Master’s Thesis
submitted in partial fulfilment of the
requirements for the course “Applied Computer Science”

A Crowdsourced Distributed Stream Processing Approach to Classify SMS Communications in Crises

Ramaninder Singh Jhajj

Institute of Computer Science

Bachelor’s and Master’s Theses
of the Center for Computational Sciences
at the Georg-August-Universität Göttingen

Georg-August-Universität Göttingen  
Institute of Computer Science  
Goldschmidtstraße 7  
37077 Göttingen  
Germany  

tel: +49 (551) 39-172000  
fax: +49 (551) 39-14403  
off: office@informatik.uni-goettingen.de  
web: www.informatik.uni-goettingen.de  

First Supervisor: Prof. Dr. Xiaoming Fu  
Second Supervisor: Dr. Mayutan Arumaithurai
I hereby declare that I have written this thesis independently without any help from others and without the use of documents or aids other than those stated. I have mentioned all used sources and cited them correctly according to established academic citation rules.

Abstract

Use of text messages is increasing during humanitarian disasters to exchange accounts of ongoing events, to request for help or vital requirements by victims, responders, aid organizations and others. One of the first successful events where text messages saved the life of many people is the Haiti Earthquake in January 2010 and the implementation of the project Ushahidi in which people can send text messages.

Due to the large amount of text messages arriving in such scenarios, it is very hard for volunteers, aid organizations to manually handle the volume of messages which can reach to thousands of messages per hour during large scale events. Current information systems also struggle to make sense of this vast body of knowledge due to the limitation in terms of accuracy and scalability of processing being two major drawbacks. If the accuracy and processing can be addressed and solutions can be found, text messages can be mined in near real-time to offer disaster responders the material which they need for decision making and situational awareness.

This thesis provides the step forward in this direction. After the detailed discussion about the humanitarian disasters against which a prototype system is proposed which focuses on the accurate distribution of the text messages into separate classes so that the people and organizations responsible for the particular action can receive only the needed information. The system also addresses the problem of scalable and fault-tolerant processing of the vast volume of data. To address the accuracy issues with the fully automatic processing system which are not capable of adjusting with the dynamic changes in the content, a hybrid approach is proposed which makes use of the human intelligence and adaptability for labeling the data using crowdsourcing combined with the supervised classification techniques to generalize human annotation behaviour and scale up the processing capabilities several folds. This hybrid processing technique is implemented on Apache Storm which is a near real-time distributed system in order to address the issues with large scale data processing and providing scalability. System uses different machine learning techniques like Part-of-Speech(POS) tags, Term Frequency-Inverse Document Frequency(TF-IDF) and slang dictionaries etc. to enhance the processing accuracy of the system. Evaluation shows that the proposed technique can provide the acceptable accuracy and can handle thousands of messages per minute. The system can be used by disaster analysts to quickly understand the needs and help in decision making.
Contents

1 Introduction 1
   1.1 Background and Motivation 2
   1.2 Problem Statement 3
   1.3 Overview of the Proposed Approach 4
   1.4 Document Structure 5

2 Foundation 7
   2.1 Crises and Disasters 7
      2.1.1 Humanitarian Disasters 7
      2.1.2 Situational Awareness and Crises Response 8
   2.2 Machine Learning 9
      2.2.1 Classification 9
      2.2.2 Distributed Stream Machine Learning 11
   2.3 Crowdsourced Stream Processing Systems 13
      2.3.1 Stream Processing Systems 13
      2.3.2 Crowdsourced Systems 14
      2.3.3 Crowdsourced Distributed Stream Processing Systems 14
   2.4 Apache Storm 15
      2.4.1 Components 15
      2.4.2 Streams 16
      2.4.3 Topologies 16
      2.4.4 Architecture 16
      2.4.5 Important Concepts and Features 17

3 Related Work 21
   3.1 Data Stream Mining 21
   3.2 Classification and Crowdsourcing 22
   3.3 Social media monitoring system focused on crises 22

4 Preliminary Analysis 27
B Latency Measures

B.1 Worker 1 ................................................................. 81
B.2 Workers 2 ................................................................. 81
B.3 Workers 4 ................................................................. 82
# List of Figures

1.1 Overall architecture of the proposed system ........................................ 5

2.1 Four phases of disaster response ....................................................... 8

2.2 General steps in text classification process ........................................ 10

2.3 Architecture of Apache Storm (Consisting of 2 Spouts and 5 Bolts) ....... 17

4.1 Word cloud of the most frequent words from the Haiti dataset ........... 28

4.2 Distribution of messages with respect to the number of words .......... 29

4.3 Hyperplane illustration in Support Vector Machines .......................... 34

5.1 Architecture of the prototype .......................................................... 39

5.2 Preprocessor Component Workflow .................................................. 42

6.1 Accuracy of the system with varying initial training set ....................... 58

6.2 Accuracy achieved by all the individual instances with the varying initial training data followed by continuous human annotated instances ............. 60

6.3 End-to-end latency of the system ..................................................... 61

6.4 Latency of the different components of the system ......................... 62

6.5 Latency of the different components of the system ......................... 63
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Summary of Dataset</td>
<td>29</td>
</tr>
<tr>
<td>4.2</td>
<td>Number of messages in each available category</td>
<td>30</td>
</tr>
<tr>
<td>4.3</td>
<td>Comparison of Accuracy, Kappa Value and Mean Absolute Error in SVM, NB and NBM</td>
<td>35</td>
</tr>
<tr>
<td>5.1</td>
<td>Examples of Unicode and HTML symbols</td>
<td>43</td>
</tr>
<tr>
<td>5.2</td>
<td>Examples of Slang Expressions</td>
<td>43</td>
</tr>
<tr>
<td>5.3</td>
<td>Examples of PTB Tagset</td>
<td>46</td>
</tr>
<tr>
<td>6.1</td>
<td>System specifications</td>
<td>60</td>
</tr>
<tr>
<td>A.1</td>
<td>Penn Treebank Part-Of-Speech (POS) tags [1]</td>
<td>80</td>
</tr>
<tr>
<td>B.1</td>
<td>Latency of different components with worker 1</td>
<td>81</td>
</tr>
<tr>
<td>B.2</td>
<td>Latency of different components with workers 2</td>
<td>82</td>
</tr>
<tr>
<td>B.3</td>
<td>Latency of different components with workers 4</td>
<td>82</td>
</tr>
</tbody>
</table>
List of Listings

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Dataset Spout (Abbreviated)</td>
<td>41</td>
</tr>
<tr>
<td>5.2</td>
<td>Punctuation between character regular expression</td>
<td>43</td>
</tr>
<tr>
<td>5.3</td>
<td>Example of ARFF Format</td>
<td>44</td>
</tr>
<tr>
<td>5.4</td>
<td>Instance Header Code</td>
<td>45</td>
</tr>
<tr>
<td>5.5</td>
<td>Instance Bolt (Abbreviated)</td>
<td>45</td>
</tr>
<tr>
<td>5.6</td>
<td>POS Tagger Component (Abbreviated)</td>
<td>46</td>
</tr>
<tr>
<td>5.7</td>
<td>POS Tagger Bolt (Abbreviated)</td>
<td>48</td>
</tr>
<tr>
<td>5.8</td>
<td>String To Word Vector Filter Code Snippet (Abbreviated)</td>
<td>48</td>
</tr>
<tr>
<td>5.9</td>
<td>Classifier Code (Abbreviated)</td>
<td>49</td>
</tr>
<tr>
<td>5.10</td>
<td>Statistics Class (Abbreviated)</td>
<td>51</td>
</tr>
<tr>
<td>5.11</td>
<td>Database Operations Class(Abbreviated)</td>
<td>53</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Every now and then, our world faces many humanitarian disasters such as earthquake, tsunami, hurricane etc. coping with disaster and crisis situations are one of the biggest challenges of our society. For handling such scenarios timely and correct information at the right time plays a very crucial role and can make a difference between a successful and an unsuccessful emergency response.

Gone are the days when the information was collected and transmitted face to face or by radio communication. Advancements in mobile networks and communication technologies, as well as improvements in the processing capabilities provide new possibilities to improve state of art emergency response methods. The increase in cell-phone use have seen everyone carrying cell-phones and as such are continuously connected. Therefore connecting to people has become much easier than before. One can imagine that during humanitarian disasters either natural or man-made, local citizens have first-hand reports regarding what is actually happening at the affected area. All this information needs to be collected from all possible channels in the affected areas, evaluated for truthfulness, filtered and then transmitted to the responsible team for required action. Use of text messages for information collection is becoming more and more useful and is being used in many scenarios. However with this increased availability of this data continuously coming in the form of data streams, filtering and assessing this information has become a key challenge [2]. In large scale emergency scenarios or natural disasters multiple organization and volunteer groups operate to handle the situation collaboratively. In such cases, information not only has to flow among single organization but also among all the collaborating organizations in order to provide proper and effective response.

This chapter serves to introduce the topic, discuss the motivation of this work, problem statement and overview of the proposed approach. It will end with the explanation of the document structure.
1.1 Background and Motivation

Crises situations such as natural disasters present some unique challenges like instant decision making by the organizations proving aid and relief work. For the purpose of decision making, availability of related data is one of the most important factors and automatic data processing can help in this aspect so that huge amount of data can be processed and presented for decision-making.

United Nations publishes a "Humanitarian Data and Trends" report every year and according to the latest report of 2014 [3], in the year 2013, there were a total of 352 natural disasters affecting 97 million people. The same report analyzed the increasing role of social media and short text messaging services for providing information in such scenarios. For a single incident Super Typhoon Haiyan almost 440,000 tweets were posted after the typhoon amongst which 44 percent were related to needs and donations [3]. The amount of data being collected in these incidents needs a level of automatic processing.

Another aspect of any disaster is the volunteer support. Volunteers are the first line of response in any kind of disaster. Especially in large-scale disasters, volunteers play a significant part in helping affected population [4]. Stallings et. al [5] defines three major activities volunteer groups perform during disasters. They are damage assessment, operations and coordination. Besides these main activities there are new forms of online volunteering which is becoming more and more useful. With the increasing use of SMS and microblogs, it is easy to gather information from a large number of people. During mass emergency if traditional way of "emergency calls to responders" is used, attending each and every call is a tedious work and is not feasible in humanitarian disasters one reason being volunteer burnout [6] or lack of required volunteers in the first place. Text messages were first used in the Haiti Earthquake of 2010 for information gathering, the project was named Ushahidi [7]. The reports [8, 9] discusses the latest surge in online volunteering and how after 2010 Haiti earthquake several technologies were used for the first time along with online volunteering to provide support. Despite damaged infrastructure in the whole affected area, people were able to report needs via SMS and Twitter. These reports were then translated from Creole into English, followed by entering these reports into Ushahidi [7] platform, then annotate data like location and finally present the reports on a map with topic filters.

This is the motivating factor for the work in this thesis project. To make the processing of large amount of data received continuously from the affected people possible so that it can aid in the decision making during such scenarios. Furthermore, having this information collected centrally has potential to reduce redundant relief efforts and speed up the process.
1.2 Problem Statement

Large-scale processing and mining of data during disasters is a new but very important field which needs to be addressed. Processing and making sense of large document collections is a field which has seen a lot of research being done on topics such as machine learning, information extraction, summarization, visualization. But these large document collections are different from the short messages in length, language and other meta-data. The language used in the short messages is one of the most important factors which need to be addressed as it may have a negative impact on processing capabilities.

**Information overload is a big issue during disasters and it can be as paralyzing to humanitarian response as the absence of information.** Fully automated techniques used to process and extract information from text during disasters have some shortcoming and are not feasible to be applied in sensitive scenarios. The issues with fully automated techniques will be discussed in details in the section 2.3.1. Due to the shortcoming in fully automated systems, volunteer based processing is preferred during such scenarios. Reason being the flexibility in adaptation and language understanding capabilities of humans. However, this approach is time-consuming and highly labor intensive which is why it is not feasible for long term disasters. A possible solution is using the combination of human-curated and automatically processed short message reports.

In this work, a novel approach will be proposed to process and classify the SMS data. The proposed solutions will also address the issue of performance bottleneck when running stream processing on single systems and preferred choice will be the use of distributed systems. The need for development of such approach is motivated by the Ushahidi project [9]. The system proposed in this work will be a step forward in solving this problem.

Research goal of this thesis is to:

Develop an approach to process, correct and classify text messages to be forwarded to the responsible teams so that relevant information can help in the decision making. This approach needs to be based on the distributed system paradigm so that issues with the system bottleneck and fault tolerance can be handled.

The questions outlines below are formulated based on the goal of this thesis.

1. How can large amount of data be processed in near real time?
2. How the text messages can be processed to enhance and correct the information in the text?
3. How relevant messages can be made available to the responsible organizations/teams?
CHAPTER 1. INTRODUCTION

1.3 Overview of the Proposed Approach

The proposed system for processing the emergency related text messages in near real-time over a distributed stream processing system consists of several sub-components. The overall architecture of the proposed system for data analysis is shown in Figure 1.1 and the components are as follows:

Text Messages  Text messages component handles the incoming stream of text messages sent by the affected people for help and aid. We receive these messages in the form of a continuous stream.

Preprocessing  Preprocessing components of the system process and make the incoming data ready for classification and training. Preprocessing include activities like parsing all the HTML tags to convert them to readable symbols, removing the stop words, replacing slang words with the proper grammatically correct words etc. After taking care of the unwanted components of the text message in preprocessing component, the text messages are then passed on to a trainer and the classifier components.

