A Large Collection Learning Optimizer Framework

Saurabh Chakravarty

Thesis submitted to the Faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE
in
Computer Science and Applications

Edward A. Fox, Chair
Weiguo Fan
Bert Huang

May 1, 2017
Blacksburg, Virginia 24061

Keywords: Digital Libraries, Text Classification, Tweets, Apache Spark

A Large Collection Learning Optimizer Framework

Saurabh Chakravarty

Abstract

Content is generated on the web at an increasing rate. The type of content varies from text on a traditional webpage to text on social media portals (e.g., social network sites and microblogs). One such example of social media is the microblogging site Twitter. Twitter is known for its high level of activity during live events, natural disasters, and events of global importance.

Challenges with the data in the Twitter universe include the limit of 140 characters on the text length. Because of this limitation, the vocabulary in the Twitter universe includes short abbreviations of sentences, emojis, hashtags, and other non-standard usage. Consequently, traditional text classification techniques are not very effective on tweets. Fortunately, sophisticated text processing techniques like cleaning, lemmatizing, and removal of stop words and special characters will give us clean text which can be further processed to derive richer word semantic and syntactic relationships using state of the art feature selection techniques like Word2Vec. Machine learning techniques, using word features that capture semantic and context relationships, can be of benefit regarding classification accuracy.

Improving text classification results on Twitter data would pave the way to categorize tweets relative to human defined real-world events. This would allow diverse stakeholder communities to interactively collect, organize, browse, visualize, analyze, summarize, and explore content and sources related to crises, disasters, human rights, inequality, population growth, resiliency, shootings, sustainability, violence, etc. Having the events classified into different categories would help us study causality and correlations among real world events.

To check the efficacy of our classifier, we would compare our experimental results with an Association Rules (AR) classifier. This classifier composes its rules around the most discriminating words in the training data. The hierarchy of rules, along with an ability to tune to a support threshold, makes it an effective classifier for scenarios where short text is involved.

Traditionally, developing classification systems for these purposes requires a great degree of human intervention. Constantly monitoring new events, and curating training and validation sets, is tedious and time intensive. Significant human capital is required for such annotation endeavors. Also, involved efforts are required to tune the classifier for best performance. Developing and tuning classifiers manually using human intervention would not be a viable option if we are to monitor events and trends in real-time. We want to build a framework that would require very little human intervention to build and choose the best among the available performing classification techniques in our system.

Another challenge with classification systems is related to their performance with unseen data. For the classification of tweets, we are continually faced with a situation where a given event contains a certain keyword that is closely related to it. If a classifier, built for a particular event, due to overfitting to what is a biased sample with limited generality, is faced with new tweets with different keywords, accuracy may be reduced. We propose building a system that will use very little training data in the initial iteration and will be augmented with automatically labelled training data from a collection that stores all the incoming tweets. A system that is trained on incoming tweets that are labelled using sophisticated techniques based on rich word vector representation would perform better than a system that is trained on only the initial set of tweets.

We also propose to use sophisticated deep learning techniques like Convolutional Neural Networks (CNN) that can capture the combination of the words using an n-gram feature representation. Such sophisticated feature representation could account for the instances when the words occur together.

We divide our case studies into two phases: preliminary and final case studies. The preliminary case studies focus on selecting the best feature representation and classification methodology out of the AR and the Word2Vec based Logistic Regression classification techniques. The final case studies focus on developing the augmented semi-supervised training methodology and the framework to develop a large collection learning optimizer to generate a highly performant classifier.

For our preliminary case studies, we are able to achieve an F1 score of 0.96 that is based on Word2Vec and Logistic Regression. The AR classifier achieved an F1 score of 0.90 on the same data.

For our final case studies, we are able to show improvements of F1 score from 0.58 to 0.94 in certain cases based on our augmented training methodology. Overall, we see improvement in using the augmented training methodology on all datasets.

A Large Collection Learning Optimizer Framework

Saurabh Chakravarty

General Audience Abstract

Content is generated on social media at a very fast pace. Social media content in the form of tweets that is generated by the microblog site Twitter is quite popular for understanding the events and trends that are prevalent at a given point of time across various geographies. Categorizing these tweets into their real-world event categories would be useful for researchers, students, academics and the government. Categorizing tweets to their real-world categories is a challenging task. Our framework involves building a classification system that can learn how to categorize tweets for a given category if it is provided with a few samples of the relevant and non-relevant tweets. The system retrieves additional tweets from an auxiliary data source to further learn what is relevant and irrelevant based on how similar a tweet is to a positive example. Categorizing the tweets in an automated way would be useful in analyzing and studying the events and trends for past and future real-world events.

Acknowledgements

I would like to acknowledge and thank the following for assisting and supporting me throughout this thesis.

- Drs. Edward A. Fox, Weiguo Fan, Bert Huang, Denilson Alves Pereira
- NSF grant IIS - 1619028, III: Small: Collaborative Research: Global Event and Trend Archive Research (GETAR)
- NSF grant IIS - 1319578, III: Small: Integrated Digital Event Archiving and Library (IDEAL)
- Digital Library Research Laboratory
- Eric Williamson – Fall 2016 CS 5604 project team member and continuing collaborator
- Graduate Research Assistant – Dr. Sunshin Lee
- All teams in the Fall 2016 class for CS 5604 (Information Retrieval)

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Background

The advent of the World Wide Web has changed the rate of information generation and the way people access that information. The type of content on the Web varies from text on a traditional webpage to text on social media portals (e.g., social network sites, microblogs). One such example of social media is the microblogging site Twitter. Twitter is known for its high level of activity during live events, natural disasters, and events of global importance.

The users are not only just consuming the content being generated, but are also propagating that information across their own social network and in some instances, adding their own take on the content. Twitter is one such example of social media. Mining data from Twitter could be useful to understand the important global events and trends of the past decade.

Improving text classification results on Twitter data would pave the way to categorize tweets relative to human defined real world events. This would allow diverse stakeholder communities to interactively collect, organize, browse, visualize, study, analyze, summarize, and explore content and sources related to biodiversity, climate change, crises, disasters, elections, energy policy, environmental policy/planning, geospatial information, green engineering, human rights, inequality, migrations, nuclear power, population growth, resiliency, shootings, sustainability, violence, etc.

We intend to use sophisticated feature representation and classification techniques to help classify the tweet collections in our digital library. Having the collections labelled would help us organize and cluster the tweets further to identify and develop hierarchies for these collections.

1.1.1 Study of Events and Trends in Social Media Data

Following are a few examples of real-world scenarios for which we would be interested in analyzing the tweets.

- *Scenario 1: Financial Trends* - A financial disaster hits the economy, similar to something along the lines of the Lehmann Brothers collapse in 2008. There are steps taken by the government and private enterprises to soften the blow of such an event on the whole financial system and the country's economy. There are reports of a liquidity crunch and a collapse of the housing related investment instruments. Job losses have also been reported in many quarters and there are reports of house foreclosures across the country.
- *Scenario 2: Three Examples of Stock Market Trends* – A) There are rumors of a possible buyout of Twitter by Google. The stock price goes up by 20% in the first 2 hours after the stock market opening. B) There is news about Rackspace soliciting an offer of a buyout by a private equity firm and the stock rises 25% during after-hours trading. C) There are reports of China planning to dump a large amount of steel in the international market which can result in depression of commodity steel prices globally. The steel commodity futures tank around the world.
- *Scenario 3: Socio-economic Events and Trends* - The Indian government suddenly announces that all the 500 and 1000-rupee denomination bills will stop being legal tender from midnight. Hordes of people line up to dispose of their unaccounted and untaxed money stashed in the form of cash, to buy luxury items and gold late into the night. There is a huge political and economic turmoil in the country because of this. Serpentine queues of people are seen in every bank for them to exchange the old bills for the new ones. There

are reports of gold prices commanding a premium of 50% on the base price just to allow people to buy gold with the demonetized currency.

- *Scenario 4: Natural Disasters* - An earthquake rated 7.4 on the Richter scale hits a metropolitan city region in Japan. There are reports of the buildup of tsunami waves from the epicenter of the quake. The nuclear plant in the vicinity is shut down. Nissan suspended operations in the car factory in the vicinity. People are urged to stay away from coastal areas. The government issues advisories for people to stay calm; very few injuries are reported.

