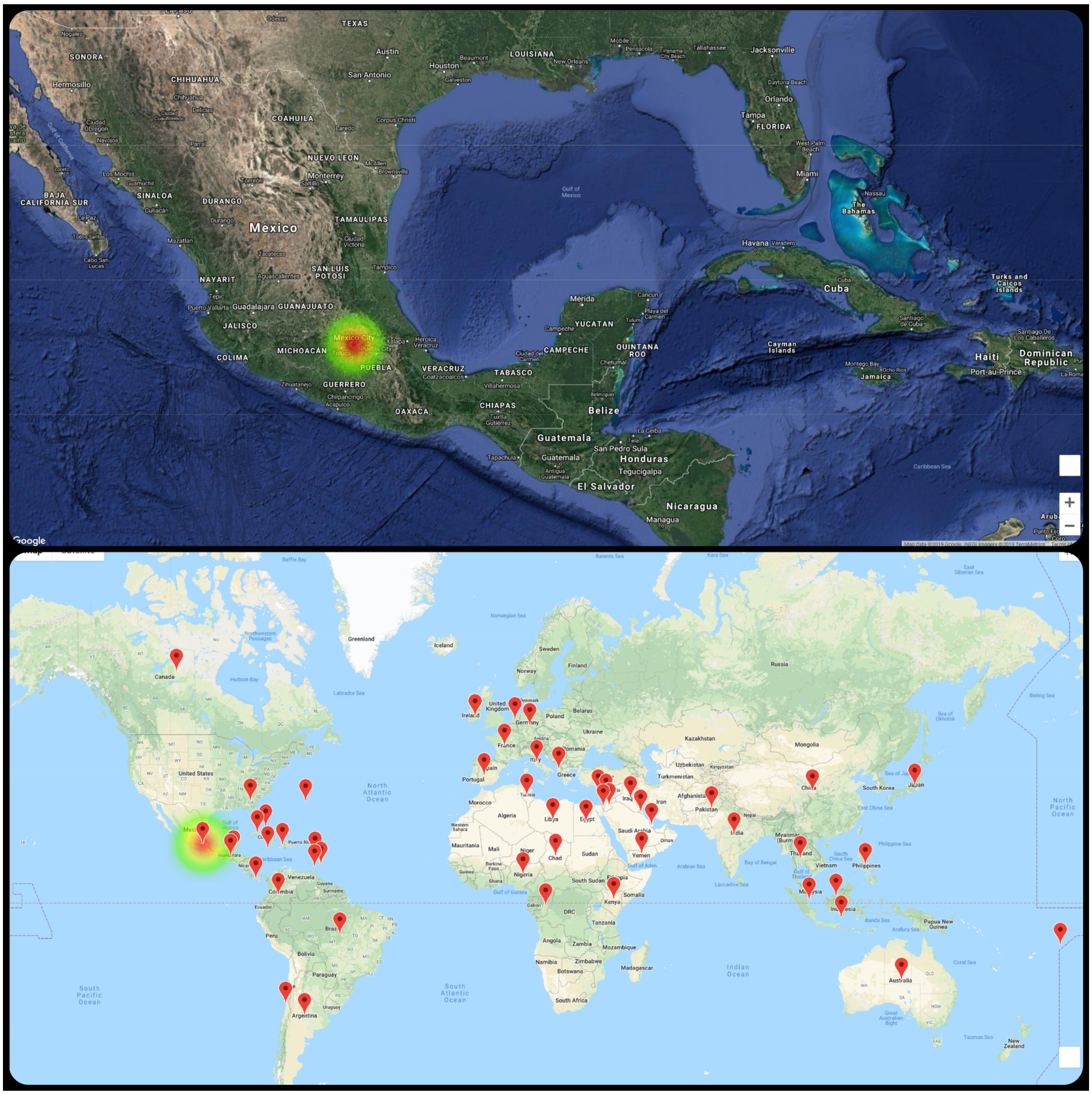


Figure 4.5: Second Experiment - Heatmap

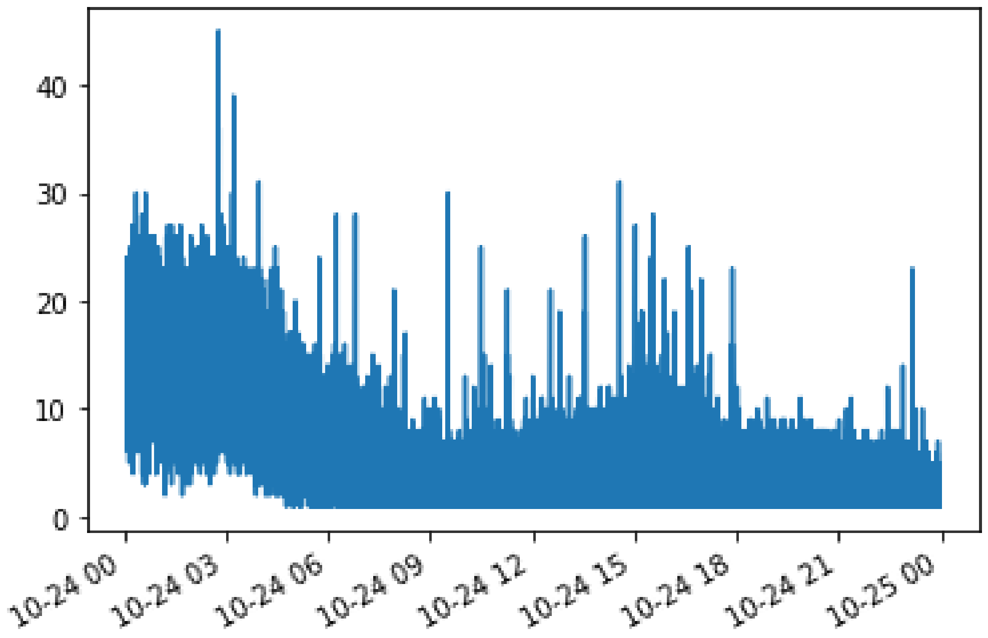


FIGURE 4.6: Second Experiment - Time Distribution

4.6 USA Elections Experiment

This chapter is the third and last experiment for our system. The goal here is not to detect a natural disaster as we did in the two previous. This experiment was conducted in order to test our system in tweets totally irrelevant with earthquake or hurricane by using the third experimental dataset, USA elections 6 November 2012. As we mentioned in Section 4.1, data are tweets from all over the world during the 6th of November where USA presidential election were held. Data are unlabeled so the only way to confirm results is from Wikipedia pages as we did [2].

To begin with, as we did in the two previous experiments, from the original file we discarded non English tweets as our system is trained to handle English words. After this step, the dataset has had 764,749 tweets. Next step, was to feed all these observations to our system pipeline where it will follow the procedure we described in Section 3.10. The outcome we are waiting for are:

- A histogram with the number of tweets talking about: Hurricane, Earthquake or the ones that are totally irrelevant with that types of disaster. Our system is trained to perform for that purpose exactly.

4.6.1 Third Experiment Results

To start with, in Figure 4.7 we could see the results of our system for the third experiment. During the procedure our system concluded that: 18,115 tweets were talking about a hurricane, 33,967 were talking about an earthquake and 702,726 were absolutely not related to earthquake or hurricane. Here we can observe a deviation from the number of total tweets because the sum of all the tweets above is 754,749 and the input data have had 764,475 tweets. This is absolutely expected due to duplicate observations and/or text containing only "http" websites. These types of tweets had been excluded during the pre-processing step of the