Trainer  Trainer is the components of the system which handles the active learning of the classifier. If the incoming text messages are human annotated then these are used for training the classifier. Trainer component also receive classified messages from the classifier components and these classified instances can also be used to continuously keep improving the classification model.

Classifier  Classifier is the most important component of the system. It creates a classification model based on the trainer component and classifies the text messages instances based on the learning. It also keeps on improving the classification model based on the annotated instances trainer receives in the future.

Database and User Interface  After the classification of the instances, the text message instances as well as the predicted label will be stored in the database along with the other tagged components for further use. This database serves as the repository which can be used to feed the user interface where the users as well as the other stakeholders, aid authorities etc. can check out the text messages.

Distributed Processing System  The classification pipeline discussed above will be implemented on the Distributed Steam Processing System in order to handle the volume of data coming in the form of streams. The system will also be fault-tolerant in order to handle the system failures in the cluster.
We will create a prototype of the proposed system and will evaluate the active learning and processing capabilities of the system as the part of this work.

1.4 Document Structure

The document is divided into 6 parts.

First part deals with the foundation and the background literature review of the available scientific literature.

Second part discusses the related work.

Third part discusses the preliminary analysis done on the available dataset and the approach derived for the system.

Fourth part details the base system implementation based on the discussed approach.

Fifth part details with the evaluation and the testing of the proposed system.

Sixth part concludes the work and discusses the limitation of the current work and directions for the future work.
Chapter 2

Foundation

In this chapter, we will review the literature regarding Crises and Disasters, Situational Awareness, Classification, Stream Processing Engines and Crowdsourcing Systems. This will help us in building the foundation of the work which will be done for this thesis.

2.1 Crises and Disasters

In this section, we will understand the basics of crises and disasters along with understanding how citizen communication during a disaster can help in situation awareness.

2.1.1 Humanitarian Disasters

As per the UNISDR terminology, a disaster is a serious disruption of the functioning of a community or a society involving widespread human, material, economic or environmental losses and impacts, which exceeds the ability of the affected community or society to cope using its own resources [10].

A humanitarian disaster (or humanitarian emergency) is an event or series of events that represents a critical threat to the health, safety, security or wellbeing of a community or other large group of people, usually over a wide area. [11]

Disasters can be classified into two types, natural and man-made. Natural disasters are the disasters which results from disturbances in nature. Natural disasters can be rapid like earthquakes, tsunamis etc. or slow onset like drought, animal plaques etc. Man-made disasters are events by humans which result in large scale issues like industrial accidents, transport accidents etc [12].

To handle these disasters and in order to help the affected population there are many aid organizations and teams established. These teams work in different phases of the disasters as per their
experts. As represented in Figure 2.1, a disaster can be described in four phases consisting of mitigation, preparedness, response, and recovery [13]. Mitigation is the phase which deals with the precautionary steps taken by the responsible authorities in order to avoid any kind of disaster or emergency situation by improving infrastructure. For example, by preventing people from constructing houses in highly earthquake-prone areas, building infrastructure to prevent floods, etc. Preparedness means preparing the population to respond to the disasters by the means of training or planning, etc. Response phase deals with the immediate actions which need to be taken after any disaster to make sure the maximum survival of the affected population. And finally, Recovery deals with the long-term process of restoring and restructuring the destroyed areas.

2.1.2 Situational Awareness and Crises Response

A general definition of situational awareness as suggested by Endsley states that "Situational awareness is the perception of the elements in the environment within a volume of time and space, the comprehension of their meaning and the projection of their status in the near future" [14].

Before any action can be taken for the disaster management, all the aid organizations as well as the governments collect as much information as they can so that an effective decision can be taken in reasonable time. This process which involves information gathering and assessment contributes to the high-level situational awareness of the disaster [15].

Situation awareness for a person/organization differs based on the situation they are in or based on what kind of aid they are providing. It is dependent on the role and situation of the particular end user. For example, information which is relevant to the organizations/hospital providing
medical aid differs from the information which is relevant to the search and rescue team. As per the United Nations guidelines of disaster relief efforts, relief teams should be divided using the clustering approach to handle different tasks related to the expertise of the organization. This is where the classification of different information comes into role in order to provide relevant information to all involved in the response and recovery operations.

2.2 Machine Learning

Machine Learning is the study which involves computation learning, artificial intelligence, pattern recognition etc. It consists of algorithms which can learn and adapt from the provided data, build models from this data and can make decisions for the data input based on these models.

2.2.1 Classification

Classification is a supervised data learning technique which is one approach of machine learning. It involves predicting and assigning a label to a new observation to which it may belong as per the trained model [16]. There are basically two types of classification:

- **Binary Classification**: As evident by the name, it classifies the new input objects into one of the two classes.
- **Multi-class Classification**: it deals with classifying the new input object into one(or more) of the multiple classes.

If we compare classification and clustering of the data, there is not much difference except that classification is predictive whereas clustering is descriptive. The past data on which classifier is trained helps in predicting the nature of the new data and as clustering does not depend on any past or training data, it is not preferable to use for prediction purposes.

Text Classification

Text classification is a part of classification in which the techniques and algorithms are applied on the documents of text. The term documents used here is in general sense and it can represent a chunk of data which can be a single sentence as well as a multi-page text document. The task is to assign one (or more) classes/labels to the documents based on the textual content. For example, consider a task where emails are required to be labelled as Spam or Ham, here two classes(spam and ham) are decided by the humans manually which will be used for classification and each email will be the document which will be labelled as either ham or spam based on the training.

Other popular areas where text classification is used include classifying news in business, politics, sports, world etc., classifying movies as good, bad, average based on the reviews and performing sentiment analysis using classification algorithms into happy, sad, neutral etc.

More formally, if there is a document set $D$ and categories $c_1, c_2, ..., c_n$ then text classification assigns at least one category $c_j$ to document $d_i \in D$ \[17\].

As with every supervised learning activity, for classification task also model have to learn how to classify the documents. For which it needs some kind of ground truth which can be provided by a dataset which is already classified into different categories. Model can learn the knowledge from this dataset called training data and apply on the fresh unlabeled data called the test data. In mathematical form classifier can be represented by the Equation \[2.1\]

$$f(d) \rightarrow C$$

Another important aspect of classification is the choice of size of the training dataset. If the classifier is trained on small dataset it may not acquire substantial knowledge to classify the new objects, on the other hand if the training data is too large compared to the test data it creates another issue called “Overfitting” \[18\]. Due to this overfitting, the resultant classifier function is tuned too finely on the training data, that the performance degrades on the test data.

Figure 2.2 represent the general text classification task graphically as explained by Ikonomakis et
2.2. MACHINE LEARNING

After reading the document and the text, text is then split into words(tokens) by the process called tokenization, followed by the processing of stemming in which all the word are reduced to their language stems. For example, the words "stemmed", "stemming" and "stemmer" are reduced to "stem". This process is generally followed by deletion of stopwords from the text tokens as they do not contribute to the classification. The remaining stemmed words are then represented in the vector form followed by the feature selection or feature transformation methods to categorize the vectors to be passed to the learning algorithm to be used create a final model which will be used for the classification task.

Short Text Classification

Previous section discuss the text classification on the documents of text. These documents are generally large text documents with rich textual content. Classifying these documents with traditional approaches like Bag-of-Words gives sufficient performance as the word frequencies are sufficiently high to capture the semantics of the document. But this is not true for the short texts like SMS messages, Social networking posts, Microblogging content etc. In this kind of communication platforms, the content per document is very small and word occurrences are not sufficiently high to rely just on traditional approaches [19]. This is also the focus of this work to deal with such small texts in order to fetch sufficient knowledge from the text messages.

2.2.2 Distributed Stream Machine Learning

Distributed Machine Learning

Availability of large amount of data and the requirement to process this data in a time-efficient way leads to the research in distributing the processing over a set of systems. The increase in digital data availability also resulted in the need to create distributed frameworks which can handle multi-machine processing. This leads to the development of several distributed machine learning frameworks which are capable of handling this huge amount of data(generally referred as Big Data) and process Machine Learning algorithms on the data in the distributed fashion.

Mahout [2] is one such example of a distributed machine learning framework which is based on Hadoop. Scalability while analyzing large amount of data is one of the goals of Mahout. It consists of a large number of algorithms for machine learning and automatic processing. GraphLab [3] is another framework which also have a distributed implementation called Distributed GraphLab [20] which is based on the shared memory model. MLBase [4] is another distributed machine learning framework which aim to ease non-experts access to ML.

Stream Processing and Machine Learning

Stream processing paradigm has emerged in order to create solutions for data which are continuously generated in the form of streams. Many systems capable of processing data streams are designed in the past to fulfill the demand of handling continuous data generation.

Stream Processing Engines (SPE) also called Data Stream Platforms, can be defined as software architectures or frameworks built to process continuous streams of data. Some of the initial SPEs were Aurora [21], STREAM [22] and TelegraphCQ [23] which operated in a centralized fashion on a single machine.

For streaming machine learning implementation, the system and the algorithm should fulfill several requirements, which are:

1. Streams are generally considered to be unbound and infinite. Which limit the processing capability of the data and restrict the algorithms from processing the data multiple times for the analysis. The analysis performed on data need to be "one go" due to the inability of memory to hold an infinite amount of data.
2. Algorithm should be able to learn continuously or at least learn whenever some change is detected in the data stream to improve the analysis ability. This is the reason why the streaming machine learning algorithms are referred to as online algorithms or active learning algorithms.
3. The time taken by the algorithm to process a newly arrived object should be bare minimum and that too using limited amount of memory.

The above-mentioned stream processing engines are just general-purpose processing engines and are not specific for machine learning through machine learning algorithms can be implemented on these engines. On the other hand, for streaming machine learning purposes, many specific frameworks are developed. One of the most famous streaming ML implementation being Massive Online Analysis (MOA) [5]. It consists of a lot of online machine learning algorithms for stream processing which can be used for classification as well as for clustering. But there is a limitation that it can works on a single machine only which limits the scalability. Vowpal Wabbit [6] is another implementation of online machine learning algorithms optimized for text data input. This implementation is also capable of parallel linear learning in the single-node settings.

Distributed Streaming Machine Learning

Existing streaming machine learning implementations are unable to scale very well in case the rate of data being received is very high. So to implement a system which can handle this high rate of data,
of data, we can take the distributed processing of machine learning algorithms with the stream processing capabilities of online algorithms.

One direction is to utilize the existing stream processing engines (SPEs) and implementation of active learning algorithms on top of these engines. Besides the above-mentioned SPEs which are restricted to a single machine, in the recent past multiple distributed stream processing engines are developed both open-source as well as commercial. Most prominent open source distributed SPEs are Apache Storm[^7] and Apache Spark[^8] and state of the art commercial SPEs are IBM InfoSpher Streams[^9] and Tibco’s StreamBase[^10] etc.

### 2.3 Crowdsourced Stream Processing Systems

#### 2.3.1 Stream Processing Systems

As discussed in the section 2.2.2 stream processing in short refers to the computation performed on an inbound, high-speed, continuous and time-varying stream of events [24]. If the stream processing can be performed timely and effectively it can help a lot in proper decision making specifically during the time of disaster. But there is a significant drawback of stream processing systems which prevents its use in such scenarios. The drawback is SPEs entirely rely on fully automated algorithms and specifically in case of disaster scenarios the incoming data is highly variable and previously unseen. Now if the decision are based on the intelligence of fully automated processing it can have a negative effect on decision-making. For example, let us assume there is a humanitarian disaster and we start collecting and processing data so that it can help in the decision-making process (similar to what was done after the Haiti earthquake in 2010). But in such scenarios there are some concerns which make automated algorithms inadequate, which are:

1. As the data we are relying on is user generated, it may be erroneous due to human mistake or due to the jargons used in such text typing.
2. The data being received continuously may have different characteristics than the previously received data, this issue is called domain adaptation problem, and due to this issue, the model trained on a specific data does not perform well on another data.

Due to this domain adaptation problem, we cannot use models trained on different past events for classifying data being received in the new disaster. Imran et. al [25][26] studied this problem and tried to use model trained on one disaster to classify data from the another disaster but concluded that these kinds of models perform very poorly and the use of such models should not be preferred.

[^7]: https://storm.apache.org/
[^8]: https://spark.apache.org/
[^10]: http://www.streambase.com/
One reason for this behavior may be the presence of unique vectors which may not have been reported in the previous situations.

So, in conclusion it becomes apparent that in scenarios where decision need to be made in real time based on continuously incoming data and where false or incorrect information can be crucially harmful, fully automated machine learning algorithms cannot be relied upon.

### 2.3.2 Crowdsourced Systems

Crowdsourcing refers to the approach where the potential of a large crowd of people is harnessed to perform certain tasks. Crowdsourced systems are the specifically designed system where an open call is made to people to contribute towards the goal and tasks are distributed to the crowd to get their input [27]. In disaster response domain, Ushahidi[^11] is one of the examples which was one of the very first systems of its kind deployed after the Haiti earthquake to help in the relief work using the power of crowdsourcing.