1.1.2 Why Analyzing Events and Trends is Important

Given the various scenarios in the previous section, we want to study how events affect the society and the people at large. Especially we would want to answer the following questions resulting out of the scenarios described previously.

i. Have the impacts of financial disasters varied over time, or have remediation steps changed radically compared to previous disasters, particularly from year 2000 onwards? Also, has the social and economic impact of these financial disasters increased over the same period for the general population? How does an impact of financial disaster vary across different countries? Is it possible to gauge peoples' plight across the country and analyze which states have been affected the most?

ii. Is it possible to mine social media in real-time to identify potential trading strategies? What would be a way to identify which Twitter users to follow for such information? How do we disambiguate rumor from actual news?

iii. Is it possible to gauge the impact of a sudden financial policy on a population? Can we gauge whether people's reactions to the move are positive or negative? Can the administration anticipate some potential challenges that people may face and work to resolve them? Is the perceived impact on the population truly statistically higher or just influenced by biased media reporting or pre-conceived fears?

iv. Is it possible to assess the geographical impact of a natural disaster? Can the emergency services identify critical areas to target their relief efforts? Have some areas been completely disconnected from the main infrastructure? Is it possible to gauge peoples' reactions to a natural disaster? Have there been any incidents of social violence and disorder that must be addressed with high priority?

1.1.3 Virginia Tech DLRL and GETAR

Virginia Tech has been researching information storage and retrieval and digital libraries. More than 1.5 billion tweets have been collected and studied. We propose to mine this huge amount of data and discover some of the knowledge contained within. Especially, we want to classify all the tweets relative to human defined real-world events, topics, trends, and categories to aid our analysis in connection with the Global Event and Trend Archive (GETAR) project.

In particular, the Digital Library Research Laboratory (DLRL) at Virginia Tech collects and maintains a large tweet collection related to important real-world events. These tweets have been stored in an HBase database [1]. Due to the different strategies for collecting, the data is organized into over 1300 different collections, such as "*Boston Blast*", "*Boston Explosion*", or a generic "*Explosions*" collection. Our goal is to develop an efficient and effective classification methodology, which reads the data from HBase, classifies the tweets into a predefined set of several hundred specific classes, and writes the result of the classification back into HBase.

There are hundreds of general categories in the collections. Some categories are broad in nature and can encompass multiple topics, real-world events, or trends. With the combination of these, we also have hierarchies. The goal in the future is to have this hierarchy clearly identified automatically via a combination of different techniques including classification and clustering.

1.2 Problem Statement

At a very high-level, we have the following as our core problem statement:

Given a tweet collection and a set of event classes in the real world, we are to build a classifier that can classify the tweets into the appropriate event class.

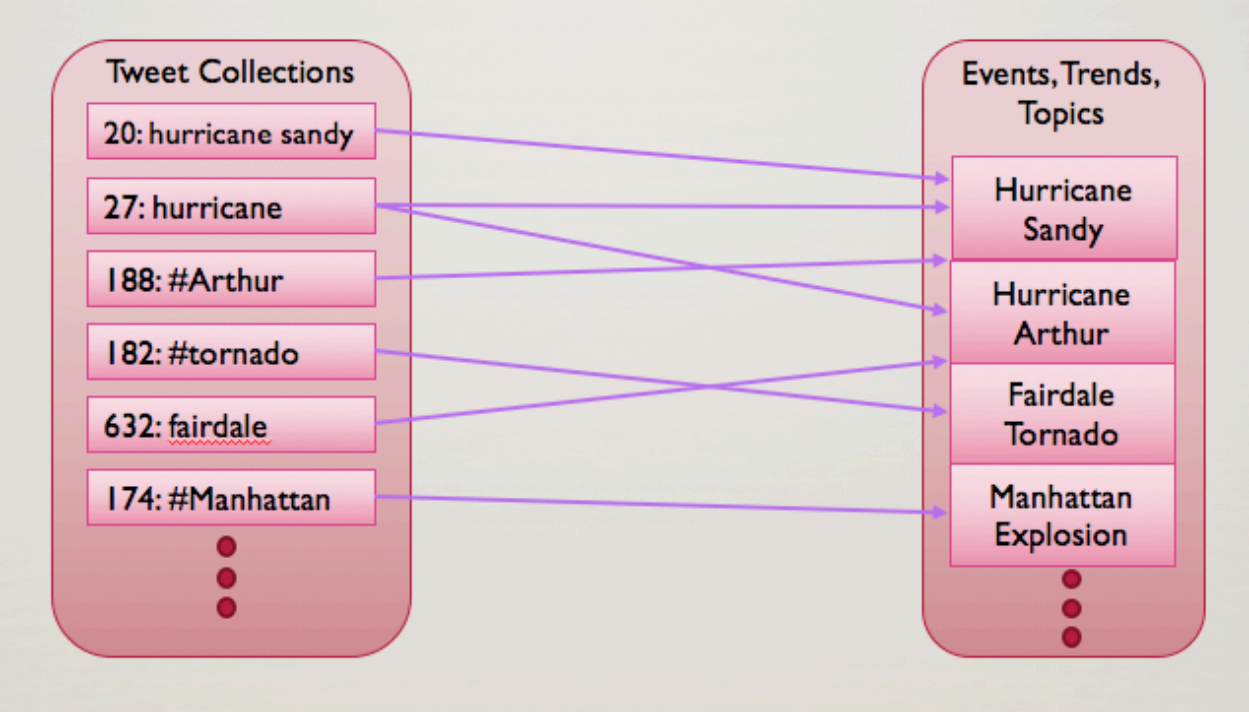

Figure 1: Problem statement

Figure 1 explains the problem pictorially. Essentially, there is a set of collections of tweets that have been retrieved based on keyword/tag search performed using the Twitter API. These are shown in the box in the left section of Figure 1. The human defined events or the real-world events as stated in the goal are shown in the box in the right section of Figure 1. The relationship between the collection of tweets and the events is many-to-many.

For instance, the tweet collection "weather2012" can have tweets related to three categories, for hurricanes "Sandy" or "Isaac" or the blizzard called "Gordon", as all of those three occurred in the same year and all are related to bad weather. On the other hand, tweets that fit under the category "Hurricane Isaac" can come from each of the three collections "hurricane," "hurricaneisaac," or "#hurricaneisaac."

Data will be collected from Twitter using keyword searches, and added to the DLRL set of collections. The DLRL collections will be constantly growing, so it will be a daunting task to classify all of the newly added event classes.

We want to develop classifiers for the collections that will label the data with high accuracy[1] given the limited amount of text that is contained in the tweets. We want to evaluate the state of the art feature representation techniques to model the text in the tweets.

We want to develop a system that can be given a small amount of training data pertaining to a category. The system can automatically develop a classifier based on that and then subsequently be able to classify the tweets in one or more collections. Having such a system to generate a classifier on demand will pave the way for us to classify all the DLRL collections with very little human intervention.

1.3 Motivation

As part of the larger goal of studying events and trends via the data in the tweet collections, there needs to be a robust and efficient classification system. The classifier that we have to develop should be able to withstand the non-conventional orthography and language usage of the Twitter domain. It is important to recognize the ability of this classifier to infer the class of the tweet by contextualizing it accurately, considering the character limit of 140 characters per tweet. Twitter data also includes a lot of noise and spam, such as advertisements and irrelevant hyperlinks, which need to be filtered.

Having the DLRL collections labelled relative to the appropriate classes as an initial step would help us get closer to our overall goal of studying events and trends. We would also need to classify a tweet in a collection as possibly having multiple labels. For example, a tweet belonging to the class “Hurricane Sandy” can also belong to the class “Hurricane”.

As new collections are continually added to the DLRL collections, it is imperative for us to classify the collections using very little human effort. The system should be able to make very effective use of a few samples for training and augment itself with more labelled data from an unlabeled auxiliary collection using sophisticated techniques. The system should also select an optimized classifier after training and classifying the collection automatically. An automated system to generate a classifier with very little human effort would aid in efficiently classifying collections.

1.4 Hypotheses

Training a classifier for categorizing tweets would require us to transform the tweet text into a vector representation for a machine learning algorithm to train on. It would also require us to choose the kind of classifier that is suitable for this kind of work. Also, we want to develop an approach to train a classifier in a semi-supervised automated manner.

The hypotheses of this work are:

- Pre-processing the text contained in the tweets can help eliminate noise related to the Twitter domain; using sophisticated NLP techniques like lemmatization, stop-word removal, and regex based string patterns removal can help improve classification accuracy.
- Using a rich feature representation like Word2Vec can result in higher accuracy in classification of tweets than methods based on Association Rules.

[1] The term accuracy means F1-score unless mentioned explicitly otherwise.