system. On the other hand, the bars earthquake and hurricane could be explained with the following reasons: 1. The posts are indeed talking about an earthquake or a hurricane, 2. Our system is not perfect even with the hybrid approach for both models. In summary, our system has predicted 702,726 tweets irrelevant to an earthquake and a hurricane on the 6th of November, 2012 from 12:00am to 11:59pm. We could say that the outcome is reasonable and fulfilled the purpose of the third experiment. The 6th of November 2012, was the day that USA presidential election had been conducted, so most of the tweets were talking exactly about this. We have to point out here that our system is not trained to predict an event like elections or any other event except an earthquake and a hurricane. Considering that, we initially expected most of our observations, in this experiment, to be identified as not related to disaster. In this experiment, there is no reason to care about location prediction and time identification, as the results confirm that we mostly have tweets irrelevant to disasters. The number of observations recognized related to an earthquake or a hurricane is small enough to conclude that there is no disaster event. Even if there actually exists an event like the above, there are only a few people talking about this, so it is taken as not reliable enough to predict place and time.

To sum up, from the outcomes above, the system seems to perform really well. Prediction of the type of tweets has been accurate enough, as our system has achieved to detect that most of our observations are not related to an earthquake or a hurricane. We knew that our observations were written on the 6th of November, 2012 the USA presidential election day. As a result most of them were related to elections rather than to a catastrophic event. Eventually, our system results are expected to be mostly irrelevant and they indeed are, as we can observe in Figure 4.7, because our system can identify only earthquakes and hurricanes.