Crowdsourcing paradigm has been used to address the issue being faced by the fully automated systems as discussed above. In these systems we can take advantage of the human intelligence and the adaptability. In fact, in the scenarios where decision making is highly sensitive and error prone, human intelligence can beat the fully automated system in text recognition, understanding, analysis and information extraction. But crowdsourced systems has its own limitation, as it completely rely on the task completion by humans, it results into low throughput as humans are limited in speed with which they can process the input data which certainly is very less as compared to the stream processing systems which can process thousands of data instances per second.

### 2.3.3 Crowdsourced Distributed Stream Processing Systems

As fully automated stream processing machine learning systems may not be appropriate for the tasks which require high precision because of the classification quality issue with dynamically evolving data. Fully crowdsourced systems for processing may also not be appropriate due to the time taken in processing which result in low throughput.

If we combine human intelligence of crowdsourced systems with the automatic high-speed processing power of stream processing engines both the problem can be solved. This combined system called Crowdsourced Stream Processing Systems can harness the advantages of both the systems while leaving the disadvantages of both the system behind. Different systems can have a different level of human involvement and automation depending on the requirement of the system.

So, a crowdsourced stream processing system can be defined as a combination of two systems, a stream processing system and a crowdsourcing system. This class of systems maintain the objective of the stream processing system i.e. high throughput processing and efficiency while using the another system i.e. the crowd and its knowledge to attain maximum accuracy. Now when this stream processing is done on the distributed system instead of a single system to further increase the processing capacity, the systems can be termed as crowdsourced distributed stream processing systems. In our problem i.e. disaster response scenario, the crowdsourced component of the system can be assigned the task to perform the initial classification of the incoming data elements which then will be used to train the classification model and then the task of classification can be automated using this trained classification model. Once the automatic processing starts, volunteers need to keep an eye on the accuracy of the system and if the accuracy start falling for certain kind of elements signifying the concept drift then these new elements can be classified by the volunteers which then will be fed to the classification model to train it further to handle new kind of elements.

2.4 Apache Storm

Apache Storm[12] is a free and open source distributed real-time computation system. Storm provides guarantee to reliably process unbounded stream of data, doing for real-time processing what Hadoop did for batch processing. A benchmark clocked it at over million tuples processed per second per node. It was created by Backtype 2011, which was acquired by Twitter that same year. The project was later moved to Apache Software Foundation as an open-source contribution.

2.4.1 Components

A Storm cluster is quite similar to Hadoop cluster. Hadoop runs "MapReduce jobs" and on Storm we run "topologies". Topologies and jobs are basically very different, one main difference being that jobs in Hadoop finish the processing after certain time period whereas topologies process messages forever until it is killed manually[28].

Storm had two types of nodes: Master node and the Worker node.

Master node runs the daemon called Nimbus, which is responsible for distributing code around the cluster, assigning tasks to machine, and monitoring for failures. Each worker node runs a daemon called the Supervisor. This supervisor listens to the work assigned by the Nimbus and then handles worker processes(start and stops) as necessary based on the work assigned to it. A topology can consist of many worker processes which can be spread across multiple machines.

[12]https://storm.apache.org/
2.4.2 Streams

A stream is just an unbound sequence of tuples. A tuple is a named list of values. Storm process these streams to transform these into new resultant streams. For processing these streams, storm uses Spouts and Bolts.

A Spout is simply a source of streams. A spout can read tuples from different sources like live feed, API or even text files and pass them as stream to bolts for further processing.

Bolts are used for stream transformations. Each bolt should ideally only performs single step transformations on the streams. A bolt can have any number of streams as input for computation and emits resultant streams to be processed by other bolts or to be stored/passed on to users.

2.4.3 Topologies

Topologies are top-level abstraction of Storm. Topology is the package of multiple Spouts and Bolts and it is passed to the Storm cluster for execution. Topology is a graph of stream transformations where each node is either a spout or a bolt [28].

Note: In stream processing, it is really important that data is processed by nodes at the rate higher or at least at the same rate as new data arrives. Otherwise new data tuples will have to wait to be processed which add latency to the system. To overcome this condition, in a Storm topology we can specify for each process how much parallelism is required so that the processes which are expected to take longer time to process can be processed in parallel to reduce the latency as much as possible.

2.4.4 Architecture

Storm architecture consists of two parts: Spouts and Bolts. Spouts are the parts from where stream enters the processing topology and these spouts pass the continuous stream of data to bolts as tuples. Bolts are the parts which process these tuples. There can be multiple layers of bolts in a topology, a bolt can process the tuples and passes it to another bolt for further processing. As shown in the Figure 2.3, we have a layer of spouts which passes tuples to a layer of bolts containing three bolts and these bolts process the incoming tuples and passes the results to the second layer of bolts.

2.4. APACHE STORM

2.4.5 Important Concepts and Features

Stream Grouping

A stream grouping tells how to pass tuples from one component to another. We can understand this better with the following example. Imagine a bolt sends website clickstream data tuples which has the following fields: timestamps and webpage. Now let’s say we have next bolt which had a parallelism of 6 instances to count the clicks on these webpages. It is better if same webpage go to the same tasks so that we need not merge the results later which could result in some latency. To avoid this, we can group stream by the field webpage. Following are the field grouping available in the Storm architecture [29]:

1. Shuffle grouping.
2. Fields grouping.
3. Partial Key grouping
4. All grouping
5. Global grouping
6. None grouping
7. Direct grouping
8. Local or shuffle grouping

*Shuffle Grouping* and *Field Grouping* are the most commonly used field grouping in Storm topologies.

**Guaranteed message processing**

In Storm topology, it is guaranteed that each message will be processed. These tuples coming out off the spout can spawn many tuples. Storm considers a spout tuple fully processed when every tuple in the tree has been processed and is considered failed if the tuple is not able to be processed within a specific timeout period, which is 30 seconds by default [30].

**Fault Tolerance**

Fault tolerance means the ability of the system to operate properly in the event of failure of one or more components. Even if the processing quality of the system decreases, it should be in proportion to the severity of the failure and should not cause complete breakdown to operate fully or atleast at the reduced level rather than failing completely [31].

As there are multiple components in Storm anything can fail at any given time. For example, if worker node(spout and bolts) fails, supervisor will try to restart it and if it still fails Nimbus will reassign the worker to another machine. In case if node dies then the tasks assigned to that particular machine will time out and Nimbus will come to know about node failure and can assign the work to a new node. So far Nimbus is kind of Single Point of Failure(SPOF). But nothing disastrous happens if Nimbus dies as the worker nodes keep on working. Only drawback is work cannot be reassigned to any other nodes in case of requirement. Making Nimbus also fail safe is work in process in the development community [31].
Chapter 3

Related Work

The data we will consider in this work is a sequence of messages continuously coming from a data stream. From that perspective, we are interested in data stream mining. We will cover a number of works that deals with data stream mining in this chapter. The work also integrate Crowdsourcing and Classification and we will have a look at some of the related work in these fields also. Finally we will have a look on the existing systems which are developed to monitor social media platforms to collect crises related information.

3.1 Data Stream Mining

Karp et al. [32] proposed an algorithm which depends on the frequency of the elements in the data stream to figure out the important nuggets of information. The proposed scheme store every element along with a counter which keep track on the frequency of that particular element and increment by one if the element is present in the input and decrease it by one if not. Once the element counter reaches 0, the element will be discarded. At any given time, the elements which have higher counter than a threshold $\theta$ will be important. This is one of the simplest implementation for data stream mining.

Teng et al. [33] proposes a regression-based algorithm which mines data streams to find frequent temporal patterns in real time. They used a time based sliding window of a predefined size and calculated element frequency in this particular window every time new data arrives. All the elements are then assigned a cumulative frequency based on the past frequencies of the element and the frequency in the current window. To optimize this approach, Teng et al. also uses the same threshold approach as used by Karp et al. i.e. only those elements which have frequency above a certain threshold $\theta$ are considered.
3.2 Classification and Crowdsourcing

Learning classifiers are one of the most used approaches for classification and mining data to extract information. The problem faced by classifiers when applied on short text messages has started getting significant attention in the recent times. Cormack et al. [34] and Gómez Hidalgo et al. [35] proposed approaches to handle spams in short text messages using various different machine learning algorithms like Naive Bayes, Decision Trees, Logistic Regression, Support Vector Machines etc. They also evaluated various feature representations such as Bag of Words(BoWs), feature selection, feature extraction, character bigrams and trigrams etc.

Sriram et al. [36] discussed about classifying tweets to a predefined set of generic classes like deals, news, events, private messages etc. based on author information and domain specific information like presence of slang words, mention of another user name, presence of percentage values or currencies etc. Similarly Lee et al. [37] also processed tweets and trending topics to classify the topics into 18 categories like technology, sports, politics. Authors analyzed two approaches, a text-based classification and a network-based approach in which top five similar topics of a given topic are identified and the networking between the topics is used for the classification purpose for which C5.0 decision tree classifier is used. Both approaches achieved 65% and 70% accuracy respectively.

The use of online text communication and social media following the 12 January 2010 earthquake in Haiti are much appreciated and is one of the first instances where online text communication helped in relief work during a disaster. Crowdsourcing platform Ushahidi played a very important role in collaborating with the crowd and the volunteers in order to translate the text messages, so that English speaking aid organizations can understand the text. Robert Munro [38] analyzed the Ushahidi project and its crowdsourced translation of text messages written in Haitian Creole to English language during the the earthquake and the following months. Using online crowdsourcing and volunteering, it was possible to translate a good number of messages during rescue operations which helped in saving many lives and providing the required support to the people in need.

3.3 Social media monitoring system focused on crises

Many systems have been developed or are under development which monitors social media especially Twitter to extract information related to crises. MacEachren et al. [39] developed SensePlace which deals with geographical analysis of tweets to provide support for the situational...
awareness during crises. The system focus on finding explicit and implicit geographic information from the tweets and to provide user interface for better understanding of the geographical data. Cameron et al. [40] proposed and developed ESA:Emergency Situation Awareness [4] and Abel et al. [41] proposed and developed Twitcident (Now Crowdsense) [4] These systems analysis Twitter messages posted during emergency scenarios and uses various natural language processing techniques and semantic enrichment for early detection of events and to extract information relevant to situational awareness.

**AIDR: Artificial Intelligence for Disaster Response** [5] is the project developed by Qatar Computing Research Institute (QCRI) [6]. The project focuses on developing artificial Intelligence for disaster-related decision-making scenarios. AIDR uses crowd and the volunteers to annotate tweets manually and then train the classifier which uses machine intelligence to automatically tag up to thousands of messages per minute [42].

For further reference, the survey paper by Imran et. al. [43] is a good resource for detailed comparison of some of the systems.

Most of the existing systems which focus on crises management or situational awareness uses dashboards or other visual methods to display the output and to enable people to have an overview of the situation. Most common elements are:

- **Collection of social media messages related to given topics/classes from different platforms.** Twitter is one of the most popular platform for analysis due to its available API which makes accessing the data very easy.

- **Most systems rely on systems for processing of the data automatically or volunteer/crowd supported processing to perform classification and extraction of information from the data using concepts like natural language processing, named entity recognition etc.**

- **Maps representing geotags and positions of the most affected areas or from where the messages are received.**

On the backdrop of all the related work, we mainly focus on correctly classifying text messages for the emergency purposes using a combination of crowdsourcing and automatic classification and to perform all this computation on top of fault-tolerant distributed system. As per my knowledge there is no such system available which uses the fault-tolerant and scalable distributed system for the processing of the streams. All the available system may or may not be stream processing but none of these uses distributed stream processing system like Apache Storm. None of the available systems so far concentrate on processing the text messages on the stream in near real-time, most of these systems concentrate on the twitter microblogging site or other such platforms. As evident
by the use of text messages in relief work of Haiti earthquake that in the coming times, the use of text messages will increase for such scenarios. This is further supported by the implementation of Text-to-911 and other systems which are trying to implement the option of sending text messages to 911 and other emergency services. With the successful implementation of such services, it will be more easy to use the text messages for the disaster scenarios. The need to process these messages is where our system concentrates on.

7http://textwith911.ca/
Chapter 4

Preliminary Analysis

In the previous chapters we have discussed the foundation literature for this work and the related work done in the field of situational awareness and stream data processing. This chapter focuses on the preliminary analysis of the processes and the techniques which can be used for the implementation on the stream processing engine. In this chapter we will discuss dataset details, pre-processing of the data and different classification approaches. The purpose of this analysis is to provide an early indication of a degree to which a state of the art classification algorithms can be used to process the streaming text messages collected during ongoing disasters.

4.1 Data Collection and Data Pre-processing

4.1.1 Data Collection

Launch of Ushahidi Mission-3636 after the Haiti Earthquake of 2010 collected almost 85000 text messages over the span of six months [44, 45] from the affected local people in the need of aid or help. Out of these 85000 messages collected, only a small subset of text messages are available publicly which is under Creative Commons Attribution Share-Alike Licence. The dataset is available on the Datahub website on the link http://datahub.io/dataset/ushahidi/resource/81d058a8-173a-49d9-8ce9-4edf5e7cafc9

4.1.2 Dataset details

Haiti dataset consists of 3584 text messages in total. Available information in the dataset is:

- Serial Number
- Incident Title
CHAPTER 4. PRELIMINARY ANALYSIS

Figure 4.1: Word cloud of the most frequent words from the Haiti dataset

- Incident Date
- Location
- Description
- Category
- Latitude
- Longitude
- Approved
- Verified

Serial Number and Incident Title are the columns which are added by the volunteers after the translations thus are not the part of the messages received originally. Similarly Approved and Verified columns, which as indicated by the name, represents if the particular text message was verified and if it was approved for the action. Out of all the columns in the dataset, we require columns Incident Date, Description which is the content of the messages, Category which contains the labels assigned by volunteers. We will use these labels for training and testing purpose.