- Utilizing a distributed computing infrastructure using Apache Spark can result in increase in run-time performance of the Logistic Regression based classifier.
- Higher accuracy can be achieved by augmenting a base classifier trained on a small amount of training data with additional examples to learn from.
- Using a Word2Vec based feature representation can also help identify what auxiliary data is similar and dissimilar to the positively labelled training data. The cosine similarity [2] can accurately indicate which tweets can be used as positive or negative examples.
- Word2Vec based feature vectors for tweets trained from the auxiliary collection can help achieve better classification accuracy than the Google word vectors since these vectors are trained on words from the same domain that we are classifying.
- Using sophisticated deep learning based classification methods like CNN can help attain consistently higher accuracy compared to simple classifiers like Logistic Regression, Naïve Bayes, Decision Trees, etc. The CNN classifier has the ability to identify the most distinguishing keywords by utilizing the max-pooling layer. This gives it an edge over simple machine learning techniques.
- The CNN based classifier has the ability to account for the word sequences in the text using a bi-gram or n-gram representation, whereas the bag-of-words approach using Logistic Regression cannot account for the sequences of the words. A CNN classifier constructed using a bi-gram or tri-gram feature representation would perform even better than a unigram representation.

1.5 Research Questions

The research questions that this thesis will attempt to answer are:

- Can a richer feature representation like Word2Vec along with Logistic Regression attain better classification accuracy than simpler, and more easily interpretable, classification techniques like Association Rules?
- Can an approach, based on augmenting training data, to a classifier, improve the accuracy of a classifier after multiple iterations of augmentation?
- How do the classification results using Word2Vec vectors generated from the Google news source compare to the results produced by the vectors generated from an auxiliary source?
- Can similarity measures like cosine similarity be used on the Word2Vec vectors to label the unlabeled auxiliary samples for training?
- Can deep learning techniques like Convolution Neural Networks (CNN) be used to classify the tweets? Does the augmented training help improve its classification accuracy?
- Does the composition of features based on a unigram, bi-gram, and tri-gram word representation have any effect on the final classification results for the CNN-based classifier?
- For the semi-supervised automated classifier framework, how well does the generated classifier perform based on a completely unseen collection?

1.6 Research Contributions

This work makes the following contributions to the field of digital libraries in general, and the techniques for text classification in particular.

- It outlines an approach to classify short texts using various classification techniques and lays out a few rules for classifier and feature selection depending on the kind of dataset and the number of classes.
- It provides an approach on how to run the classification of tweets using Logistic Regression and Word2Vec on the Apache Spark platform using the full features of the platform including but not limited to distributed computing and fast execution.
- It describes an approach on how to extract additional training from an auxiliary data based on a very small initial training set. It also evaluates the efficacy of using Google News and the auxiliary source data based word vectors, and compares the results generated by the various classifiers using these vector representations.
- It introduces an implementation of deep learning techniques for classifying the tweets and compares the results using unigram, bi-gram, and tri-gram word context representation.
- It provides an automated approach for developing a classifier for large collections with very limited supervision. This is very useful for the Digital Library community as developing classifiers for every purpose is a time intensive process.

1.7 Outline of the Thesis

- Chapter 1, above, outlines the problem statement, motivation, hypotheses, research questions, and research contributions.
- Chapter 2 presents a survey of the relevant literature in the area of feature selection and text classification.
- Chapter 3 elaborates on the design of the system.
- Chapter 4 elaborates on the data sets used in the case studies.
- Chapter 5 elaborates on the preliminary case studies for our work.
- Chapter 6 elaborates on the large collection optimizer based case studies for our work.
- Chapters 7 and 8 present the conclusion and the possible future work recommendations.
- Chapter 9 lists the references used for this work.
- Appendix A elaborates on the implementation details for the work.
- Appendix B elaborates on the implementation artifacts for the work so that it can be reproduced.

2 Literature Review

The textbook [3] for the fall 2016 CS5604 class on Information Storage and Retrieval introduces the classification problem we are trying to solve: Given a document and a set of classes, what is the subset of those classes that this document belongs to? It also discusses the different feature selection methodologies for text classification. These features are then used in the training of the classification methods described. The classification methods discussed in the book are Support Vector Machine (SVM), Naive Bayes, and Vector Space Classification. The textbook helps us get a head start into the problem from a breadth perspective and gives us a platform to start studying more recent techniques on feature selection, vector space representation of words, and classification, from the latest research literature in the area.

2.1 Feature Selection

The process of text classification involves extracting features out of the text data and finding an appropriate vector space representation for it. This is the feature selection stage of the process. Once the feature representation of a piece of text is obtained, it can be fed into any classification algorithm like logistic regression or SVM, and be used for training. This is the classification stage of the process.

As part of our literature survey, we came across the following feature selection methods. In all but the last in this list, the sequence of the words is ignored; a bag of words representation is employed.

- **Chi-squared statistic -** This technique [4] draws its discriminative ability from the lack of independence between terms and the documents. The key feature of the technique is that it can reduce the dimensionality of the feature space to ensure the high performance of the classifier.
- **Mutual Information -** This technique [3] is used to measure the global goodness of a term in feature selection. This technique measures the mutual information between a term and a class.
- **Tf-idf -** Term frequency–inverse document frequency weight measures how important a word is, relative to a specific document and document collection [3]. This value increases as the term appears more frequently but is scaled by how often the term appears in the entire corpus. This makes terms that are common throughout the text such as "the" to not be weighted heavily even though it would appear many times in each specific document.
- **Information Gain -** This is another technique to measure the goodness criterion. It measures the number of bits required for category prediction by knowing the presence or the absence of a term in the document [3].
- **Word Class Popularity -** This technique explores the relative distribution of a feature among the different classes. The goal of this technique is to identify the features that discriminate the classes the most. A good discriminant term will have a skewed distribution across classes. This technique uses the Gini-coefficient of inequality to analyze the distribution of a feature across the classes [5].
- **Word2Vec -** This technique [6, 7] generates the word vectors out of the training corpus based on the context in which they occur. The context of the word is defined by the word and its neighbors that surround it. For example, a word might have words preceding it and

succeeding it. The word along with its neighbors form the context, though the length of the window in which the neighbors are defined is a parameter that can be tweaked. The two forms in which the context of a word is identified are the CBOW (continuous bag of words) and the skip-gram method. The skip-gram based technique predicts the surrounding words given the current word. The CBOW technique predicts the current word given the surrounding words. A neural network is trained based on these techniques and the trained hidden layer weights are used to generate the word vectors. Once the word vectors are generated, the words that are closer in context to each other are close to each other in vector space based on their cosine distances. This attribute of the word vectors makes them very useful for text classification. The word vectors based on the skip-gram technique give the state of the art results [8] for text classification. From a feature selection perspective, since each word generates a vector, we calculate the average of the values in the vector and use that value as the feature value for a given word. The bigger the corpus is, the more effective the word vectors become from the perspective of text classification.

2.2 Classification Techniques

The work on this thesis is divided into preliminary and large collection classification case studies. The preliminary case studies in Chapter 5 focus on finding the best feature selection and transformation technique for representing text. We plan to use simple classification techniques to train the classifiers for the preliminary case studies. As part of our literature survey, we came across the following classification techniques.

- **Logistic Regression** - Logistic regression [3] measures the relationship between the categorical dependent variable and one or more independent variables by estimating probabilities using a logistic function, which is the cumulative logistic distribution.
- **Support Vector Machine (SVM)** – This is a classifier that will perform linear or nonlinear classification through a kernel trick [3]. This classifier works by linearly separating the classes so that each class falls onto one side of the separator. This classifier maximizes the distance between the separator and the points on either side to identify the separator it will use.
- **Multi-layer Perceptron (MLP) -** It is a feed-forward artificial neural network [9] model that maps sets of input data onto a set of appropriate outputs. An MLP consists of multiple layers of nodes in a directed graph, with each layer fully connected to the next one. It uses the supervised learning technique named back-propagation to learn the network weights.
- **Naïve Bayes** – This classification method is based on applying Bayes Rule under the assumption of independence. This means it treats each feature as independent of the others with respect to the class the document will fall into. Despite the assumptions, Naïve Bayes has been shown to perform well in real world situations [3].
- **Association Rules -** AR uses the training data to create Association Rules for each class by identifying rules that will lead to a specific class identification [10] [11] . The rules can then be used by a rule engine to quickly predict the class of new documents. The challenge with the particular AR classifier we consider is to see how well it performs on long texts since its efficacy has typically been evaluated on short texts.

2.3 Deep Learning Techniques

To further progress with our task of classifying large collections, we plan to research into novel techniques on minimizing the human effort required for labelling the collection. The core part of this task would be training a classifier with very little training data. We would want to evaluate the efficacy of this methodology coupled with the choice of feature selection and classifier as part of the preliminary experiments. We also want to evaluate classifiers that have shown state of the art results in text classification. Though these kinds of classifiers are not simple to implement, we want to compare the classification efficacy of basic classification techniques with some deep learning based classifiers.