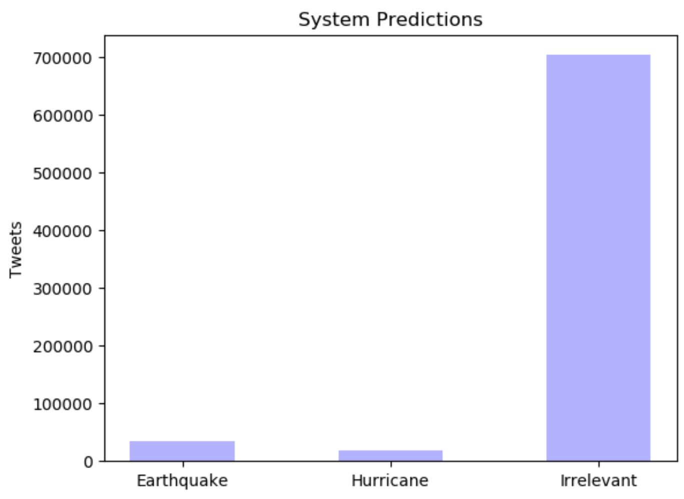


Figure 4.7: USA elections Experiment - System Predictions

Chapter 5

Conclusions and Future Work

This thesis aims to detect a disaster event from social media posts (Twitter in our case) by using NLP techniques alongside ML and DL algorithms. In this chapter, we sum up our findings and contributions, but we also answer the research questions we introduced in Section 1.2. First, we briefly highlight the challenging issues in mining and utilizing Twitter data. Then, we present NLP techniques for text manipulation (pre-processing) in order to achieve an outcome that could be used by NLP models. We approached two totally different techniques for text representation with their advantages and disadvantages and we gave a brief comparison between them. Another task is that we have specified and compared a number of different algorithms from Machine and Deep Learning for text classification purposes. We introduced a hybrid model with a basic voting procedure to improve our system performance. Eventually, we have proposed a different technique for geo-location prediction for our event detection purpose and a time detection. Finally, we discuss the limitations and make suggestions for future work.

5.1 Discussion

This work answered the following research questions:

- How can the detection of an event could be implemented, given raw data from Twitter database?
- In case of an event prediction, how one could monitor the exact place and time of that phenomenon.

- Can we detect such event occurrence in real-time by monitoring tweets?
- What NLP techniques do we need to apply on the data in order to exclude useless information?
- In the state-of-art of the NLP field, could word embedding prevail over Bag of Words model in this scenario?
- Should we use ML or DL algorithms for text classification?

Indeed with this thesis we confirmed that a disaster event (or any other type of event with appropriate changes), could be detected given raw Data from Twitter. This was done by knowledge discovery process consisting of five phases: gathering data, pre-processing data, representing text in numerical vectors, training several algorithms to predict events, combining algorithms for more accurate predictions. Moreover, the second question is also answered in some way, but it is also part of future work. We believe that this system, with some additions, can perform real-time event detection in case we have access to tweets in real time (more in Section 5.2.1)

In addition, many applications like event detection require sufficient and reliable geo-spatial information, which is often hard to obtain. The path that we have followed to solve this problem was to introduce a different technique for place detection. More information and discussion can be found in Section 4.2. From Nepal and Patricia experiments, we confirmed that this spatial model can predict a disaster event. Social media sites (e.g. Twitter) generate massive volumes of user-generated text data which are often noisy, but also very informative. Considering that, we tested a number of NLP techniques, referenced in Section 3.3, in order to keep only the useful text of a tweet. Also, as described in Section 3.4, global vectors (GloVe model) could actually prevail over the Bag of Words model in this scenario, because of its ability to represent context and syntax of all words in a dataset/document.

Another big section of this thesis was the effort to answer where ML or DL will perform better in case of text classification. The outcome says what was expected. DL algorithms could beat ML when we have a lot of observations to train our models, that is why LSTM managed to succeed the best accuracy out of all algorithms used. Another fact we realized after this thesis is that ensemble learning algorithms and Logistic Regression could also attain very good results with a small deviation from Deep Learning algorithms, but they need a lot more time in case of having a lot of data. Finally, the approach of a hybrid model we introduced in our study was an attempt to have better accuracy with the assumption that in observations in which one algorithm makes a wrong prediction, maybe another model makes the correct one. Consequently, the voting system we have built acts as a counter of votes and picks out the most voted prediction. Nevertheless, if most algorithms make wrong predictions, the outcome of the voting will also be wrong. From the experimental results we believe that the hybrid model performed as well as expected.

5.2 Limitations and Future Work

In this section, we identify a number of limitations and potential improvements to our work.

5.2.1 Limitations

Most of the limitations we encountered have to do with the Twitter API. First of all in order to have access to this platform as a developer you need to ask for authorization and state the purpose of use. In our case, after communicating with Twitter services, we managed to get access to Standard API service, which is an entry free package for researchers. We have to thank Twitter API for their kind services which were a fundamental part of this thesis.

On the other hand, the Standard API, which was used in this thesis, has many limitations, such as:

- We can only search tweets within the last 7 days with a 2 hours delay from the time posted on the platform. Considering that, we couldn't monitor a disaster event in real time.
- This package provides only a sample of tweets with a limit of 150 tweets per call and 180 calls per day.
- Filtering tweets was the most significant problem we encountered since searching parameters is connected with logical OR. For example, if we want to

search for tweets containing the word hurricane and posted in the US, the outcome we'll get is a batch of tweets talking about hurricane and another batch which was posted in the US. So we were unable to narrow our search.