Figure 4.1 shows the word cloud of the frequent words in the Haiti earthquake dataset we have. More frequent the word, larger the font will be in the word cloud. The most frequent words in the dataset are need, help, food, please, water followed by people, delmas, haiti, name, house, tents etc.

Average number of words in text message are 27.46 in the available dataset with the minimum of 1 word and maximum of 306 words(Table 4.1). Messages with the total number of words in 300s are very rare and these can either be multiple relay messages or the email sent to the volunteers, source is not made clear in the dataset or any supporting document. Text messages with just 2-3 words are not much useful or atleast cannot contribute much to the automatic processing, but these
messages are very less in number. As mentioned in the Table 4.1, 97.4% of the messages have more than 5 words and 88.6% messages have at least 10 words. Figure 4.2 illustrates the distribution of messages with respect to the number of words in the messages.

Table 4.2 shows the number of messages available for each category. Most number of messages are available for the category Vital Lines and the least number of messages are available for Public Health. Detailed analysis of the categories and the category selection process is discussed in the report "Independent Evaluation of the Ushahidi Haiti Project" by Morrow et al. [44].

First overview of the text messages suggests that the emergency messages in such situations can contribute significantly in distributing important information to the responsible aid organizations. Some of the examples of the text messages are:

- my name is carline pierre im not dead . i am under the rumbles in university caraibes , which is in delmas 29 . please come and get me ! im waitin for you
- we are family members of juan antonio zuniga ornelas (male-26 years old-hispanic look)7 weeks ago, he was around the area when we lost total contact with him. we suppose, he is
CHAPTER 4. PRELIMINARY ANALYSIS

<table>
<thead>
<tr>
<th>Category</th>
<th>No. of Messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Infrastructure Damage</td>
<td>137</td>
</tr>
<tr>
<td>Medical Emergency</td>
<td>462</td>
</tr>
<tr>
<td>People Trapped</td>
<td>72</td>
</tr>
<tr>
<td>Person News</td>
<td>244</td>
</tr>
<tr>
<td>Public Health</td>
<td>65</td>
</tr>
<tr>
<td>Security Threats</td>
<td>66</td>
</tr>
<tr>
<td>Services Available</td>
<td>421</td>
</tr>
<tr>
<td>Vital Lines</td>
<td>1994</td>
</tr>
</tbody>
</table>

Table 4.2: Number of messages in each available category

• injured in some way, and this could be the main reason, he is unable to contact us. we beg for your help, zuniga’s family. thanks phone (951) 443-9756

• i live in the site marassa 7. i ask some help like water, food and toilet thank you

4.1.3 Data Preprocessing

Pre-processing in general is used in a lot of research projects and applications which consider using raw and unstructured data. Thelwall et al. [46] in the article “Sentiment strength detection in short informal text” concluded that “Text based communication in English seems to frequently ignore the rules of grammar and spelling”. In order to produce meaningful insight from the available data, data cleaning and preprocessing is one of the first steps. It is an essential step before the data is ready for analysis. Text message is an informal mode of communication and the content is unstructured and contains typographical errors, slang usage, expressions etc. which may have a negative affect on the automatic analysis approach. The techniques applied for the cleaning is quite straightforward like punctuation correction, slang words conversion, stopword handling etc. Following are the steps which are used for data cleaning and preprocessing most of the raw data.

Parse HTML characters  Dataset we received from datahub repository contains lot of HTML entities like &lt; &amp; etc. It is thus very important to parse these entities and convert to standard html tags.

Decoding dataset content  Text data could contain data with different form of decoding like “Latin”, “UTF8” etc. Latin data encoding usually contains more complex symbols and characters. It is better if the complete dataset is in a single standard format. One of the most widely used encoding format is UTF-8 and it is highly recommended. We have converted the whole dataset in the UTF-8 encoded format.
4.2. CLASSIFICATION APPROACHES

Apostrophe Lookup  Apostrophe creates an issue of word sense disambiguation. For example, didn’t should be changed to did not. For classification to be proper, it is better if text sentences are in proper structure and abide by the rules of context-free grammar.

Stopwords removal  Stopwords almost never adds to the quality of the classifiers, as they are the most frequently used in almost every text document or string and will not be related to any particular class we are trying to classify into. So, it is better to remove the stop words completely from the text sentences. It can be done on the data clearing level or at the pre-processing level just before processing sentence by the classifier model.

Slang words  In textual data, especially in text messages or social media content where we have a limitation on the length, people use slang words heavily and quite frequently. It’s better if these words can be converted to the standard words. The words like OMG should be converted to Oh My God and FYI to For Your Information etc.

Standardizing Words  Standardizing words involves removing the repeating words which are used quite frequently in general text communication. For example, Pleaseee Heelppp need to be standardized to Please Help before it can be passed on to classifier for processing.

Stemming  Stemming is the technique to reduce the conjugations of verbs to their stem (the original root form). It is used to reduce the diversity of words. For example, the words "stemmed", "stemming" and "stemmer" are reduced to "stem" and words like "liked", "liking" and "like" are reduced to "lik". This technique helps in improving the classification by reducing the features and the amount of data.

4.2 Classification Approaches

As classification is the approach to distribute data items into predefined classes based on the properties representing different classes. The goal of classification is to address specific requirements [47][48] and to group data into the specified group unlike clustering which involves the task of grouping together similar objects into clusters having similar properties and not necessarily adhering to the predefined classes or groups.

4.2.1 Naive Bayes

Naive Bayes classifiers are linear classifiers. In Naive Bayes classification approach a most likely class $c$ is assigned to a given document $d$ based on the probabilistic approach which is based on
Bayes theorem [49].

\[ P(w_j|x_i) = \frac{P(w_j) \times P(x_i|w_j)}{P(x_i)} \] (4.1)

The objective function of naive bayes probability is to maximize the probability given in the Equation 4.1 given the training data in order to formulate the decision rule.

\[ c \leftarrow \arg \max_{j=1,...,m} P(w_j|x_i) \] (4.2)

Naive Bayes classifier uses a set of training data for the learning task in which it estimates the model parameters and then uses this model to classify new data into classes. Naive Bayes classification approach consider all the features are independent of each other and also independent of the position within the document this is where adjective naive comes in the name. The underlying distribution of naive bayes algorithm for classification is that it uses binary distribution which represent feature vectors as binary (i.e. 0s or 1s) meaning if a vector is present it will be 1 otherwise 0. Despite this assumption being unrealistic in the real world applications and data, naive bayes is proven successful in many practice applications [49–51].

### 4.2.2 Naive Bayes Multinomial

Naive Bayes Multinomial (NBM) is based on the Naive Bayes algorithm but with the difference in the underlying distribution model. In this approach, instead of considering feature vectors as just binary, their discrete counts are used. For example, if we have a classification problem, we just do not consider if a particular word is present in the document or not instead we consider how often that particular word is present in the document [52].

### tf-idf Measure

The Term Frequency-Inverse Document Frequency (tf-idf) measure represents the importance of a word in the given dataset. This measure consists of two measures clubbed together, the frequency of the term and the inverse frequency of the documents.

Term frequency is the number of times a word appears in a document. For calculating the term frequency, all occurrences of a word \( w \) is counted in the document \( d \) of the document set \( D \). Equation 4.3 is the mathematical representation of the tf-idf measure.

\[ tf(d, w) = |\{w \in d\}| \] (4.3)

Document frequency is the number of documents in which a particular word \( w \) appears. Equation
4.3 EXPERIMENTS

4.4 represents the mathematical equation of document frequency. From this document frequency, inverse document frequency (idf) is calculated as represented in the equation 4.5. Inverse Document Frequency represents the rareness of a word across all the documents in the dataset.

\[
df(w, D) = |\{d \in D : w \in d\}|
\]

(4.4)

\[
idf(w, D) = \log \frac{|D|}{df(w, D)}
\]

(4.5)

Term frequency combined with the inverse document frequency represents the word which occurs often in a document and does not present often in the whole set of document. This combination is represented mathematically in the equation 4.6 where \( w \) represents a word in a document \( d \) which belongs to the document set \( D \).

\[
tf - idf(w, d, D) = tf(w, d) \times idf(w, D)
\]

(4.6)

4.2.3 Support Vector Machines

Support Vector Machines (SVM) is another popular and traditional approach for text classification. SVM is a linear classifier which depends on finding hyperplane using the training data so that it divides the document vectors in one class from another \[53\]. The margin in SVM is the distance of the hyperplane to the nearest items in the either side of the class \[54\]. Margin should be the largest possible distance. The instances closest to the hyperplane are called the support vectors, this is from where the algorithm got its name support vector machines. Figure 4.3 shows the graphical representation of the hyperplane and the support vectors.

As the case with other classifiers also, SVM also depends on the training data, larger the training data better the accuracy of the classifier. When using SVM for text classification term frequency-inverse document frequency and removal of stop words improves the accuracy slightly.

4.3 Experiments

Preliminary experiments are designed to explore what features and classification algorithm (out of 3 mentioned in the previous section) result in the best performance for the short text messages dataset.
We used WEKA [55] for the initial experiments and used the default implementations of Naive Bayes, Naive Bayes Multinomial and Support Vector Machines.

For all the algorithms, final preprocessing properties selected are:

- IDF Transform - True
- TFT Transform - True
- Lower Case Token - True
- Minimum Term Frequency - 5
- Normalize document length - No Normalization
- Output Word Count - True
- Stemmer - Lovins Stemmer
- Tokenizer - Word Tokenizer
- Use Strop Words - True
- Words To Keep - 1000
- Cross Validation Folds - 10

Table 4.3 shows the experimental results for Support Vector Machines(SVM), Naive Bayes(NB) and Naive Bayes Multinomial(NBM) algorithms on the Ushahidi text message dataset. All the measures obtained are using 10-fold cross-validation.

As it is evident from the Table 4.3, Naive Bayes Multinomial outperforms the Naive Bayes and Support Vector Machines for text message dataset. Naive Bayes Multinomial output results are 67.67% correct and 32.33% incorrect predictions with respect to Support Vector Machines algorithm.
4.3. EXPERIMENTS

<table>
<thead>
<tr>
<th></th>
<th>SVM</th>
<th>NB</th>
<th>NBM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correctly Classified Instances</td>
<td>64.92% (2265)</td>
<td>51.65% (1802)</td>
<td>67.67% (2361)</td>
</tr>
<tr>
<td>Incorrectly Classified Instances</td>
<td>35.08% (1224)</td>
<td>48.35% (1687)</td>
<td>32.33% (1128)</td>
</tr>
<tr>
<td>Kappa Statistics</td>
<td>0.3842</td>
<td>0.3066</td>
<td>0.4365</td>
</tr>
<tr>
<td>Mean Absolute Error</td>
<td>0.1775</td>
<td>0.1089</td>
<td>0.0729</td>
</tr>
<tr>
<td>Total Number of Instances</td>
<td>3489</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4.3: Comparison of Accuracy, Kappa Value and Mean Absolute Error in SVM, NB and NBM

with 64.92% correct classifications and the worst performance was of Naive Bayes with only 59.65% correct classifications. Mean Absolute error of NBM is least at 0.0729 with SVM having 0.1775 and NB at 0.1089.

In the preliminary analysis, it is quite clear that the suitable algorithm for the implementation in the distributed real-time streaming system is Naive Bayes Multinomial. We will use this algorithm for the implementation on top of Apache Storm so that distributed processing capabilities of Storm can improve the performance of the classification algorithm in the environments where the processing speed is considerably important.

In the next chapter, we will discuss about the development of the system prototype and the various components of the system.
Chapter 5

System Development

In this chapter, we will discuss the prototype of the system which has been developed within the scope of this master thesis. It is based on Apache Storm\(^1\) to provide near real-time capabilities. Development of the prototype is based on the ideas and concepts which are discussed in the previous chapters and uses natural language processing like POS tagging and supervised learning.

The following sections provide the overview of the design objectives, followed by the explanation of the architecture and the components of the prototype.

5.1 Design Objectives

In this section we will discuss the design principles that will help building the system. These principles are applicable to almost all the stream processing systems in order to work efficiently. The objectives of the system are common for the general stream processing literature\(^{[56,57]}\). These objectives as studied earlier are general and not specific to the use in disaster-related scenarios, we will adopt these objective to fit our use case.

5.1.1 Minimum use of the crowd

The labeling of the messages need to be done by the volunteers or it need to be crowdsourced. There are many commercial crowdsourcing platforms like Amazon MTurk\(^2\), RapidWorkers\(^3\), etc. These services can be used for labeling the messages but as these platforms are paid services, the system should need as minimum crowd support as possible to make the process acceptable.

---

1. https://storm.apache.org
2. https://www.mturk.com/
5.1.2 Low Latency and High Throughput

Latency of the system is the time it takes to process a single item or an instance from end to end. Low latency of any system is the desired state as it symbolizes the efficiency of the system to deliver results in minimum possible time. In stream processing systems low latency is one of the most important properties. Latency of any system can be measured as the average time the system takes to process single item from end-to-end. As we will have several different components for the proposed system latency of the overall system as well as each components will be measured and discussed while evaluating the system.