- **Convolutional Neural Networks –** CNN based deep learning techniques [12] have shown results comparable to the state of the art in text classification. A CNN based classifier has the ability to incorporate the word sequencing in its learning technique and is quite effective in text classification.
- **Recurrent Neural Networks –** RNN based [13] techniques have shown excellent results for text classification. Combining RNN and CNN allows building sophisticated layered neural networks that help in capturing the contextual information and learning the most distinguishing keywords more effectively than traditional classifiers.

3 Design

3.1 Approach

To address the problem, hypotheses, and research questions in sections 1.2, 1.4, and 1.5, respectively, we present our approach to the development and implementation of our classification system – A Meta-classifier for Large Collections.

The following steps are planned to accomplish this work:

- A literature review of the field (see Chapter 2) was performed to identify the various feature representation and transformation techniques for converting the text into vector space features. The representations were examined to measure their effectiveness and efficacy for text classification of short texts.
- Requirements gathering and analysis was performed to understand the problem at a high-level, considering how we could achieve the end goals without getting too distracted in perfecting one small part of the whole system.
- Since we had to classify large collections, we wanted to build a system that could scale well to meet our requirements. We researched the various ways to optimize our implementation to run in a distributed fashion using Apache Hadoop and Spark [1].
- We performed evaluation of the different classification techniques including but not limited to Association Rules and Logistic Regression with Word2Vec and IDF features. We performed extensive evaluation of the experimental results from this step to select the techniques for our next line of work to generate the meta-classifier.
- Once we selected a feature representation and the classification techniques, we performed the step of augmenting the training process using examples selected from an auxiliary data source and training a classifier based on that. We also implemented a CNN based classifier to compare against as it has been shown to have text classification results comparable to the state-of-the-art. Further experiments and evaluation allowed successful completion of our investigation.

3.2 High Level Architecture

The vision for GETAR is to have a complete system with a UI, supported by a sophisticated backend that will enable an end user to search for global events and trends so that they can be mined for study and analysis. The Fall 2016 class for CS5604: Information Storage and Retrieval [14], had designed and developed a system to serve the GETAR vision. Figure 2 [15] shows the high-level architecture for all the components as developed by the class.

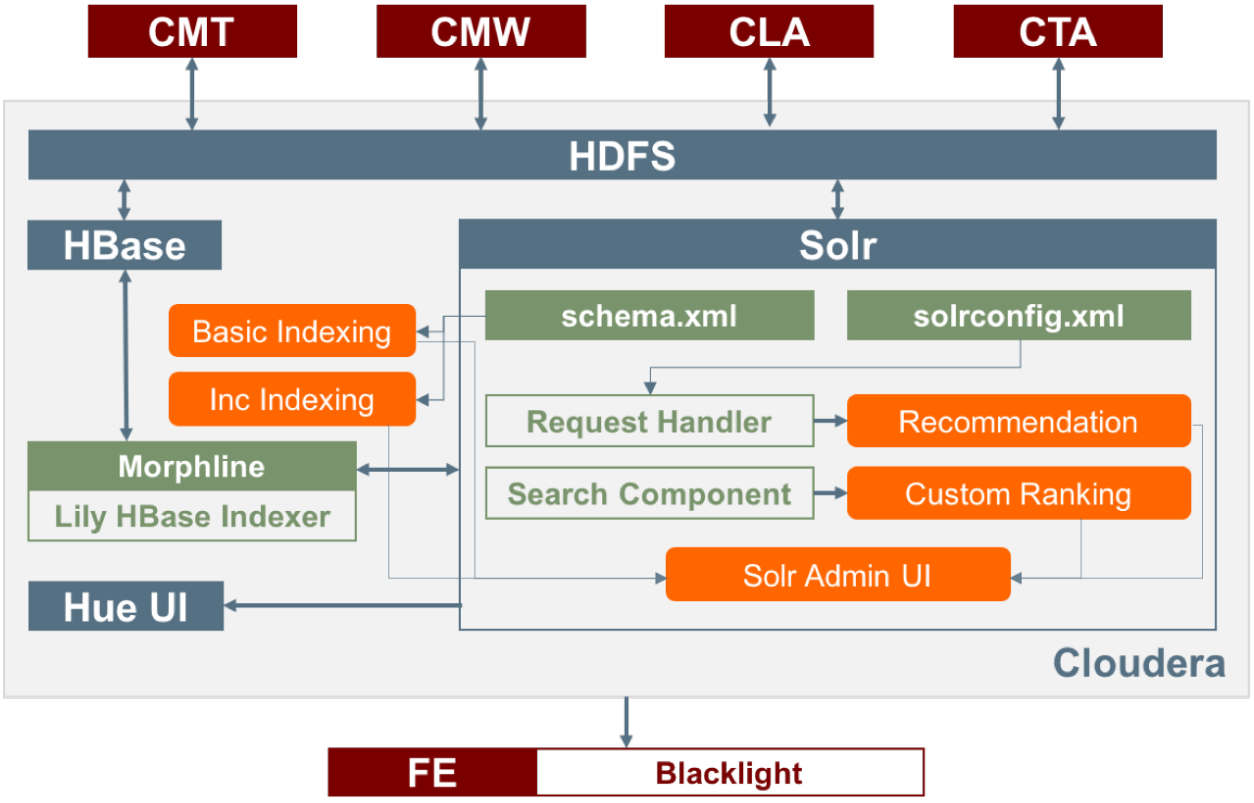

Figure 2: High level architecture [15]

The various team components highlighted above had the following contribution and role in the overall system.

- CMT (Collection Management – Tweets) [16]
- CMW (Collection Management – Webpages) [17]
- CLA (Classification) [18]
- CTA (Clustering and Topic Analysis) [19]
- SOLR (Data Indexing) [15]
- FE (Front-end) [20]

3.3 The Classification Pipeline

The classification component is shown along with other components that are part of the system that the class built over the whole semester. As described in Section 1.6, our contribution in building the above system was the classification pipeline.

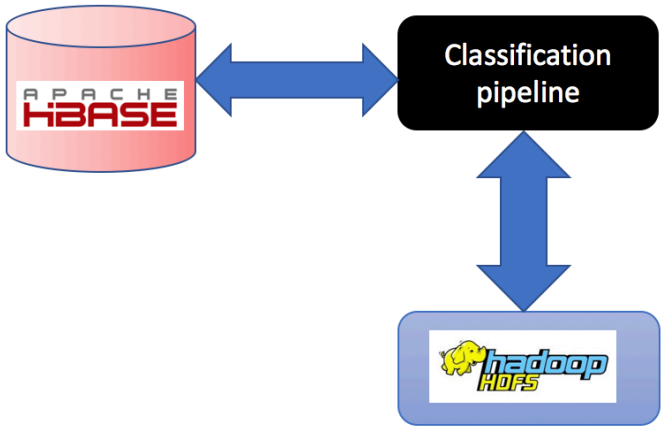

Figure 3: High level view

In Figure 3, we show a high level black box view of the classification pipeline and the data sources that it interacts with. The classification pipeline accesses the HBase database to read the raw tweets from the table ***ideal-cs5604f16*** and writes back the classification label to the **real-world-events** column in the same table. This column is part of the column family **clean-tweet** that was created for storing classification related columns.

3.3.1 Data Pre-Processing

To aid faster development of the classification system, we clean the tweets for our training data. This flow is shown in Figure 4. We remove all the non-English words, short URLs, and emojis as part of this process. We also interpret the hashtag and mentions into terms by breaking them into multiple words based on the casing. An example would be when the string "#HurricaneSandy" is broken into the tokens "hurricane" and "sandy". We also remove stop words and lemmatize all the words that remain. The pre-processing of the tweets was based on the Standford CoreNLP Toolkit [21].

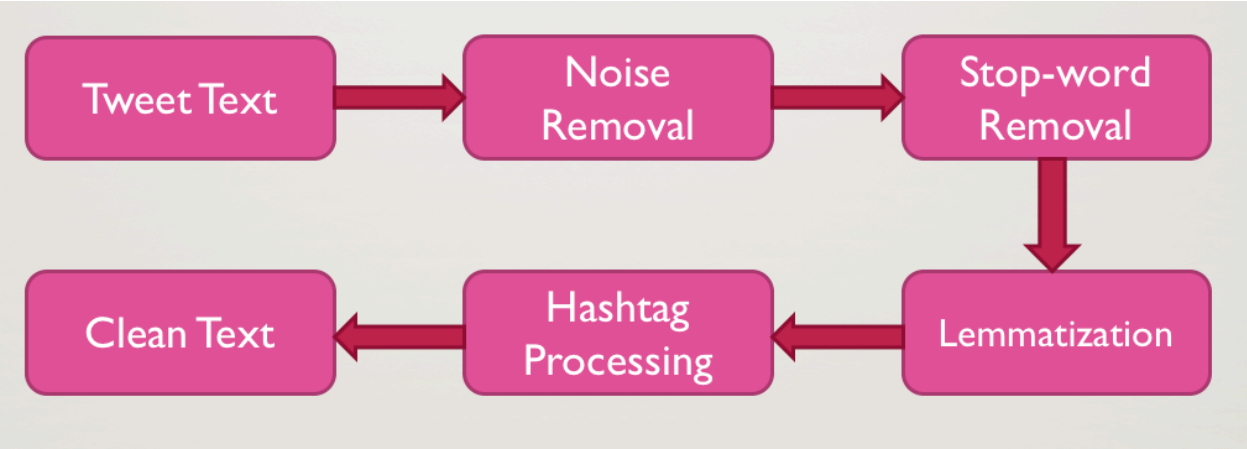

Figure 4: Data pre-processing phase

Table 1 shows the effect of pre-processing on a raw tweet.