Another limitation we encountered is that only 1%-2% of daily posts in Twitter are geo-tagged. Subsequently, our observations were limited and we couldn't train our models properly, especially Deep Learning models which require big data as feed in order to make the right predictions. That's the reason why, we developed another approach for place detection, as described in Section 4.2. A serious limitation in studies like this is computational power. Algorithms and NLP techniques studied in this thesis require powerful systems to manipulate enormous batches of data. Access to more computational power would be very helpful to train our algorithms and test them in bigger experimental datasets. To conclude, the appropriate service for this study is Twitter Enterprise API, which is a non-free commercial package and in order to get access to this service you need to must run a business or be professor at a university. The commercial API provides unlimited access to Twitter databases with many tools for search in real time or specific time.

5.2.2 Future Work

As already discussed in the limitations section, a significant problem of this study has been the lack of full access, using the Enterprise package, in Twitter API. That means that we couldn't try our system in real-time tweets. In case of more computational power and full access to Twitter platform, it would be interesting to try this system for real-time event detection.

Another task that could be accomplished, is the manipulation of the over-fitting problem, that Random Forest and Decision Tree had suffered, for a more complete hybrid model. Finally, it would be interesting to improve our geo-location prediction by using a technique called Named-Entity Recognition in order to reduce the time of computation.

Appendix A

Bias-Viariance

Seema Singh had mentioned that: Whenever we discuss model prediction, it's important to understand prediction errors (bias and variance). There is a trade-off between a model's ability to minimize bias and variance. Gaining a proper understanding of these errors would help us not only to build accurate models but also to avoid the mistake of over-fitting and under-fitting.

- Bias is the difference between the average prediction of our model and the correct value which we are trying to predict. Model with high bias pays very little attention to the training data and oversimplifies the model. It always leads to high error on training and test data.
- Variance is the variability of model prediction for a given data point or a value which tells us spread of our data. Model with high variance pays a lot of attention to training data and does not generalize on the data which it hasn't seen before. As a result, such models perform very well on training data but has high error rates on test data.

In the above diagram, center of the target is a model that perfectly predicts correct values. As we move away from the bulls-eye our predictions become get worse and worse. We can repeat our process of model building to get separate hits on the target.

In supervised learning, under-fitting happens when a model unable to capture the underlying pattern of the data. These models usually have high bias and low

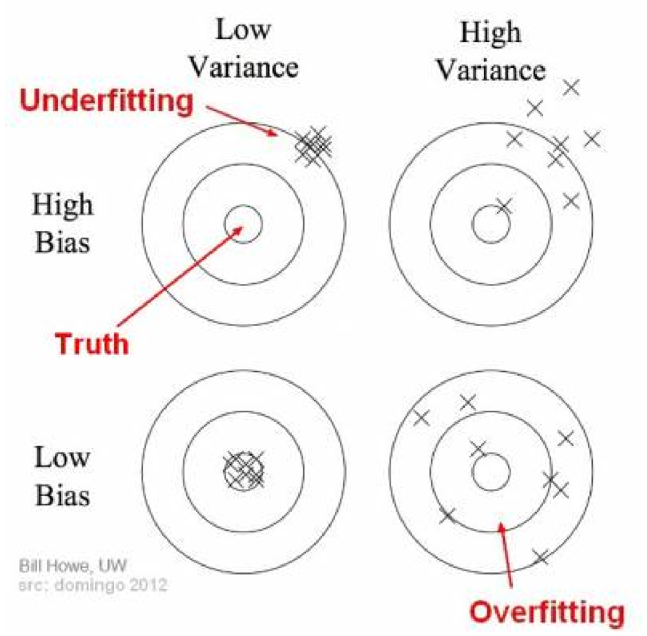


FIGURE A.1: Bias and variance using bulls-eye diagram

variance. It happens when we have very less amount of data to build an accurate model or when we try to build a linear model with a nonlinear data.

In supervised learning, over-fitting happens when our model captures the noise along with the underlying pattern in data. It happens when we train our model a lot over noisy dataset. These models have low bias and high variance. These models are very complex like Decision trees which are prone to over-fitting.[42]

Appendix B

Twitter API

Official Twitter for developers platform defines Twitter API as :Twitter is what's happening in the world and what people are talking about right now. You can access Twitter via the web or your mobile device. To share information on Twitter as widely as possible, we also provide companies, developers, and users with programmatic access to Twitter data through our APIs (application programming interfaces). This article explains what Twitter's APIs are, what information is made available through them, and some of the protections Twitter has in place for their use.

At a high level, APIs are the way computer programs "talk" to each other so that they can request and deliver information. This is done by allowing a software application to call what's known as an endpoint: an address that corresponds with a specific type of information we provide (endpoints are generally unique like phone numbers). Twitter allows access to parts of our service via APIs to allow people to build software that integrates with Twitter, like a solution that helps a company respond to customer feedback on Twitter.