Throughput is the speed of the system in the terms of data being processed. Higher the throughput, better the system. Ideally the performance of stream processing systems in term of throughput should be near real-time [56][58]. In general, the stream processing systems which depend on crowd for the input have lower throughput than the fully automated stream processing system but in scenarios like emergency where precision of the work is also of high importance, human intelligence is required to improve the precision of the processing system. Moreover, even in crowdsourcing system low latency is very much possible as concluded by Bernstein et al. in their work "Crowds in two seconds: Enabling real-time crowd-powered interfaces" [59].

5.1.3 High quality

The output results of the system should be of high quality. In crowdsourced stream processing systems, quality depend on both crowd annotation quality as well as the machine processing. The work done by the crowd/volunteers may vary hugely in the quality. As automatic systems depend on the work done by the crowd, system quality will also vary based on the training data quality. Depending on the domain, it varies how we can measure the quality of the system. The quality matrix can be a single scalar measure or may depend on the several aspects based on the domain. In our domain, the accuracy of the classifier gives the quality matrix of the whole system.

5.1.4 Load Adaptability and Scalability

Adaptability is the ability of the system to automatically adapt to the changes in rate with which data is coming. In streaming setup, data input rate may experience sudden burst or may drop significantly. System should be able to handle the changes. Sudden bursts in input rate may require load shedding which involves dropping data which cannot be handled by the system. One solutions for handling such dynamic changes is buffering. Using a buffer queue to store incoming messages in cases when system is not able to process at the moment is a good option [60].
5.2 Architecture

The architecture of the prototype system is aligned to a Storm topology. Every component of the system is wrapped into a Bolt which start working as soon as Spout provides the incoming data. Prototype topology consists of one Spout to input data and following eight Bolts:

1. Preprocessor
2. Instance
3. String To Word Vector
4. POS Tagger
5. Classifier
6. Statistics
7. Database
8. Stats Writer

Figure 5.1 illustrates the workflow of the components and the topology on a whole. The Dataset
Spout emits the text messages from a local dataset into the pipeline for processing. This Spout can easily be changed to handle the live stream of data also, but for the prototype we use a local dataset as discussed in detail in the section 4.1.2. The text messages emitted by the Spout are then passed to the Preprocessor Bolt which performs the preprocessing activities like HTML encoding, fixing slang words, remove elongations, fixing gerund forms of the words. After the preprocessing is finished, message item is passed to the Instance Bolt which converts the message into an instance compatible with WEKA format. After the text message item is converted to the WEKA instance it is passed to the next bolt which is String To Word Vector Bolt. This bolt converts the text message string to vectors using properties like TF-IDF, stop word removal, stemming etc based on the selected properties and values as discussed in the section 4.2. POS Tagger is another Bolt which receives the text message in the raw form after processing by the Preprocessor Bolt. POS tagger then predicts the part-of-speech labels for each token and forwards them to the Classifier Bolt as well as stores it to the database which in this case is MySQL database. Classifier Bolt on receiving feature vector from String to Word Vector and POS Tagger is able to use the initial human annotated instances for training and then predicts the labels for the instances which follows. Due to the online learning capabilities of the classifier, model may keep on learning from the instances which are automatically predicted by the classifier or from the human annotated instances which arrive after the prediction is started in order to keep improving the model. Finally these predicted labels are passed on to Database Bolt and Stats Writer Bolt in order to be stored in the database and for the calculation of the statistics like accuracy of the classifier, latency etc.

In the next section, each component of the prototype is discussed in detail.

5.3 System Components

The incoming messages during a disaster may vary extremely in the content and the needs. The automatic system first need to learn how to handle different types of messages coming from the users. To provide this knowledge to the system, as we discussed in the previous chapters, initial messages need to be annotated by the volunteers and then feed to the system from which it can learn to process different types of messages.

In this work we are emphasizing on the automatic part of the system and make an assumption that pre-annotated messages are available to be feed to the system. These annotation can be done using some online crowdsourcing system or by onsite volunteers. There are several websites and companies which provide crowdsourcing facilities like Amazon MTurk, RapidWorkers, MicroWorkers or companies which provide specialized crowdsourcing for cleaning and labeling.
More detail study into crowdsourcing system is available in the work done by Yuen et al. [61] which provide a survey of crowdsourcing systems and Zhao et al. [62] discusses the evaluation of the crowdsourcing system, its current state and the future directions of research. As for the automatic classification system, prototype of the proposed system consists of the following components:

### 5.3.1 Dataset Spout

The *Dataset Spout* is the entry point of the prototype system topology. At initialization, it reads the text messages from the dataset and emits them into the Storm stream continuously. Listing 5.1 shows the code snippet which is used for accessing the dataset and emitting the tuples to the first bolt for further processing.

```java
@override
public void open(Map conf, TopologyContext context, SpoutOutputCollector collector) {
    linesRead = new AtomicLong(0);
    _collector = collector;
    try {
        fileName = "dataset_file_path_and_name";
        reader = new BufferedReader(new FileReader(fileName));
    } catch (Exception e) {
        throw new RuntimeException(e);
    }

    @Override
    public void nextTuple() {
        try {
            String line = reader.readLine();
            if (line != null) {
                long id = linesRead.incrementAndGet();
                _collector.emit(new Values(line), id);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

Listing 5.1: Dataset Spout (Abbreviated)
```

[61] http://www.crowdflower.com
5.3.2 Preprocessor

The *Preprocessor Bolt* is the first bolt in the topology and as evident by the name, processes the text message tuples and prepare them for the classification task. *Preprocessor* performs multiple operations on the text message tuple and uses regular expressions for the purpose. The overall workflow of the bolt is illustrated in the Figure 5.2.

Bolt start by splitting the text messages tuple into several components and create an object of the class *SMS* which holds the content of the message, if it is human annotated, id of the message etc. The next process is to trim the text to remove leading and trailing white-spaces. Then the unicode and the HTML symbols from the content of the text are replaced with the corresponding symbols. Table 5.1 contains and example of the unicode and the HTML symbols.

After replacing unicode and HTML symbols, the next step is to remove repeating characters from the words. In this step regular expression is used to check if a word contains three or more same repeating characters. Sixth step of the processing is to correct slang words. In which *Preprocessor* tries to substitute slang expressions to proper English phrases, otherwise it can have negative affect on the classifier model. Table 5.2 contains some examples of the slang expressions and their replacements.

Next step is to fix the possible punctuations between characters. For example, *H.E.L.P* will be replaced with *HELP* and *U.N.O* will be replaced with *UNO*. Regular expression is used for detecting
5.3. SYSTEM COMPONENTS

<table>
<thead>
<tr>
<th>Symbols and Codes</th>
<th>Unicode</th>
<th>Replacement</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>\uD83D \uDE12</td>
<td>:)</td>
<td>Sad emoji</td>
<td></td>
</tr>
<tr>
<td>\u2019</td>
<td>'</td>
<td>Single quotation</td>
<td></td>
</tr>
<tr>
<td>\u0022</td>
<td>&quot;</td>
<td>Double quotation</td>
<td></td>
</tr>
<tr>
<td>\u002c</td>
<td>,</td>
<td>Comma</td>
<td></td>
</tr>
<tr>
<td>\u0025</td>
<td>%</td>
<td>Percentage</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>HTML</th>
<th>Replacement</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;039;</td>
<td>'</td>
<td>Single quotation</td>
</tr>
<tr>
<td>&quot;</td>
<td>&quot;</td>
<td>Double quotation</td>
</tr>
<tr>
<td>&amp;</td>
<td>&amp;</td>
<td>Ampersand</td>
</tr>
<tr>
<td>&amp;064;</td>
<td>@</td>
<td>At</td>
</tr>
</tbody>
</table>

Table 5.1: Examples of Unicode and HTML symbols

<table>
<thead>
<tr>
<th>Slang Word</th>
<th>Replacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td>you</td>
</tr>
<tr>
<td>omg</td>
<td>oh my god</td>
</tr>
<tr>
<td>fyi</td>
<td>for your information</td>
</tr>
<tr>
<td>plz</td>
<td>please</td>
</tr>
</tbody>
</table>

Table 5.2: Examples of Slang Expressions

dots in the words as illustrated in the Listing 5.2, but before applying this regular expression it is checked if the character string is an email address or phone number, as in those cases punctuation marks are allowed in the characters.

```
1 public static final String ALTERNATING_LETTER_DOT = "\[a-zA-Z]\\.(?:[a-zA-Z]\(\s+\))?\d+";
2 public static final Pattern ALTERNATING_LETTER_DOT_PATTERN = Pattern.compile(  
   ALTERNATING_LETTER_DOT);  
```

Listing 5.2: Punctuation between character regular expression

Final task of the Preprocessor component is that it tries to fix gerund forms of the words as they are quite frequently used in the short text communication. For example, goin is the gerund form which will be replaced with going. Fixation of gerund forms is not perfect at the moment and can be improved further using a dictionary for comparing words and replacing only if the word is not in the dictionary and then again checking after the replacement.

Following is an example of the raw text messages and the resulting text message after the preprocessing:
Before Preprocessing:
Please see help! No 1 is comin to help us we need water & food. FYI we are 10 ppl stuck in delmas near sports stadium

After Preprocessing:
Please help! No 1 is coming to help us we need water & food. For your information we are 10 people stuck in delmas near sports stadiums

5.3.3 Instance

The Instance Bolt is the component which is responsible for converting the data from the SMS class to the WEKA compatible instance. An instance consists of two distinct sections, instance header and the instance data. Which we will understand in details in order to understand how instances are created.

WEKA’s instance interface follows the ARFF (Attribute Relation File Format) format that describes the set of attributes and the set of instances which need to be processed. ARFF format have two sections, Header and Data. The header section contains the name of the relation, list of the attributes and their types and the data section contains the actual data. Listing 5.3 illustrates an example of the ARFF format.

```plaintext
@RELATION haiti

@ATTRIBUTE sms STRING

@ATTRIBUTE class {Public Health, Medical Emergency, Vital Lines, ........ }

@DATA
"birthing clinic in jacmel haiti urgently needs a o- blood transfusion 4 woman who just gave birth. please see via @coreyrateau", "Public Health"
"we are still under the sheets. we do not have: tents, prelates, sanitary articles and household etc. bastien the city alix fontamara 27", "Vital Lines"
```

Listing 5.3: Example of ARFF Format

Generally for offline classification, a full dataset is converted to ARFF file format and processed but as in stream we handle one text message tuple at a time, the instance which will be created for the classifier will have header section and a single message tuple in the data section. In our use case, the first part of the instance i.e. header will be created in the topology initialization code and will be reused for all the instances in the bolt as it will be same for all the instances until topology is not stopped or refreshed. Listing 5.4 illustrates the code from the topology initialization class which
creates an instance header. First step is to create a list of WEKA’s Attribute class describing the attributes(lines 2,3,4) and then this attribute list is used to construct an instance Header instHeader which will be passed to the Instance bolt to be used with each incoming tuple.

Listing 5.4: Instance Header Code

Listing 5.5 shows the abbreviated code snippet from the Instance bolt which handles the main task of instance creation. Bolt constructor collects the instance header created in the topology initialization(line 6). Execute method, receives the incoming tuple which is in the form of SMS class, creates an instance object with the headers and set the values of the text message and the class label if available(lines 13-16). This instance object is then emitted further and the acknowledgment is sent to the previous bolt.

Listing 5.5: Abbreviated Code Snippet from the Instance Bolt
5.3.4 POS Tagger

The POS Tagger component determines the part-of-speech labels for the preprocessed instances. It receives the preprocessed tuples from the Preprocessor component. Apache OpenNLP is used in the system implementation. OpenNLP part-of-speech tagger tool uses maximum entropy and tries to predict whether the words in the tuple are nouns, verbs, or any of the other 70 tags depending on the context of the message. Maxent model is used for the POS tagger. OpenNLP POS Tagger uses Penn Treebank (PTB) tagset. Some examples of PTB tagset are illustrated in Table 5.3. Full PTB tagset is in the Table A.1 in the Appendix A.