Table 1: Effect of pre-processing a tweet

Raw Tweet	RT: @AssociationsNow A Year After Texas Explosion Federal Report Outlines Progress on Fertilize... http://t.co/8fDbMu9asU #meetingprofs D&%$
Processed Tweet	year texas explode federal report outline progress fertilize meeting prof

3.3.2 Association Rules

Association rule [22] mining finds interesting associations and correlation relationships among large sets of data items. Association rules show attribute value conditions that occur frequently together in a given data set.

A typical example of association rule mining is Market Basket Analysis [22]. Data is collected using bar-code scanners in supermarkets. Such market basket databases consist of a large number of transaction records. Each record lists all items bought by a customer on a single purchase transaction. Managers would be interested to know if certain groups of items are consistently purchased together. They could use this data for adjusting store layouts (placing items optimally with respect to each other), for cross-selling, for promotions, for catalog design, and to identify customer segments based on buying patterns.

Association rules provide information of this type in the form of if-then statements. These rules are computed from the data and, unlike the if-then rules of logic, association rules are probabilistic in nature. In addition to the antecedent (if) and the consequent (then), an association rule has two numbers that express the degree of uncertainty about the rule. In association analysis, the antecedent and consequent are sets of items (called itemsets) that are disjoint (do not have any items in common).

The first number is called the support for the rule. The support is simply the number of transactions that include all items in the antecedent and consequent parts of the rule. The support is sometimes expressed as a percentage of the total number of records in the database.

The other number is known as the confidence of the rule. Confidence is the ratio of the number of transactions that include all items in the consequent, as well as the antecedent (the support), to the number of transactions that include all items in the antecedent.

The Association Rules classifier [10] is a supervised learning method, whose generated model is a set of association rules with 100% [22] confidence to identify each class. The classifier processes the tokens (words) in a tweet and learns rules based for a given class (e.g., hurricane sandy). In the prediction phase, the classifier generates sets of tokens from the string to be predicted, matches them with the antecedents of the rules in the model generated in the training phase, and using a voting scheme, decides the prediction.

3.3.3 Feature Selection and Transformation

As part of the pre-processing phase for our project, we clean the tweets, lemmatize the words contained, and remove the stop words. In spite of this, the number of words in a bag of words representation is still large. Feature selection methods assist in further reducing the dimensionality

of the feature set by removing the irrelevant words. The goal of reducing the curse of dimensionality[23] is to improve classification accuracy and reduce over fitting.

Methods for feature selection for text document classification use an evaluation function that is applied to a single word. The goal is to identify a subset of words that assist in discriminating between the classes the most. Techniques like Document frequency (DF), Term frequency (TF), Mutual information (MI), Information gain (IG), and Chi-square statistic (CHI) use feature-scoring methods to rank the features by their independently determined scores, and then select the top scoring features.

Another technique to reduce the size of the feature space is referred to as feature transformation. This approach does not eliminate features because of their low scores, but compacts the feature dimension based on feature concurrencies.

Words are central to text classification. The challenges with traditional feature selection techniques are that they are based on a bag-of-words representation. This representation fails to capture the neighboring context of a word in a sentence. The absence of this context results in the loss of the semantic relationship of the word with its neighboring words.

Word embeddings [6] offer distributional features about words. They capture the context of the word in its neighborhood. This results in an extension to the bag-of-words representation along with context and word sense information. Word embeddings are low-dimensional, dense vector representation of words. The compact representation along with capturing of the context make this a strong choice for the feature representation for words in text classification scenarios.

Work in [7] defines specific objective functions for efficient training of word embeddings, by simplifying the original training objective of a neural objective model. The two variants of the objective functions are as follows:

a. **Continuous bag-of-words (CBOW)** – Given a word, predict the context.
b. **Skip-gram** – Given a context, predict the word.

Figure 5 shows a Word2Vec CBOW neural language model. It is a one layer, 300-neuron neural network with [1 x V] Boolean vector as input and a [1 x V] float vector as output, where V is the vocabulary size, in this case 10000. The input to the neural network is a one-hot representation of a word in the form of a [1 x V] Boolean vector. The word “ant” is the given word in the example, and the neural network objective is to maximize the probability of the words that could be its neighbors. The words are fed into the neural network from a training corpus and it generates the relative probabilities for all the words in the corpus.

The goal of the Word2Vec implementation is to just store the word weights that are in the hidden layer representation of neural network. For the example given in Figure 5 [24], the output of importance is the [300 x 10000] matrix that gets generated for the word corpus, which in this case consists of 10000 words. This is shown in Figure 6 [24]. For our work, we use the training tweets to generate the word vectors.

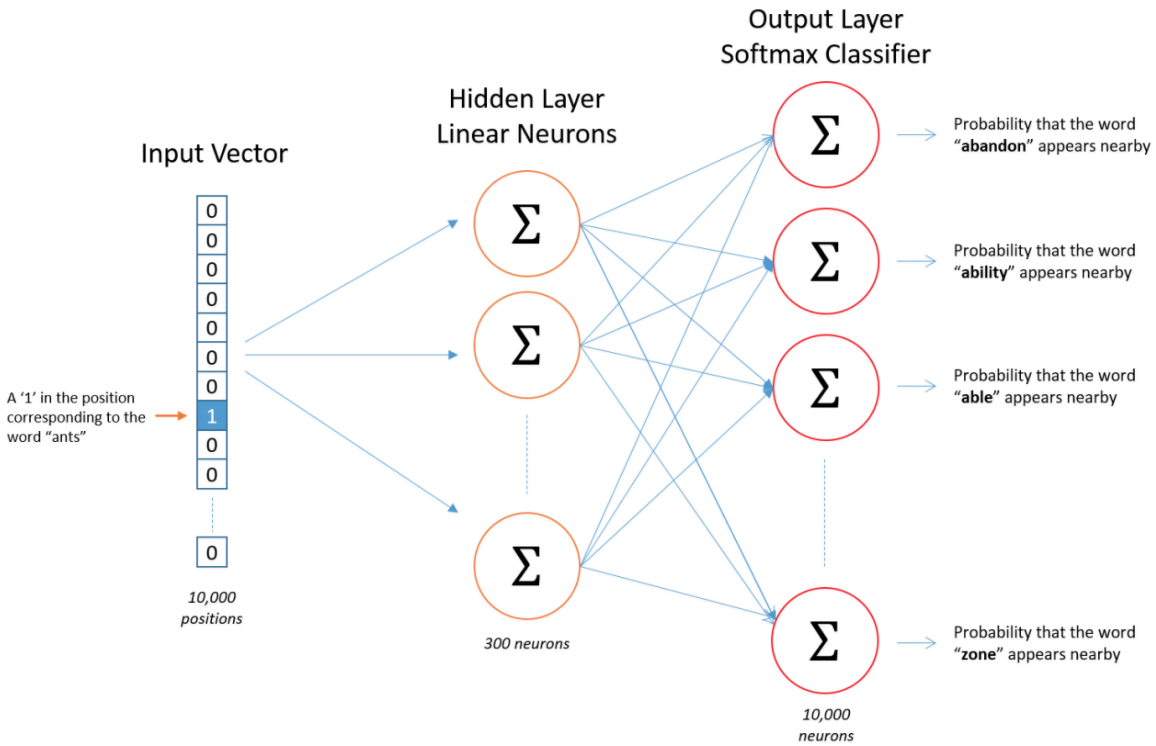

Figure 5: A neural language model for Word2Vec [24]

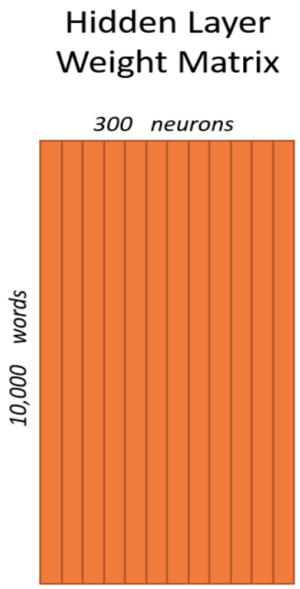

Figure 6: The hidden layer weight matrix [24]