Twitter data is unique from data shared by most other social platforms because it reflects information that users choose to share publicly. Our API platform provides broad access to public Twitter data that users have chosen to share with the world. We also support APIs that allow users to manage their own non-public Twitter information (e.g., Direct Messages) and provide this information to developers whom they have authorized to do so.

Appendix C

Cosine Similarity

Richard Wicentowski mentioned that: The GloVe algorithm is designed to produce similar vectors for semantically similar words. We can test how well the GloVe vectors capture semantic similarity with cosine similarity. In the figure below , vectors u and v are separated by an angle of θ :

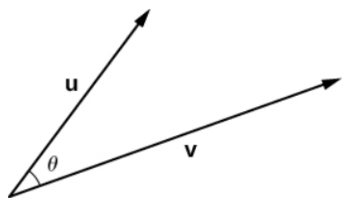


FIGURE C.1: Cosine Similarity Vectors

Two vectors that are separated by a small angle are more similar than two vectors separated by a large angle. Rather than find the actual angle in between the two vectors, we will find the cosine of the angle between the two vectors. Notice in the plot below that when the angle between two vectors is 0, the cosine of the angle between the two vectors is 1. As the angle between the two vectors increases, the cosine of the angle decreases. When the two vectors are as far apart as possible (180 degrees rotated from one another), the cosine of the angle between them is 1 In order to compute the angle between two vectors, you can use the following formula: $\cos(u, v) = \frac{u}{|u||v|}$ where $u * v$ is the dot product between the two vectors, and $|u|$ is the length of vector u, which you can calculate by taking the square root of the sum of the squares of the elements in the vector: $|u| = \sqrt{\sum_i u_i^2}.$[48]

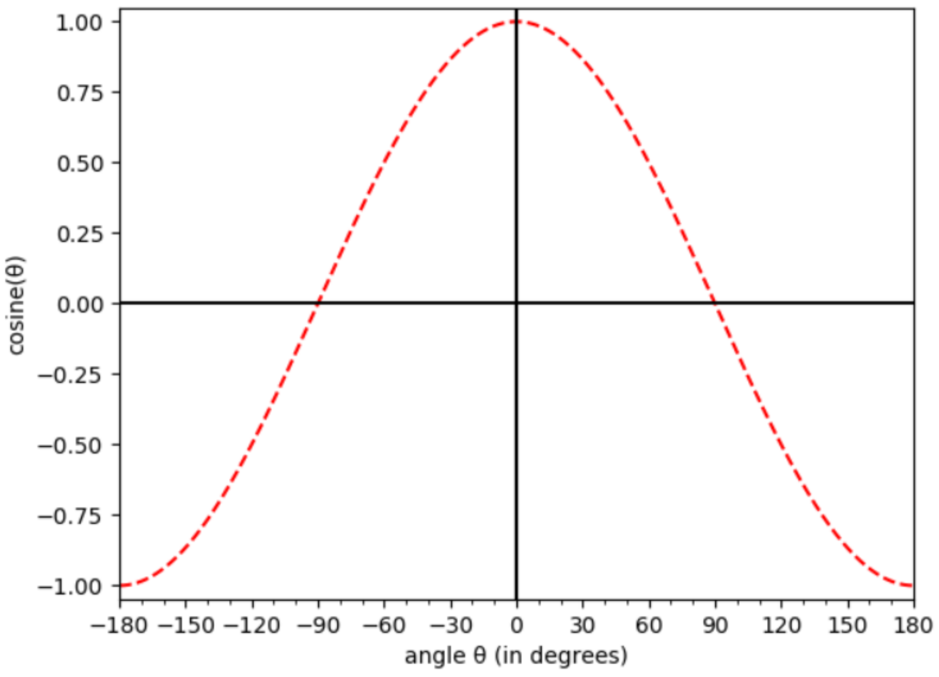


Figure C.2: Angle - Cosine Diagram

Appendix D

Bidirectional LSTM

Intan Nurma Yulita describes Bidirectional LSTM as: Bi-LSTM is a combination of Long Short-Term Memory (LSTM) and Bi-directional Recurrent Networks (Bi-RNN). Recurrent Neural Network (RNN) which is a special development of Artificial Neural Networks (ANN) to process sequences and time series data. RNN has the advantage to encode dependencies between inputs. However, for long data sequences, RNN causes exploding and vanishing state against its gradient. After that, Long Short Term Memory (LSTM) is created to overcome long-term problems of the RNN. LSTM consists of some gates. For input layer, there is input gate. As for the output layer, there is forget gate and output gate 13. However, both LSTM and RNN can only get information from the previous context so that further improvements are made using the Bidirectional Recurrent Neural Network (Bi-RNN). Bi-RNN can handle two information both from the front and back 14. The combination of Bi-RNN combined with LSTM produces Bi-LSTM, as illustrated in D.1. So the advantages of LSTM in the form of storage in cell memory

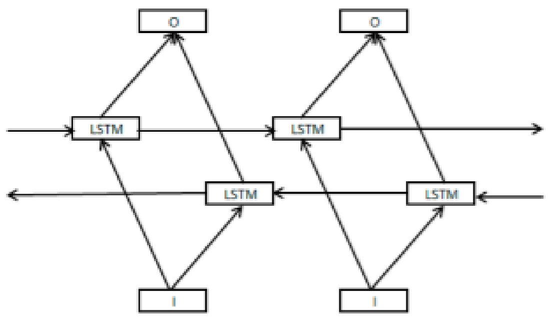


Fig. 2. Bi-LSTM.