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FW</td>
<td>Foreign Word</td>
</tr>
<tr>
<td>JJ</td>
<td>Adjective</td>
</tr>
<tr>
<td>NNS</td>
<td>Noun Singular</td>
</tr>
<tr>
<td>SYM</td>
<td>Symbol</td>
</tr>
<tr>
<td>VB</td>
<td>Verb, Base Form</td>
</tr>
</tbody>
</table>

Table 5.3: Examples of PTB Tagset

Listing 5.6 displays the code snippet of the POS Tagger component. On the initialization, POS tagger is created using the model file (lines 9-15). The function getTaggerWords is used to determine the tags using the model created while initializing the POS Tagger class. First it create a list of raw words from the text string and then uses these raw words to find the tags (line 40). Finally the words and the tags are clubbed together using the instances of TaggerWord class which is a user-defined class containing a word and the corresponding tag.

```java
public class POSTagger {
    private static final Logger LOGGER = Logger.getLogger(POSTagger.class);
    private static POSTagger instance;
    private HashSet<String> excludedTags;
    private POSTaggerME tagger;

    private POSTagger(String modelFile, HashSet<String> excludedTags) {
        InputStream modelIn = null;
        try {
```
modelIn = new FileInputStream(modelFile);
POSModel model = new POSModel(modelIn);
tagger = new POSTaggerME(model);
if(excludedTags != null) {
    this.excludedTags = excludedTags;
} catch(Exception e){
    LOGGER.error("Error in POS Tagger: " + e.getMessage());
} finally {
    if(modelIn != null) {
        try{ modelIn.close(); }
        catch(IOException e){
        }
    }
}

public List<TaggedWord> getTaggerWords(StringReader reader) {
    List<TaggedWord> taggedWords = new ArrayList<TaggedWord>();
    StreamTokenizer streamTokenizer = null;
    ArrayList<String> rawWords = new ArrayList<String>();
    try {
        streamTokenizer = new StreamTokenizer(reader);
        while(streamTokenizer.nextToken() != StreamTokenizer.TT_EOF) {
            if(streamTokenizer.ttype == StreamTokenizer.TT_WORD) {
                rawWords.add(streamTokenizer.sval);
            }
        }
    } catch(IOException ex) {
        LOGGER.error("Exception in tokenizing: " + ex.getMessage());
    }
    String[] tags = tagger.tag(rawWords.toArray(new String[rawWords.size()]));
    for(int i=0; i<tags.length; i++) {
        String tag = tags[i];
        if((excludedTags != null) && excludedTags.contains(tag)) {
            continue;
        }
        taggedWords.add(new TaggedWord(rawWords.get(i), tags[i]));
    }
    return taggedWords;
}

Listing 5.6: POS Tagger Component (Abbreviated)

Listing 5.7 illustrates the snippet of POS Tagger Bolt which is receives the input data, convert it into a StringReader object and will pass to the tagger component(discussed in Listing 5.6) to get the list
of `TaggerWords` which is then emitted to the next bolt.

```java
sms inputPost = (sms)input.getValueByField("smsPost");
String content = inputPost.getContent();
StringReader reader = new StringReader(content.replace("\", ""));
List<TaggedWord> taggedWords = tagger.getTaggerWords(reader);
collector.emit(new Values(taggedWords, inputPost));
```

Listing 5.7: POS Tagger Bolt (Abbreviated)

### 5.3.5 String To Word Vector

`String To Word Vector` is the bolt which converts the string attributes of the instance to a set of vector attributes which represent word occurrences from the text content depending on the tokenizer used [63][64]. Listing 5.8 shows the initialization of the filter used and the word tokenizer. Training set is used to create first batch of vectors which will be used for training the model.

Lovins Stemmer algorithm [67] is used for stemming of the text words. It performs a lookup on a table of 294 endings, 29 conditions and 35 transformation rules based on the principle of longest match and removes the longest possible suffix from the word. It is a single pass algorithm i.e. it always removes a maximum of one suffix from a word which makes it a good choice for the streaming algorithms [68].

After stemmer declaration, all other properties used in the `String To Word Vector` object are similar to what we analyzed and discussed in the Section 4.3 of the Chapter 4. Only property which is not used from the list is `Minimum Term Frequency` as we are handling one instance at a time in the system and having a minimum term frequency of 5 for any word/term is very rare.

```java
private StringToWordVector filter = new StringToWordVector(MAX_NUMBER_OF_WORDS_TO_KEEP);
//Stemmer
LovinsStemmer stemmer = new LovinsStemmer();
//Tokenizer
WordTokenizer tokenizer = new WordTokenizer();
filter.setOutputWordCounts(true);
filter.setIDFTransform(true);
```
5.3. SYSTEM COMPONENTS

5.3.6 Classifier

The most important component of the topology pipeline is the Naive Bayes Multinomial classification algorithm. Naive Bayes Multinomial Classifier is discussed in detail in the section 4.2.2. As it is a supervised learning algorithm, it requires a set of training data with associated labels. The training data used for learning consists of feature word vectors which are generally a set of numerical values calculated using the String To Word Vector approach as explained in the section 5.3.5. Classification algorithm trains a model with the training data and then predicts the class labels for the instances of data items which follows. For the classifier, Naive Bayes Multinomial classification algorithm is supported by the Online OzaBoost[69] ensemble learning method. Ensemble learning methods provide very good performance but with the limitation that they were used in batch processing mode only. Oza et al. first implemented the online learning version of boosting method which only require one pass through the training and testing data [70]. This online implementation is present in the MOA algorithms [71] which was rewritten for this thesis work to accommodate a distributed setting and the sharing of the classifiers. It is largely derived from the MOA implementation of OzaBoostr.

Listing 5.9 illustrate the OzaBoost classifier class in the abbreviated form with only two important functions explained: the training function and the prediction function.

```java
public class OzaBoost extends AbstractClassifier {

    @Override
    public void trainOnInstanceImpl(Instance inst) {
        try {
            lock.acquire();
            Classifier newClassifier = ((Classifier) getPreparedClassOption(this.
            baseLearnerOption)).copy();
            ensemble.add(new ClassifierInstance(newClassifier));
            double lambda_d = 1.0;
            for(ClassifierInstance c : ensemble){
```

Listing 5.8: String To Word Vector Filter Code Snippet (Abbreviated)
double k = this.pureBoostOption.isSet() ? lambda_d : MiscUtils.
poisson(lambda_d, this.classifierRandom);

if (k > 0.0) {
Instance weightedInst = (Instance) inst.copy();
weightedInst.setWeight(inst.weight() * k);
c.getClassifier().trainOnInstance(weightedInst);
}

if (c.getClassifier().correctlyClassifies(inst)) {
c.setScms(c.getScms() + lambda_d);
lambda_d *= this.trainingWeightSeenByModel / (2 * c.getScms());
} else {
c.setSwms(c.getSwms() + lambda_d);
lambda_d *= this.trainingWeightSeenByModel / (2 * c.getSwms());
}

} catch (InterruptedException e) {
 e.printStackTrace();
} finally {
 lock.release();
 }

} catch (InterruptedException e) {
 e.printStackTrace();
} finally {
 lock.release();
 }

return combinedVote.getArrayRef();
}

Listing 5.9: Classifier Code (Abbreviated)
5.3.7 Statistics

The Statistics Bolt is the part which calculates the accuracy of the classifier system. This is required to check how efficient the classifier is on the dataset. For calculating the statistics, we use the class labels which are already present in the dataset. This will not be the case for the online dataset as we will not have the already labeled dataset to check the accuracy of the system. In the live system, this component can be modified to check the accuracy based on human annotated messages if needed.

If the text message instance is marked as not human annotated, then the predicted class (calculated from the votes by the prediction function in the Classifier bolt) is used for statistics calculations. On the other hand, if the instance is human annotated then the labels assigned by the volunteers are directly used as the class labels for statistics calculations. Accuracy and the Kappa are calculated using the following formulas:

\[
\text{accuracy} = \frac{\text{TotalCorrectlyPredictedInstances}}{\text{TotalInstances}} \quad (5.1)
\]

\[
\text{kappa} = \frac{\text{accuracy} - \text{RandomGuessAccuracy}}{1 - \text{RandomGuessAccuracy}} \quad (5.2)
\]

Lines 30 and 37 illustrate the calculations done for accuracy and kappa values. These values are then emitted to the next bolt(line 38). The instance is acknowledged to the previous bolt(line 27).

```java
@Override
public void execute(Tuple input) {
    sms inputPost = (sms) input.getValueByField("smsPost");
    int pred = 0;
    if (!inputPost.isHumanAnnotated()) {
        if (input.getValue(0).getClass() == double[].class){
            double[] dist = (double[]) input.getValueByField("votesForInstance");
            double max = -1;
            for (int i=0;i<dist.length;i++){
                if (dist[i]>max){
                    pred = i;
                    max = dist[i];
                }
            }
        }
    } else if (inputPost.isHumanAnnotated()) {
        pred = classes.indexOf(inputPost.getLabel());
    }
    int actual = classes.indexOf(inputPost.getLabel());
```
if (pred == actual) {
    totalPredictedCorrectly++;
}
stats[0][actual]++;
stats[1][pred]++;
totalCount++;
collector.ack(input);

if (totalCount % REPORTING_FREQUENCY == 0) {
    double accuracy = (double)totalPredictedCorrectly / (double)totalCount;
    double randomGuessAccuracy = 0;
    for (int i = 0; i < stats[0].length; i++) {
        randomGuessAccuracy += (stats[0][i] / totalCount) * (stats[1][i] / totalCount);
    }
    double kappa = (accuracy - randomGuessAccuracy) / (1 - randomGuessAccuracy);
    collector.emit(new Values(totalCount, accuracy, Double.isNaN(kappa) ? 0 : kappa, inputPost, classes.get(pred)));
}

Listing 5.10: Statistics Class (Abbreviated)

5.3.8 Stats Writer

Stats Writer Bolt is the component which is just used to write the accuracy and kappa stats calculated in the Statistics bolt to the output file. This final output is used for the evaluation of the system for its performance.

5.3.9 Database

Database bolt is responsible for establishing the connection to the database and performing all the storage related activities of the classified instances. MySQL is used as a database for the prototype system.

Two utilities classes are created for database related operations, first one is the MySQL Connection class which declare methods for establishing connection to the database. Second utility class declares the methods to perform database operations.
Listing 5.11 shows the database operation required for the MySQL database. Persist function performs the insertion operations for the instance and the predicted class. Similarly, storePOSTags function stores all the POS tags of the instance for further use. As a single instance will have many different POS tags, instead of inserting each tag individually a batch operation is used to insert all the tags related to an instance in a single go. This will save a lot of queries and will reduce the latency of the system as a result.

```java
public class MySqlOperations {
    private static final Logger LOG = Logger.getLogger(MySqlOperations.class);
    private MySqlConnection conn;

    public MySqlOperations(String ip, String database, String username, String password) {
        conn = new MySqlConnection(ip, database, username, password);
        conn.open();
    }

    public void persist(sms inputPost, String predictedClass) {
        PreparedStatement statement = null;
        try {
            statement = conn.getConnection().prepareStatement("INSERT INTO haiti_predicted_classes (INPUT_ID, PREDICTED_CLASS) values (?, ?)"");
            statement.setLong(1, inputPost.getInputId());
            statement.setString(2, predictedClass);
            statement.executeUpdate();
        } catch (Exception ex) {
            LOG.error("Error in inserting data: " + ex.getMessage());
        } finally {
            if (statement != null) {
                try {
                    statement.close();
                } catch (Exception ex) {
                    LOG.error(ex.getMessage());
                }
            }
        }
    }

    public void storePOSTags(sms inputPost, List<TaggedWord> taggedWords) {
        PreparedStatement statement = null;
        Connection connection = conn.getConnection();
        try {
            connection.setAutoCommit(false);
            statement = connection.prepareStatement("INSERT INTO haiti_postags (INPUT_ID, WORD, TAG) values (?, ?, ?)"");
            for (TaggedWord wordTag : taggedWords) {
                statement.setLong(1, inputPost.getInputId());
                statement.setString(2, wordTag.getWord());
                statement.setString(3, wordTag.getTag());
                statement.executeUpdate();
            }
        } catch (Exception ex) {
            LOG.error(ex.getMessage());
        } finally {
            if (statement != null) {
                try {
                    statement.close();
                } catch (Exception ex) {
                    LOG.error(ex.getMessage());
                }
            }
        }
    }
}
```
statement.setString(2, wordTag.getWord());
statement.setString(3, wordTag.getTag());
statement.addBatch();
)
statement.executeBatch();
connection.commit();
) catch(Exception ex) {
    LOG.error("Error in inserting data in table: " + ex.getMessage() );
}
) finally {
    if(statement != null) {
        try {
            statement.close();
        } catch(Exception ex) {
            LOG.error(ex.getMessage());
        }
    }
}
Listing 5.11: Database Operations Class (Abbreviated)
Chapter 6

Performance and Storm Evaluation

In this chapter, the approach and the prototype system will be evaluated. Evaluation is divided into two parts, a quality evaluation and a performance evaluation. The performance evaluation discusses the speed of the developed system on a single node and the quality evaluation tries to quantify the accuracy of the system with respect to the pre-labeled Haiti earthquake dataset we have.

6.1 Quality Evaluation

The output of the system is a predicted label for each instance, which is compared to the original class labels of the text messages and can be used to calculated the accuracy of the system.

The overall accuracy of the system represents the percentage of correctly predicted instances and the total number of instances in the dataset. The accuracy $A$ for $n$ classes relies on the confusion matrix $M$ and can be calculated by the Equation 6.1

$$A = \frac{\sum_{i=1}^{n} M(i,j)}{\sum_{i,j=1}^{n} M(i,j)}$$

As the algorithm implementation of the system is capable of active learning, prediction model which is prepared from the initial training data will keep on learning when more human annotated messages arrive even when the model is actively predicting the message labels. For accuracy measure we started with measuring the accuracy of the system with some initial training data only and without any further training data arriving continuously, after the first measure we will test the accuracy involving continuous training data also.

Figure 6.1 illustrates the accuracy of the system with different sizes of training dataset. All the training dataset for this measure is provided at the start of the system and there is no further
training data supplied on which the model can train upon. As it is evident from the figure, with increasing size of training data the overall accuracy of the system increases. With just 100 messages in the training set, the system can achieve the accuracy of approximately 23.63% which increases to almost 56.34%-57.47% with the training set of 700-800 messages respectively. With 1000 messages in the training dataset, the system can achieve the accuracy of almost 62.05%. If the system is trained with 1500 and 2000 messages, accuracy can reach up to 72.08% and 79.13% respectively.