3.3.4 Training

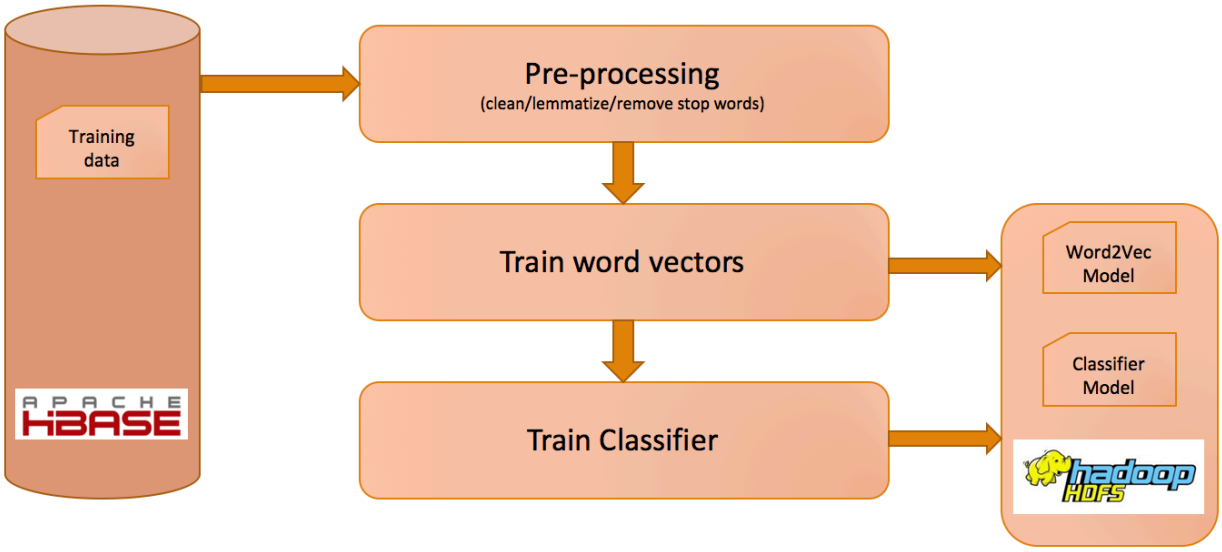

Figure 7: The training phase

In the training step, we read[2] training data that we curated[3], from the HBase data source, to train a classifier. As shown in Figure 7, we read the raw tweets from HBase and then perform pre-processing. Next, for the training samples, we transform the text in each tweet into an appropriate feature representation. We train the classifier using these features. Once we have trained our classifier, we save the model file in HDFS.

The training pipeline is executed in an offline manner. This means that we generate the features and classifier models beforehand to ensure that this step is not repeated during run-time. All the software artifacts (models) generated out of this phase are persisted in HDFS and used later in the prediction phase which runs online.

[2] The reading of the data from HBase is discussed in Appendix A3. HBase Access
[3] The generation of training data is discussed in Appendix A2. Training Data Curation

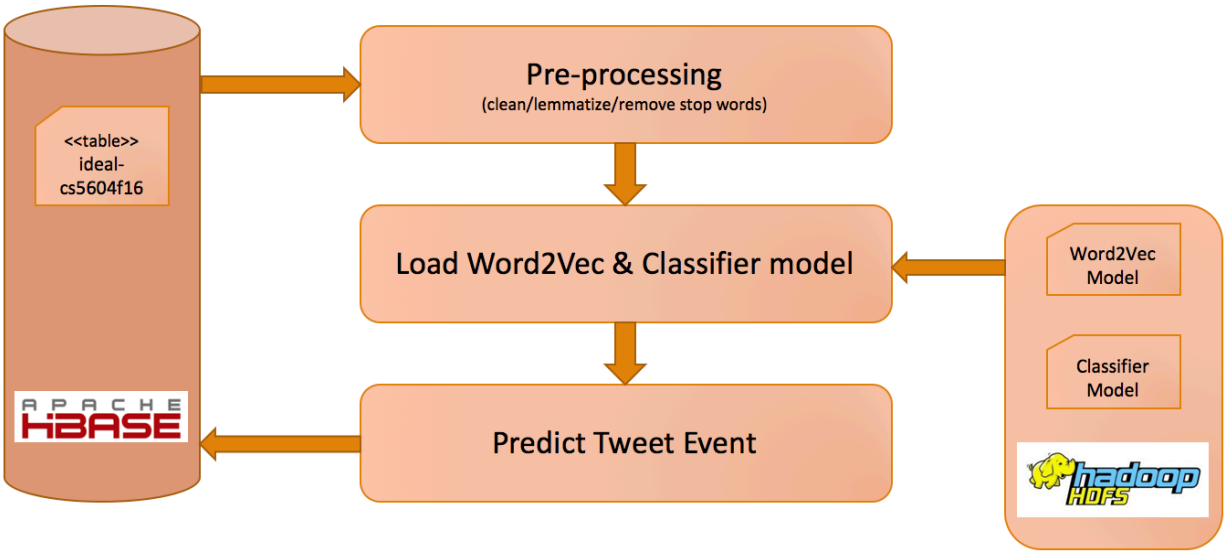

Figure 8: The prediction phase

3.3.5 Prediction

As part of the prediction phase shown in Figure 8, a block of tweets is read from the table *ideal-cs5604f16* in HBase and labelled by the classifier. This labelled data is written back to the same table in HBase. The appropriate feature and classifier models are loaded from HDFS in the beginning of the phase and are resident in memory through the end of the prediction phase.

3.4 Classification Methodology

Based on our research, we use the high-level approach shown in Figure 3, to identify the best choice of the feature selection and classification techniques. Our experiments are organized in the following way.

- **Training data creation** - We generate training data by selecting 3 broad event classes and then selecting tweets from the 3 real-world events each from HBase belonging to the broad collection. The 3 broad categories are selected to vary the type of tweets that we have in our experiments. We choose the 3 sub-categories to validate that the classifier can correctly classify the tweets that fall within the same broad category, i.e., "Hurricane" is an example of a broad category and "Hurricane Sandy" and "Hurricane Issac" are sub-categories of it. The tweets are manually annotated and randomly divided using a ratio of 70:30 for the train/test mix for the experiments. Section 4.1.1 describes the dataset in detail.
- **Classification via Association Rules** - We use the Association Rules based classifier and generate a baseline result. Section 3.3.2 describes the technique in detail.
- **Feature selection via Word2Vec method** - We use this technique to generate the word vectors [6, 7] on the same training data and generate features for the tweet texts using the word vector model that we have generated out of the training tweets. Section 3.3.3 describes the technique in detail.
- **Classification via Logistic Regression** – We use the multi-class logistic regression classifier to train a model based on the Word2Vec feature selection technique as described in Section 3.3.3. We also perform a 10-fold cross-validation as described in Section 5.3.1, to select the best model and save it into HDFS. We perform this step so that we can load

the best model out of the file system instead of training the classifier again. This helps in reducing run time.

- **Evaluation of results** – Since we are implementing a multi-class classifier, we compute the micro-F1 scores [25] across all classes to evaluate the overall classification efficacy of the classifiers. We compare the results of this Association Rules based results with our proposed feature selection/transformation and classifier techniques to choose the best classification method for our following case study. Section 5.3.2 describes the results in detail.
- **Writing to database** - The classification results for tweets are recorded in the database by writing to the "real-world-events" column family for each tweet. Each tweet is associated with one or more of the categories to which we have determined it belongs. Appendix A3 describes this in more detail.

3.4.1 Runtime Optimizations

Spark is a general-purpose cluster computing system that empowers other higher-level components to leverage its core engine. While it allows building other higher-level applications on top of it, it has a few components that are tightly integrated with its core engine to take advantage of the future enhancements at the core. Spark is built on top of the Hadoop MapReduce framework to provide an extension to it based on its basic primitive, the Resilient Distributed Dataset (RDD).

The main idea behind RDDs is that they are immutable collections of statically typed objects spread across a Hadoop cluster. The partitioning of the RDDs and storage is designed to be user controlled. The Spark SDKs have extended the programming language to support RDD operations (map, filter, etc.) that have the capability to be executed lazily depending on user implementation. The RDDs are designed to be automatically rebuilt in case of failure, and a lineage of a failed RDD operations can be detected automatically and be assigned for computation to another node implicitly.