FIGURE D.1: Bidirectional LSTM

and Bi-RNN with access information from context before and after make Bi-LSTM excel 15. It causes the Bi-LSTM to have the advantage of LSTM with feedback for the next layer. However, on the other hand, Bi-LSTM can also handle data with dependence on long range.[52]

Appendix E

Max Pooling

Official page of Computer Science Wiki describes Max Pooling/Pooling as follows : Max pooling is a sample-based discretization process. The objective is to down-sample an input representation (image, hidden-layer output matrix, etc.), reducing its dimensionality and allowing for assumptions to be made about features contained in the sub-regions binned. This is done to in part to help over-fitting by providing an abstracted form of the representation. As well, it reduces the computational cost by reducing the number of parameters to learn and provides basic translation in-variance to the internal representation. Max pooling is done by applying a max filter to (usually) non-overlapping sub regions of the initial representation. Let's say we have a $4x4$ matrix representing our initial input. Let's say, as well, that we have a $2x2$ filter that we'll run over our input. We'll have a stride of 2 (meaning the (d_x, d_y) for stepping over our input will be $(2, 2)$) and won't overlap regions. For each of the regions represented by the filter, we will take the max of that region and create a new, output matrix where each element is the max of a region in the original input.[1]

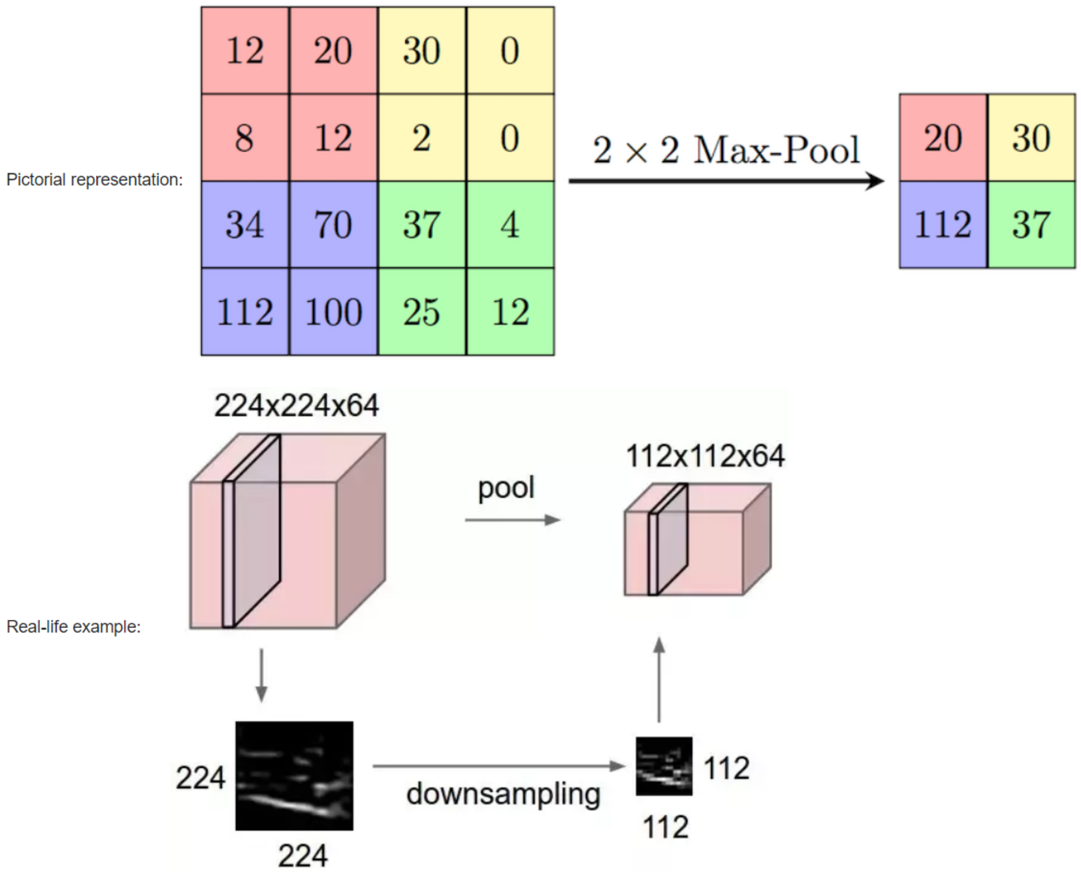


FIGURE E.1: Max Pooling Example

Bibliography

[1] Max-pooling / pooling. URL https://computersciencewiki.org/index.php/Max-pooling_/_Pooling.