We saw that with a fixed initial set of training data we can reach an acceptable accuracy. But in disaster scenarios where time is really an important factor and volunteers may not be able to wait for long time before the system can be implemented and sufficient amount of messages can be human annotated. One feasible scenario is to start the system with a very small training dataset to train and then once the system is online, it can be fed with more human annotated messages as volunteer finish tagging new messages. So, for the next evaluation trial we will measure the accuracy with few messages in initial training set followed by a continuously sending human annotated messages later in the stream at different levels.

Figure 6.1 illustrates the accuracy graph for the whole dataset with message instance ids on x-axis and accuracy on y-axis. Mode used for the accuracy calculation is trained with initial training set of 10, 20, 50 and 100 messages followed by the stream of messages of which 1% are human
annotated. As evident from the figure, while training the accuracy is the highest and it falls to the lowest as soon as it starts predicting. The stream following the training steam contains 1% human annotated messages, model keep on training on these message instances and the accuracy increases as more and more human labeled messages arrive. Prediction model reaches the highest accuracy at around 2000 messages are processed of which 1% are pre-labeled. With 100 messages in initial training set, model reaches the highest accuracy of around 56.2% and then settles to around 43.2% accuracy till the end of the dataset. Similarly with the training set of 10 messages and 1% continuous training messages, it reaches the highest accuracy of around 62.0% and settles at around 47.66%. Similar trend is with the training set of 20 and 50 which reaches the highest accuracy of 53% and 50.6% before finally settling down to 41.13% and 41.34% respectively.

Figure 6.2b illustrates the similar trend. Where maximum 68.45% accuracy is achieved with initial training data of 10 messages and settles down at the 53.77% accuracy. Figure 6.2c represents the trend with 10% human annotated messages in the stream by which highest accuracy of approx. 70.45% and final accuracy of 56.3% can be achieved.

Figure 6.2d which illustrates 20% human annotated messages performs the best for obvious reasons that it has sufficient data to learn from. Maximum accuracy achieved with 20% human annotation is 78.6% and final is 65.34% respectively.

Here, we see that with increasing the percentage of human annotated messages from 1% to 20% improve the accuracy of the system from 43.2% final accuracy to 65.34% final accuracy. So we can conclude with small amount of initial training data followed by continuous annotation of more and more instances by the volunteers, acceptable accuracy can be achieved and in this way system can also be in working state in a short period of time without much wait. Moreover before the volunteer burnout starts, system will be performing with the good accuracy.

### 6.2 Performance Evaluation

The performance evaluation analyzes the system’s processing capabilities and how much data the system can process. Latency is one of the factors which informs us about the processing capabilities of a system. It is the time taken by the system or the component of the system in processing a single unit which in our use case is text messages. So here we will evaluate the latency of the system from end-to-end as well as the latency of each component. The specifications of the system used for the evaluation purpose is illustrated in Table 6.1. Evaluation tests are performed on stand-alone mode of Storm on a single system. If the system can be implemented in a cluster of machines the performance of the system will increase a lot.

Figure 6.3 illustrates the end-to-end latency of the system with different number of messages input to the system with different number of workers. We choose to test with the number of workers 1, 2 and 4. With 1000 input messages at an instance results in latency of 57.83ms with worker 1,
CHAPTER 6. PERFORMANCE AND STORM EVALUATION

(a) 1 percent continuous human annotates instances  
(b) 5 percent continuous human annotates instances  
(c) 10 percent continuous human annotates instances  
(d) 20 percent continuous human annotates instances

Figure 6.2: Accuracy achieved by all the individual instances with the varying initial training data followed by continuous human annotated instances

<table>
<thead>
<tr>
<th>CPU</th>
<th>Intel Pentium</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory</td>
<td>4GB</td>
</tr>
<tr>
<td>OS</td>
<td>Windows 7</td>
</tr>
<tr>
<td>Processor</td>
<td>64 Bit</td>
</tr>
<tr>
<td>Clock Speed</td>
<td>2.13 GHz</td>
</tr>
</tbody>
</table>

Table 6.1: System specifications
6.2. PERFORMANCE EVALUATION

Figure 6.3: End-to-end latency of the system

68.76ms with 2 workers and 61.59ms with 4 workers. We get the best latency for 6000 messages at a time. This gives the latency of 25.62ms with 1 worker, 52.01ms and 52.43ms with 2 and 4 workers respectively. Another thing worth mentioning is if we increase the number of messages further to 7000 and onward. The system starts dropping the messages which it is not able to process. With 1 worker and 7000 input messages, 2713 messages were dropped unprocessed. Dropping rate with 2 workers is still 0 with 7000 messages but 3067 messages were dropped while using 4 workers. And this number of messages dropped increases further when we increase the size of input data. For 10,000 messages, system drops 5380, 5275 and 2519 with workers 1, 2 and 4 respectively.

Further we will discuss the latency of individual components of the system. As discussed in the Chapter 5 the developed system consists of 9 different components. Figure 6.4 and Figure 6.5 illustrates the latency of all 9 components with the same setup of workers as well as input data size. Besides the illustrative figures in this chapter, latency values in the form of tables are present in Appendix B in Tables B.1, B.2 and B.3.

For 6000 messages input (which gives the best performance), the latency of the Preprocessing component is recorded at 9.39ms with 1 worker, 11.41ms and 21.87ms with 2 and 4 workers respectively. For Instance Bolt component, it is 9.24ms for 1 worker and for 2 and 4 workers it drastically increases to 25.27ms and 21.37ms. String to Word Vector performance is quite similar to
the *Instance Bolt* component with 8.89ms, 25.25ms and 21.36ms for 1, 2 and 4 workers respectively. *String to Word Vector* component as well as *Instance Bolt* records the best possible performance latency wise with 1 worker. *Classifier* component’s performance is similar to that of *Preprocessing* component with not much variance in the latency values, it logs the latency at 17.22ms with 1 worker, 23.62ms with 2 workers and 19.37 with 4 workers.

Figure 6.5 illustrates the latency graph of the remaining 5 components. For 6000 message inputs, *Statistics Bolt* component, which is used for calculating the relevant statistics of the system records the latency of 17.22 with 1 worker, 23.61 with 2 workers and 22.03 with 4 workers. There is not much different in the latency of the statistics component with different number of workers. *Statistic writer* is one of the component which records the most variation in the latency of the system with different workers. It logs the latency of 2.69 with one worker, 23.55 with 2 workers and 21.94 with 4 workers. *POS Tagger* component registers the latency of 19.69, 25.43 and 25.40 with 1, 2 and 4 workers.
6.2. PERFORMANCE EVALUATION

(a) Statistics Component
(b) MySQL Component 1
(c) StatsWriter Component
(d) POS Tagger Component
(e) MySQL Component 2

Figure 6.5: Latency of the different components of the system
workers respectively. The most extensive operation being the storage of the output to the MySQL persistent storage. MySQL components are the major contributing components towards the overall latency as they are the slowest and affect the system negatively with latency between almost 47ms and 51ms with multiple workers.

6.3 Storm evaluation

Evaluation of Storm as the processing engine is also important. In this section, we will evaluate the Storm engine as a whole. Storm is really easy to setup and run a topology. Moreover, a lot of sample code is available with the setup. For example, storm-starter package that contains some basic topology configuration etc. It gives a good start to understand the working of topologies on Storm. Moreover, the community is large and very active which helps in case any issues are encountered.

One strength of storm is the use of permissive data model. A tuple is a named list of values and a value can be an object of any type. Storm also supports transactional topologies that guaranteed that tuples will be processed exactly once. Storm is already is production use in many companies which proves the stability of the whole system in the production environments. Storm is also fault-tolerant which is very important for the scenarios like disaster response management. Upon failure, workers restart and tasks are reassigned to the workers to continue the processing.

One of the weaknesses of Storm if that it is hard to find the precise configuration for the system which will perform best for all scenarios. It can be tricky to find the best possible configuration and sometimes trial and error method need to be used to figure out the best suitable configuration. Another weakness of Storm is that it cannot dynamically scale and scalability need to be handled by the administrator handling the system.
Chapter 7

Conclusion and Future Work

In this chapter, the work done in this thesis will be concluded. First a short reflection will be given in which the overall work will be summarized. Subsequently limitations of the work will be highlighted and finally the chapter finishes with the discussion about the unsolved issues and a direction for future work will be proposed.

7.1 Reflections

In this thesis work, we examined the problem of large scale text data processing on the continuous stream. Our main focus was on text messages sent by the affected people for help or information during the humanitarian disasters. The main issue with this type of data is that the large amount of data may arrive at high speed during crises. To handle the dynamic change in the incoming data and to provide fault-tolerance Apache Storm is used and the prototype system is developed in accordance with the Storm topology. Which basically works with distributing the work in multiple parts and building a pipeline for data processing. Ingestion of Apache Storm in the system as the base platform solves the goal of building the solution which can handle large amount of data in near real-time.

The text message communication is generally short and limited in words, contains typographical errors and slang words etc. To overcome this we proposed an approach which can improve the messages with typographical correction, HTML parsing, slang word correction and stemming etc. using slang dictionaries and the word dictionaries. This processing of the messages is done in the initial stage of the pipeline before the message instances can be passed to the prediction modules.

Another challenge specially in crisis informatics is that the methods or algorithms implemented should be aware of the particularities of the domain. This is the reason why a pre-trained model on some past crises or disasters cannot be used for the future [25]. For each disaster or crises a new
annotation is needed and this can be done extremely well by human resources due to the human adaptability and intelligence. So the systems should be able to take input from human experts to learn. To address this issue, we used a new class of systems which are called *Crowdsourced Stream Processing Systems* which make use of the human intelligence to judge the text messages to learn and then automate this process for further processing on a distributed stream processor. Another focus of the work is to figure out how the system can work efficiently with as less human annotated data as possible to provide acceptable results. We theoretically examined several aspects of the data that can affect the performance of the system and did preliminary analysis on the available dataset in order to figure out the best possible approach to implement in the system. After preliminary analysis, we concluded the *Naive Bayes Multinomial Algorithm* will be a best fit for such data and also can be implemented as an active learning algorithm which will keep on continuously learning to improve the performance.

We can conclude that the prototype system is capable of learning from very small training data to start with and keep on learning as the more and more human annotated is received. We can also conclude that the system is capable of delivering the acceptable results both in terms of accuracy as well as the speed of data processing.

### 7.2 Limitations

As discussed in the *Introduction* and Preliminary Analysis chapters, the work focused exclusively on the text messages which can be received during and after the disaster for the rescue and recovery work. Disaster response is complex and in practice this channel is merely one of several from which aid organizations collect and aggregate information. This means that the proposed technique is also only useful during disaster in which telephonic communication is possible or can be re-established easily after the impact.

### 7.3 Future Work

Due to the time restrictions of this work, there are some issues which remain unexplored and can be addressed in the future work. One such open issue being the implementation of a user interface which can provide the outcome of the system to the users. There can be an administration panel which can provide the require input on which the system will work. For example, providing the list of classes which system need to use for the classification approach.

Another possible extension could be to enrich the proposed topology with more functionality in order to enrich the message text which can help and further improve the accuracy of the system. The backend persistent storage used in the proposed prototype is MySQL database which have a
negative effect on the latency of the system. Another databases, possible NoSQL databases like Cassandra or HBase can be evaluated with the topology if it can improve the latency of the system further.

Another area of focus for future work is towards the tight integration of the crowdsourcing platforms with the system and to figure out the best possible way to assign work to the volunteers. Work need to be done about how the messages should be allocated to the crowd for annotation so that as much diversity of the messages as possible can be addressed in the training set and system should be able to learn all different aspects of the scenario.