The power of Spark comes from the ability to partition data across the cluster and perform computation on a piece of data in a parallel fashion. The Spark SDK exposes operations on the RDD primitive to enable the partitioning to be user controlled. In addition to the partitioning, it also allows an ability to checkpoint an RDD using the cache operation so that the data is persisted in memory and repeated access to the data does not incur an I/O penalty.

As part of our work, we will use the RDD caching and distributed computing capabilities of Spark. This has been described in detail in Appendix A8.

3.4.2 Augmented Training - A Semi-Supervised Approach

During our work that is discussed in the previous chapter, we observed that the activity of training data curation is very tedious. It is human labor intensive, and is susceptible to errors. Since we have new events and trends developing at any time, it is very challenging for a human actor to spend effort curating training data and developing classifiers for every new category. Nevertheless, we cannot eliminate the need for a human actor to curate some training data. Accordingly, we are motivated to find a way to minimize this effort. The challenges or questions we want to address with our design are as follows.

- Is it possible to train a classifier with less than 20 training samples with a somewhat balanced mix of positive and negative training examples?
- When tweets related to a new event start coming in, the number of positive examples to train on might be very limited. Can we still build a classifier in such a scenario?
- Will the classifier be able to classify tweets that contain words that were not present in the training dataset but may be present in the tweets coming in the future that are supposed to be labelled as positive?
- Can we train a classifier in a semi-supervised way with the minimal initial training data and augment it with additional training samples so that it can learn iteratively?
- How can we determine, at runtime, which samples in the auxiliary data are to be labeled as positive or negative? Can the rich feature representation of Word2Vec help us label the auxiliary dataset? Can we do some statistical analysis on the auxiliary data to achieve our goals?
- How will we measure the learning effectiveness of our classifier while it is learning iteratively on auxiliary training examples? What methodology can we use to evaluate the effectiveness of the classifier being developed?

We use an iterative process based on [26] where we augment the training process by providing additional training examples from an auxiliary data source. We perform validation using a held-out validation set. We start training a classifier using the initial training data. We perform 20 iterations, providing 20 training examples randomly sampled from the auxiliary source at each iteration, and perform classification on the validation set at the end of each iteration. If the F1-score on the validation set is more than 0.98, we exit out of the loop. If the training examples for a given iteration improve the accuracy over the best observed accuracy, we add these examples to the training set. Otherwise they are discarded. We use a different similarity measure to predict the labels for the retrieved augmented data. This is being done so that we can use the rich Word2Vec feature representation for calculating the similarities. Section 3.4.4 describes this in detail.

3.4.3 Methodology

High accuracy can be achieved by augmenting the system with additional training data [26]. The additional training data is sourced from the collection that contains the tweets that have to be classified. The system keeps on retrieving more auxiliary training data until it reaches a certain level of accuracy on the validation set or has run through a certain number of iterations. We use the term base accuracy to define the best achieved accuracy by the classifier across all iterations. We use the term validation accuracy to define the classification F1-score that is obtained by classifying the validation set. The training and validation sets are described in Section 4.2.1. The following steps describe the whole process:

- Label the unlabeled examples in the auxiliary source using sophisticated techniques. The labelling technique is described in detail in Section 3.4.4.
- Add training examples in batches of 20 tweets from the auxiliary source.
- Train and validate the classifier against the validation set.
- If the batch of training examples improves the F1-score over the best observed score, add to the training data, else discard the current batch .
- Perform the loop until the F1-score reaches 0.98 or a set number of iterations have been performed.

The algorithmic approach to augmenting training is shown in Figure 9.

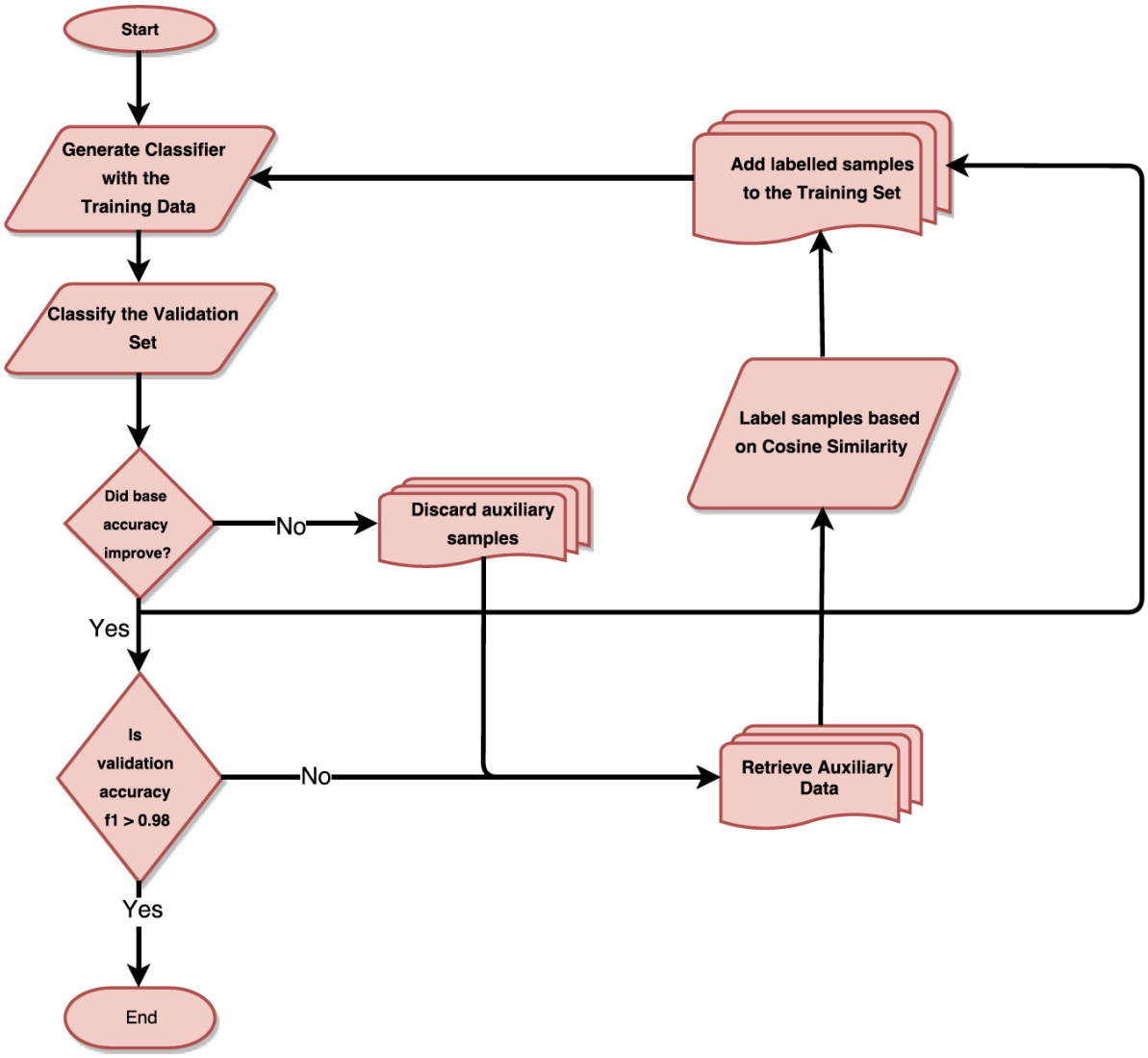

Figure 9: Flow for augmenting training data using a semi-supervised approach

3.4.4 Similarity Measure for Labeling Auxiliary Data

We are using the Word2Vec [27]vectors as the representation for the words as described in Section 3.3.3. The representation is very rich and conducive to vector operations [7]. Using cosine similarity [2] [3] as a measure of similarity between tweets can prove to be effective [6] [28] to differentiate the positive and negative labels in the auxiliary data. We use the cosine similarity as a heuristic to enable us to label unlabeled examples.

To validate whether the cosine similarity would be an ideal measure of similarity, we performed a case study to measure the similarities between the positive training samples and tweets from the

auxiliary dataset. The results of this case study are discussed in Section 6.1.2. Based on our initial experiments with the cosine similarity measure, we observe the following:

- The samples that are highly similar to the positively labeled training examples have a high mean cosine similarity score with the positively labeled training samples. We add these samples as positive examples to augment the training set.
- The negative samples that are highly dissimilar to the positively labeled training examples have a low mean cosine similarity score with the positively labeled training samples. We add these samples as negative examples to augment the training set.
- There are some potential positive samples that have mean cosine similarity scores close to the upper end of the cosine similarity scores observed for the potential negative examples. The same is observed for some potential negative samples having cosine similarity scores close to the lower end of the cosine similarity scores observed for the potential positive examples. Figure 10 shows the scenario where the cosine similarity values of the positive and negative training examples are separated by a good margin. Figure 11 shows the scenario where the cosine similarity values of the positive and negative examples have an overlap.