[2] 2012 United States presidential election. Wikipedia. URL https://en.wikipedia.org/wiki/2012_United_States_presidential_election.

[3] Meteorological history of hurricane patricia, Jul 2018. URL https://en.wikipedia.org/wiki/Meteorological_history_of_Hurricane_Patricia#Peak_strength.

[4] April 2015 nepal earthquake, Mar 2019. URL https://en.wikipedia.org/wiki/April_2015_Nepal_earthquake.

[5] Dropout (neural networks), Jan 2019. URL https://en.m.wikipedia.org/wiki/Dropout_(neural_networks).

[6] N. Alsaedi. Event Identification in Social Media using Classification-Clustering Framework Cardiff University. 2017.

[7] F. Atefeh and W. Khreich. A Survey Of Techniques For Event Detection In Twitter. 0(0), 2013.

[8] Banerjee. An Introduction to Recurrent Neural Networks – Explore Artificial Intelligence – Medium. May 2018. URL https://medium.com/explore-artificial-intelligence/an-introduction-to-recurrent-neural-networks-72c97bf0912.

[9] S. Bansal. A Comprehensive Guide to Understand and Implement Text Classification in Python. Analytics Vidhya, Feb 2019.

[10] A. Bilal. Artificial Neural Networks and Deep Learning – Becoming Human: Artificial Intelligence Magazine, Jan 2018. URL https://becominghuman.ai/artificial-neural-networks-and-deep-learning-a3c9136f2137.

[11] J. Brownlee. Supervised and unsupervised machine learning algorithms. Machine Learning Mastery, 2016. URL https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/.

[12] J. Brownlee. Logistic Regression for Machine Learning. Machine Learning Mastery, Sep 2016. URL https://machinelearningmastery.com/logistic-regression-for-machine-learning/.

[13] J. Brownlee. What are word embeddings for text?, Nov 2017. URL https://machinelearningmastery.com/what-are-word-embeddings/.

[14] J. Brownlee. A gentle introduction to k-fold cross-validation, May 2018. URL https://machinelearningmastery.com/k-fold-cross-validation/.

[15] J. Brownlee. A gentle introduction to the bag-of-words model. Machine Learning Mastery, Feb 2019. URL https://machinelearningmastery.com/gentle-introduction-bag-words-model/.

[16] C.-P. Cheng. Applying the Naive Bayes classifier with kernel density estimation to the prediction of protein- protein interaction sites. Bioinformatics. Aug, 126(15):1841–8, 2010.

[17] DeFilippi. Boosting, Bagging, Stacking - Ensemble Methods with sklearn and mlen. medium.com, Aug 2018.

[18] J. D'Souza and J. D'Souza. An introduction to bag-of-words in nlp – greyatom – medium. medium.com, Apr 2018. URL https://medium.com/greyatom/an-introduction-to-bag-of-words-in-nlp-ac967d43b428.

[19] J. Frankenfield. Deep learning, Dec 2018. URL https://www.investopedia.com/terms/d/deep-learning.asp.

[20] T. Hastie. Trees, Bagging, Random Forests and Boosting Two-class Classification. December 2004. URL http://jessica2.msri.org/attachments/10778/10778-boost.pdf.

[21] J. Jordan. Hyperparameter tuning for machine learning models., Dec 2018. URL https://www.jeremyjordan.me/hyperparameter-tuning/.

[22] B. Krenker. Introduction to the artificial neural networks, artificial neural networks - methodological advances and biomedical applications. Prof. Kenji Suzuki (Ed.), 2011.

[23] S. Kumar, H. Liu, S. Mehta, and L. V. Subramaniam. Exploring a scalable solution to identifying events in noisy Twitter streams. 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM), pages 496–499, 2015.

[24] K. Kurita. Paper dissected: "glove: Global vectors for word representation" explained, May 2018. URL http://mlexplained.com/2018/04/29/paper-dissected-glove-global-vectors-for-word-representation-explained/.

[25] C. Mackay. "GloVe: Global Vectors for Word Representation ". Notes and Queries, s5-IV(96):346, 1875.

[26] C. McCormick. Adaboost tutorial. URL http://mccormickml.com/2013/12/13/adaboost-tutorial/.

[27] V. K. Neppalli, C. Caragea, and D. Caragea. Deep Neural Networks versus Naive Bayes Classifiers for Identifying Informative Tweets during Disasters. 15th International Conference on Information Systems for Crisis Response and Management (ISCRAM 2018), (May):677–686, 2018. URL http://files/3598/VenkataKishoreNeppallietal.-2018-DeepNeuralNetworksversusNaiveBayesClassifier.pdf{%}0Ahttp://idl.iscram.org/files/venkatakishoreneppalli/2018/1589{_}VenkataKishoreNeppalli{_}etal2018.pdf.