Finally, the proposed system topology can be tested with another dataset or dynamic changing data to validate the usefulness of the proposed approach. Right now, due to scarcity of the data, this is not possible. But as with the trial implementation of TextTo911 more data may be available in public domain for the research purpose in the near future which can be used for the purpose.
Bibliography


# Appendix A

## Penn Treebank POS Tags

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC</td>
<td>Coordination conjunction</td>
</tr>
<tr>
<td>CD</td>
<td>Cardinal number</td>
</tr>
<tr>
<td>DT</td>
<td>Determiner</td>
</tr>
<tr>
<td>EX</td>
<td>Existential there</td>
</tr>
<tr>
<td>FW</td>
<td>Foreign word</td>
</tr>
<tr>
<td>IN</td>
<td>Preposition or subordinating conjunction</td>
</tr>
<tr>
<td>JJ</td>
<td>Adjective</td>
</tr>
<tr>
<td>JJR</td>
<td>Adjective, comparative</td>
</tr>
<tr>
<td>JJS</td>
<td>Adjective, superlative</td>
</tr>
<tr>
<td>LS</td>
<td>List item marker</td>
</tr>
<tr>
<td>MD</td>
<td>Modal</td>
</tr>
<tr>
<td>NN</td>
<td>Noun, singular or mass</td>
</tr>
<tr>
<td>NNS</td>
<td>Noun, plural</td>
</tr>
<tr>
<td>NNP</td>
<td>Proper noun, singular</td>
</tr>
<tr>
<td>NNPS</td>
<td>Proper noun, plural</td>
</tr>
<tr>
<td>PDT</td>
<td>Predeterminer</td>
</tr>
<tr>
<td>POS</td>
<td>Possessive ending</td>
</tr>
<tr>
<td>PRP</td>
<td>Personal pronoun</td>
</tr>
<tr>
<td>PRP$</td>
<td>Possessive pronoun</td>
</tr>
<tr>
<td>RB</td>
<td>Adverb</td>
</tr>
<tr>
<td>RBR</td>
<td>Adverb, comparative</td>
</tr>
<tr>
<td>RBS</td>
<td>Adverb, superlative</td>
</tr>
<tr>
<td>RP</td>
<td>Particle</td>
</tr>
<tr>
<td>SYM</td>
<td>Symbol</td>
</tr>
<tr>
<td>Tag</td>
<td>Description</td>
</tr>
<tr>
<td>-----</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>TO</td>
<td>to</td>
</tr>
<tr>
<td>UH</td>
<td>Interjection</td>
</tr>
<tr>
<td>VB</td>
<td>Verb, base form</td>
</tr>
<tr>
<td>VBD</td>
<td>Verb, past tense</td>
</tr>
<tr>
<td>VBG</td>
<td>Verb, gerund or present participle</td>
</tr>
<tr>
<td>VBN</td>
<td>Verb, past participle</td>
</tr>
<tr>
<td>VBP</td>
<td>Verb, non-3rd person singular present</td>
</tr>
<tr>
<td>VBZ</td>
<td>Verb, 3rd person singular present</td>
</tr>
<tr>
<td>WDT</td>
<td>Wh-determiner</td>
</tr>
<tr>
<td>WP</td>
<td>Proper</td>
</tr>
<tr>
<td>WP$</td>
<td>Possessive wh-pronoun</td>
</tr>
<tr>
<td>WRB</td>
<td>Wh-adverb</td>
</tr>
</tbody>
</table>

Table A.1: Penn Treebank Part-Of-Speech (POS) tags [1]
Appendix B

Latency Measures

B.1 Worker 1

Latency of the overall system from end-to-end as well as the latency of each and each component is illustrated in table in details with system configured to use 1 worker. Table also shows the number of messages dropped while running the topology with different number of messages.

<table>
<thead>
<tr>
<th>Components</th>
<th>1000</th>
<th>2000</th>
<th>3000</th>
<th>4000</th>
<th>5000</th>
<th>6000</th>
<th>7000</th>
<th>8000</th>
<th>9000</th>
<th>10000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance</td>
<td>15.92</td>
<td>12.15</td>
<td>11.24</td>
<td>9.74</td>
<td>10.30</td>
<td>9.24</td>
<td>5.30</td>
<td>5.36</td>
<td>4.43</td>
<td>4.30</td>
</tr>
<tr>
<td>StringToWordVector</td>
<td>15.70</td>
<td>11.79</td>
<td>11.02</td>
<td>9.57</td>
<td>9.89</td>
<td>8.89</td>
<td>5.30</td>
<td>5.33</td>
<td>4.42</td>
<td>4.20</td>
</tr>
<tr>
<td>OzaBoost</td>
<td>5.01</td>
<td>8.02</td>
<td>7.70</td>
<td>7.88</td>
<td>8.40</td>
<td>17.22</td>
<td>4.03</td>
<td>4.07</td>
<td>3.07</td>
<td>3.33</td>
</tr>
<tr>
<td>Statistics</td>
<td>4.25</td>
<td>7.98</td>
<td>10.00</td>
<td>23.29</td>
<td>8.89</td>
<td>17.22</td>
<td>4.19</td>
<td>10.24</td>
<td>7.26</td>
<td>4.29</td>
</tr>
<tr>
<td>MySQLBolt1</td>
<td>42.02</td>
<td>41.47</td>
<td>40.99</td>
<td>28.18</td>
<td>38.30</td>
<td>17.15</td>
<td>27.90</td>
<td>16.78</td>
<td>14.09</td>
<td>21.15</td>
</tr>
<tr>
<td>StatWriter</td>
<td>3.63</td>
<td>7.82</td>
<td>9.98</td>
<td>5.08</td>
<td>8.90</td>
<td>2.69</td>
<td>4.17</td>
<td>2.28</td>
<td>1.74</td>
<td>4.29</td>
</tr>
<tr>
<td>POSTagger</td>
<td>14.20</td>
<td>12.22</td>
<td>10.27</td>
<td>22.33</td>
<td>22.19</td>
<td>19.69</td>
<td>17.30</td>
<td>17.21</td>
<td>13.21</td>
<td>15.24</td>
</tr>
<tr>
<td>MySQLBolt2</td>
<td>55.36</td>
<td>49.48</td>
<td>45.12</td>
<td>40.90</td>
<td>44.43</td>
<td>25.12</td>
<td>35.99</td>
<td>30.08</td>
<td>25.57</td>
<td>26.82</td>
</tr>
<tr>
<td>Messages dropped</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2713</td>
<td>3509</td>
<td>3442</td>
<td>5380</td>
<td></td>
</tr>
<tr>
<td>Total Latency</td>
<td>57.83</td>
<td>50.65</td>
<td>46.15</td>
<td>41.76</td>
<td>45.11</td>
<td>25.62</td>
<td>36.18</td>
<td>30.51</td>
<td>25.87</td>
<td>27.26</td>
</tr>
</tbody>
</table>

Table B.1: Latency of different components with worker 1

B.2 Workers 2

Table illustrates the latency of the system when setup with 2 workers. Latency of the overall system as well as the latency of each and each component is shown along with the number of messages dropped while running the topology with different number of messages.
APPENDIX B. LATENCY MEASURES

Table B.2: Latency of different components with workers 2

<table>
<thead>
<tr>
<th></th>
<th>1000</th>
<th>2000</th>
<th>3000</th>
<th>4000</th>
<th>5000</th>
<th>6000</th>
<th>7000</th>
<th>8000</th>
<th>9000</th>
<th>10000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preprocessor</td>
<td>17.74</td>
<td>11.53</td>
<td>10.23</td>
<td>10.58</td>
<td>9.34</td>
<td>11.41</td>
<td>10.41</td>
<td>7.01</td>
<td>34.62</td>
<td>28.60</td>
</tr>
<tr>
<td>Instance</td>
<td>17.03</td>
<td>11.40</td>
<td>10.04</td>
<td>9.88</td>
<td>8.81</td>
<td>25.27</td>
<td>9.69</td>
<td>6.61</td>
<td>34.42</td>
<td>28.53</td>
</tr>
<tr>
<td>StringToWordVector</td>
<td>16.87</td>
<td>11.19</td>
<td>9.94</td>
<td>8.86</td>
<td>8.80</td>
<td>25.25</td>
<td>9.57</td>
<td>6.49</td>
<td>34.27</td>
<td>28.53</td>
</tr>
<tr>
<td>OzaBoost</td>
<td>4.65</td>
<td>6.84</td>
<td>6.80</td>
<td>8.01</td>
<td>6.73</td>
<td>23.61</td>
<td>9.03</td>
<td>5.43</td>
<td>33.32</td>
<td>27.22</td>
</tr>
<tr>
<td>Statistics</td>
<td>4.34</td>
<td>6.76</td>
<td>20.56</td>
<td>34.63</td>
<td>8.91</td>
<td>23.61</td>
<td>9.02</td>
<td>16.05</td>
<td>33.30</td>
<td>27.20</td>
</tr>
<tr>
<td>MySQLBolt1</td>
<td>51.94</td>
<td>49.71</td>
<td>51.95</td>
<td>50.4</td>
<td>40.30</td>
<td>47.31</td>
<td>39.23</td>
<td>27.63</td>
<td>46.39</td>
<td>42.64</td>
</tr>
<tr>
<td>StatWriter</td>
<td>4.34</td>
<td>6.37</td>
<td>20.52</td>
<td>34.64</td>
<td>8.89</td>
<td>23.55</td>
<td>9.01</td>
<td>16.00</td>
<td>33.29</td>
<td>27.13</td>
</tr>
<tr>
<td>POSTagger</td>
<td>12.89</td>
<td>10.17</td>
<td>15.23</td>
<td>20.96</td>
<td>20.62</td>
<td>25.43</td>
<td>16.71</td>
<td>15.45</td>
<td>34.43</td>
<td>37.15</td>
</tr>
<tr>
<td>MySQLBolt2</td>
<td>63.25</td>
<td>56.26</td>
<td>57.93</td>
<td>55.36</td>
<td>44.73</td>
<td>51.33</td>
<td>42.65</td>
<td>31.47</td>
<td>50.91</td>
<td>47.55</td>
</tr>
</tbody>
</table>

Table B.3: Latency of different components with workers 4

<table>
<thead>
<tr>
<th></th>
<th>1000</th>
<th>2000</th>
<th>3000</th>
<th>4000</th>
<th>5000</th>
<th>6000</th>
<th>7000</th>
<th>8000</th>
<th>9000</th>
<th>10000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preprocessor</td>
<td>15.35</td>
<td>12.89</td>
<td>21.78</td>
<td>23.78</td>
<td>28.05</td>
<td>21.87</td>
<td>22.71</td>
<td>19.72</td>
<td>33.07</td>
<td>28.31</td>
</tr>
<tr>
<td>StringToWordVector</td>
<td>14.98</td>
<td>11.80</td>
<td>20.86</td>
<td>23.44</td>
<td>27.95</td>
<td>21.36</td>
<td>22.23</td>
<td>19.36</td>
<td>44.74</td>
<td>28.13</td>
</tr>
<tr>
<td>Statistics</td>
<td>5.51</td>
<td>6.01</td>
<td>35.02</td>
<td>19.53</td>
<td>27.86</td>
<td>22.03</td>
<td>22.39</td>
<td>17.78</td>
<td>43.53</td>
<td>27.09</td>
</tr>
<tr>
<td>MySQLBolt1</td>
<td>49.04</td>
<td>55.96</td>
<td>55.54</td>
<td>60.29</td>
<td>48.69</td>
<td>47.47</td>
<td>45.48</td>
<td>36.77</td>
<td>53.66</td>
<td>53.24</td>
</tr>
<tr>
<td>StatWriter</td>
<td>4.99</td>
<td>5.85</td>
<td>34.96</td>
<td>19.44</td>
<td>27.86</td>
<td>21.94</td>
<td>22.39</td>
<td>17.77</td>
<td>43.50</td>
<td>27.03</td>
</tr>
<tr>
<td>POSTagger</td>
<td>14.11</td>
<td>11.17</td>
<td>19.82</td>
<td>38.14</td>
<td>27.82</td>
<td>25.40</td>
<td>30.11</td>
<td>19.38</td>
<td>44.95</td>
<td>34.11</td>
</tr>
<tr>
<td>MySQLBolt2</td>
<td>60.24</td>
<td>63.97</td>
<td>63.91</td>
<td>66.96</td>
<td>54.64</td>
<td>51.63</td>
<td>51.05</td>
<td>40.00</td>
<td>60.29</td>
<td>58.31</td>
</tr>
</tbody>
</table>

B.3 Workers 4

Table illustrates the latency of the system when setup with 4 workers. Latency of the overall system as well as the latency of each and each component is shown along with the number of messages dropped while running the topology with different number of messages.

<table>
<thead>
<tr>
<th></th>
<th>1000</th>
<th>2000</th>
<th>3000</th>
<th>4000</th>
<th>5000</th>
<th>6000</th>
<th>7000</th>
<th>8000</th>
<th>9000</th>
<th>10000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preprocessor</td>
<td>15.35</td>
<td>12.89</td>
<td>21.78</td>
<td>23.78</td>
<td>28.05</td>
<td>21.87</td>
<td>22.71</td>
<td>19.72</td>
<td>33.07</td>
<td>28.31</td>
</tr>
<tr>
<td>StringToWordVector</td>
<td>14.98</td>
<td>11.80</td>
<td>20.86</td>
<td>23.44</td>
<td>27.95</td>
<td>21.36</td>
<td>22.23</td>
<td>19.36</td>
<td>44.74</td>
<td>28.13</td>
</tr>
<tr>
<td>Statistics</td>
<td>5.51</td>
<td>6.01</td>
<td>35.02</td>
<td>19.53</td>
<td>27.86</td>
<td>22.03</td>
<td>22.39</td>
<td>17.78</td>
<td>43.53</td>
<td>27.09</td>
</tr>
<tr>
<td>MySQLBolt1</td>
<td>49.04</td>
<td>55.96</td>
<td>55.54</td>
<td>60.29</td>
<td>48.69</td>
<td>47.47</td>
<td>45.48</td>
<td>36.77</td>
<td>53.66</td>
<td>53.24</td>
</tr>
<tr>
<td>StatWriter</td>
<td>4.99</td>
<td>5.85</td>
<td>34.96</td>
<td>19.44</td>
<td>27.86</td>
<td>21.94</td>
<td>22.39</td>
<td>17.77</td>
<td>43.50</td>
<td>27.03</td>
</tr>
<tr>
<td>POSTagger</td>
<td>14.11</td>
<td>11.17</td>
<td>19.82</td>
<td>38.14</td>
<td>27.82</td>
<td>25.40</td>
<td>30.11</td>
<td>19.38</td>
<td>44.95</td>
<td>34.11</td>
</tr>
<tr>
<td>MySQLBolt2</td>
<td>60.24</td>
<td>63.97</td>
<td>63.91</td>
<td>66.96</td>
<td>54.64</td>
<td>51.63</td>
<td>51.05</td>
<td>40.00</td>
<td>60.29</td>
<td>58.31</td>
</tr>
</tbody>
</table>

Table B.3: Latency of different components with workers 4