[28] K. Nordhausen. Ensemble Methods: Foundations and Algorithms by Zhi-Hua Zhou, volume 81. 2013. URL https://EconPapers.repec.org/RePEc:bla:istatr:v:81:y:2013:i:3:p:470-470.

[29] C. Olah. Understanding lstm networks, Aug 2015. URL http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[30] D. Oldenhave, S. Hoppenbrouwers, and T. V. D. Weide. PDF hosted at the Radboud Repository of the Radboud University Nijmegen This full text is a preprint version which may differ from the publisher ' s version . Gamification to Support the Run Time Planning Process in Adaptive Case Management. (June 2015), 2014.

[31] O. Ozdikis. Locality-adapted Kernel Densities for Tweet Localization. pages 1149–1152, 2018.

[32] J. Pennington. URL https://nlp.stanford.edu/projects/glove/.

[33] A. Pérez, P. Larrañaga, and I. Inza. Bayesian classifiers based on kernel density estimation: Flexible classifiers. International Journal of Approximate Reasoning, 50(2):341–362, 2009. URL http://dx.doi.org/10.1016/j.ijar.2008.08.008.

[34] T. Sakaki, M. Okazaki, and Y. Matsuo. Earthquake Shakes Twitter Users: Real-time Event Detection by Social Sensors. Proceedings of the 19th International Conference on World Wide Web, pages 851–860, 2010. URL http://doi.acm.org/10.1145/1772690.1772777.

[35] P. Sayak. Hyperparameter optimization in machine learning. DataCamp Community, Aug 2018. URL https://www.datacamp.com/community/tutorials/parameter-optimization-machine-learning-models.

[36] H. Sayyad, M. Hurst, and A. Maykov. Event Detection and Tracking in Social Streams. International AAAI Conference on Web and Social Media, pages 1–4, 2009. URL papers3://publication/uuid/48355112-0D9C-4196-8CC5-109AF70EAB71.

[37] R. E. Schapire. Explaining adaboost. Empirical Inference: Festschrift in Honor of Vladimir N. Vapnik, pages 37–52, 2013.

[38] M. Schonlau. Background on spatial-temporal modeling. The Stata Journal, 5(3):330–354, 2005.

[39] F. Schoonjans. Logistic regression, Nov 2018. URL https://www.medcalc.org/manual/logistic_regression.php.

[40] Sciforce. Word vectors in natural language processing: Global vectors (glove). medium.com, Aug 2018.

[41] A. Singh. A comprehensive guide to ensemble learning (with python codes). Analytics Vidhya, Jun 2018. URL https://www.analyticsvidhya.com/blog/2018/06/comprehensive-guide-for-ensemble-models/.

[42] S. Singh. Understanding the Bias-Variance Tradeoff. Towards Data Science, May 2018. URL https://towardsdatascience.com/understanding-the-bias-variance-tradeoff-165e6942b229.

[43] P. Srivastava. Essentials of deep learning : Introduction to long short term memory, Dec 2017. URL https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/.

[44] M. Sunasra. Performance metrics for classification problems in machine learning. medium.com, Nov 2017.

[45] A. Weiler, M. Grossniklaus, and M. H. Scholl. Survey and Experimental Analysis of Event Detection Techniques for Twitter. 60(April 2016):329–346, 2017.

[46] J. Weng, Y. Yao, E. Leonardi, F. Lee, J. Weng, Y. Yao, E. Leonardi, and B.-s. Lee. Event Detection in Jianshu. 2011.

[47] B. Whitaker and B. Whitaker. [emnlp] what is glove? part iii – towards data science, May 2018. URL https://towardsdatascience.com/emnlp-what-is-glove-part-iii-c6090bed114.

[48] R. Wicentowski. Lab 03. Computer Science. URL https://www.cs.swarthmore.edu/~richardw/classes/cs65/f18/lab03.html.

[49] Wikipedia. Bayes classifier. SpringerReference, 2006. URL https://en.wikipedia.org/wiki/Naive_Bayes_classifier.

[50] S. Yan. Understanding LSTM and its diagrams – ML Review – Medium. medium.com, Mar 2016. URL https://medium.com/mlreview/understanding-lstm-and-its-diagrams-37e2f46f1714.

[51] V. Yordanov. Introduction to Natural Language Processing for Text. Towards Data Science, Nov 2018. URL https://towardsdatascience.com/introduction-to-natural-language-processing-for-text-df845750fb63.

[52] I. N. Yulita, M. I. Fanany, and A. M. Arymuthy. Bi-directional Long Short-Term Memory using Quantized data of Deep Belief Networks for Sleep Stage Classification. Procedia Computer Science, 116:530–538, 2017. URL https://doi.org/10.1016/j.procs.2017.10.042.