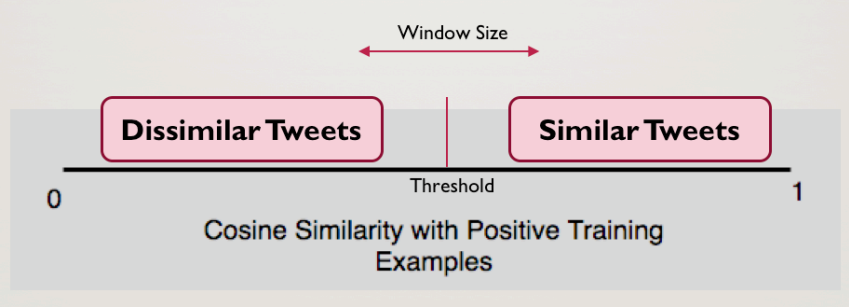

Figure 10: Training examples that have the cosine similarity values spread apart

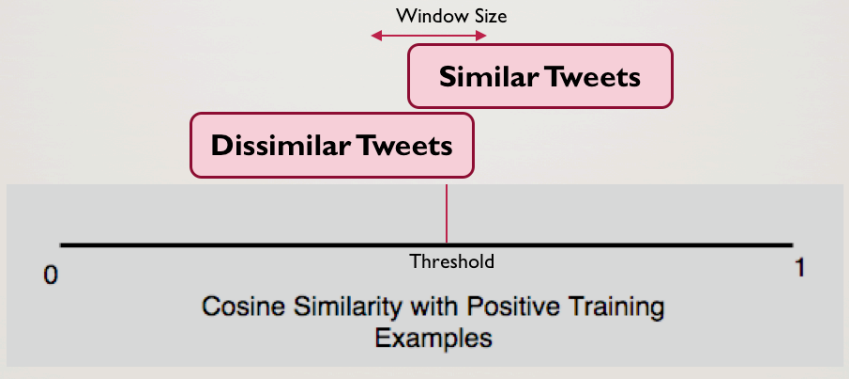

Figure 11: Training examples that have an overlap in the cosine similarity values

In order to deal with the situation encountered in the case of the observation in Figure 11, we look at the mean cosine similarity scores for the different datasets and conclude that we need a way to separate the positive examples from the negative ones. These samples have a high chance of inducing uncertainties in the classifier training phase. The uncertainties are evident during the training process. The negative examples start coming in as positively trained examples and the classifier gets trained with ambiguous training examples. Figure 11 shows the position of the threshold and the window size values. We need to have this window somewhere between the mean

cosine similarities of the potential positive and negative auxiliary examples. Any examples that fall within this window are discarded. Section 6.1.2 lists the cosine similarity scores and window sizes for 2 data sets. We will use these techniques for the respective datasets for our case studies, going forward.

Another aspect of the cosine similarity threshold and window size parameter is to let the user look at the tweets in the training set with their respective cosine similarity values and define it for a real-world event. Capturing these details from the users of the system is useful even when the user is not familiar with machine learning. We believe that system users are knowledgeable about a real-world event and will be able to correctly choose the threshold and the window size values based on their observation. The users should be able to look at the tweets with their cosine similarity scores and be able to decide on a good threshold and window size to separate the positive examples from the negative ones. Section 6.1.2 shows an example of positive and negative tweets along with their cosine similarity values that can be presented to users for them to select the values of the parameters.

3.5 Using Deep Learning Techniques for Classification

As part of our design that is discussed in Section 3.4, the feature representation and classification methods that we are using for training the classifier do not consider the word ordering in the tweet text. This effectively translates into a bag-of-words approach since the presence of words in a tweet text accounts for the label.

The work described in [12] uses Convolutional Neural Networks for text classification, which has the ability to incorporate an n-gram based representation for a given tweet text. We would want to evaluate the effectiveness of a CNN based deep learning technique since it has been shown [12] to attain or improve upon the state of the art results in text classification.
A CNN is a layered neural network. It is composed of the following components.

- **Input** - This is the input to the CNN. For text classification, this is the vector representation of the sentence. For our case studies, we have used a Word2Vec representation of a sentence as the input to the CNN.
- **Filters** - A filter is a variable dimension matrix which is used to perform convolution on the input. For text classification purposes, a filter is generally of the same width as the sentence vector, but its height can be varied to capture an n-gram representation of a sentence. This is very useful in sentence classification since the context of a word along with its neighbors can be represented as a feature in the intermediate layer of the CNN.
- **Convolution and Activation** - During the convolution step, filters perform convolutions on the sentence matrix and generate variable length feature maps. A single convolution operation is a sum of the elements of a dot product of the filter and the sentence sub-matrix followed by an activation function [29] applied on the scalar value of the dot product summation. For our implementation, we used the ReLU [29] activation function. To perform the convolution operations on the whole sentence matrix, the filter is moved through the input sentence matrix in step-wise increments based on a given step size value. For text classification, a step-size value of 1 is chosen generally. After performing convolution on the whole sentence matrix, feature maps are generated for each filter.

- **Max-pooling and Concatenation** - The max pooling layer selects the maximum value from each feature map. All the maximum values of all the feature maps are concatenated to form a single feature vector.
- **Fully Connected Layer with Softmax Output** - In this layer, all the individual feature values in the univariate vector that are generated as a part of the previous step, are fully connected to the output layer to depict the two possible output states. The final result is a weighted result of all the values of the feature vector for a weight vector for each output value, respectively. This is the class label that is the output of the CNN.

Figure 12 outlines the architecture of the CNN that we used for our implementation based on [12] [30].

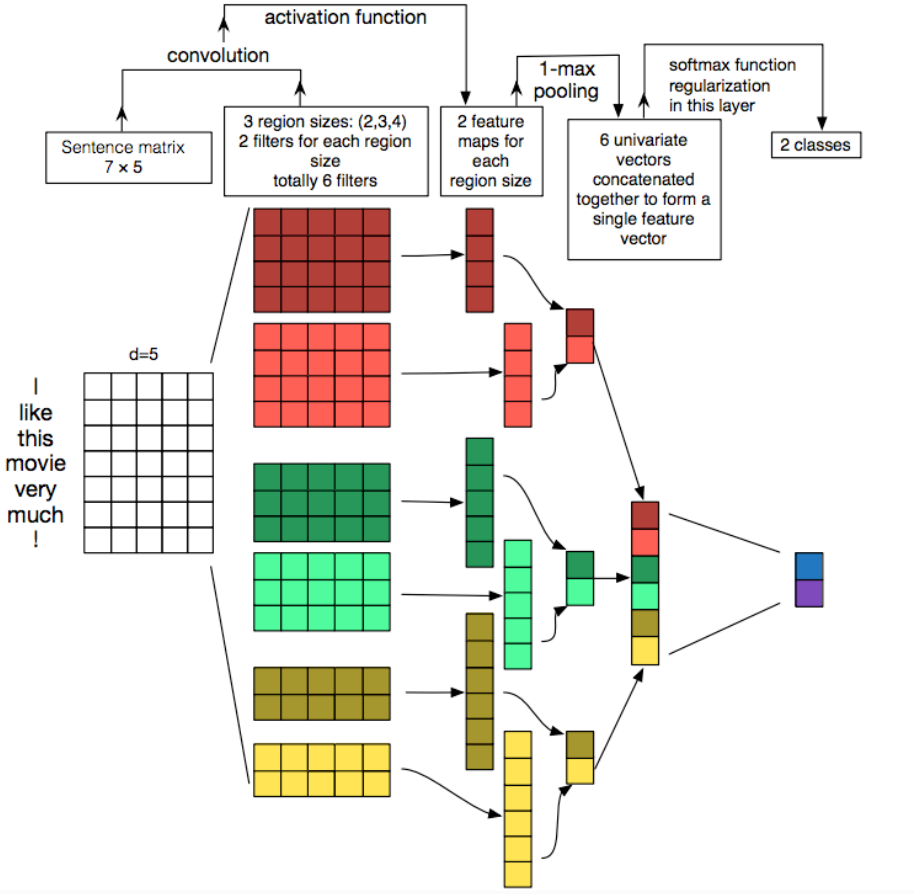

Figure 12: The CNN reference architecture used in the implementation [12] [30]

3.6 The Large Collection Learning Optimizer

Our overall goal is to be able to classify new events in our collections as we start having tweets coming in for such events. We want to develop a framework that can generate a new classifier as needed and can classify the collections that we ask the classifier to label. In particular, we want a human actor to specify the class label, provide the training and validation data, and specify the auxiliary and the final collection that is to be labelled. The framework should be able to generate classifiers based on the different approaches that we have mentioned in our design previously, and choose the best performing classifier to classify the final collection that we want to have labelled.

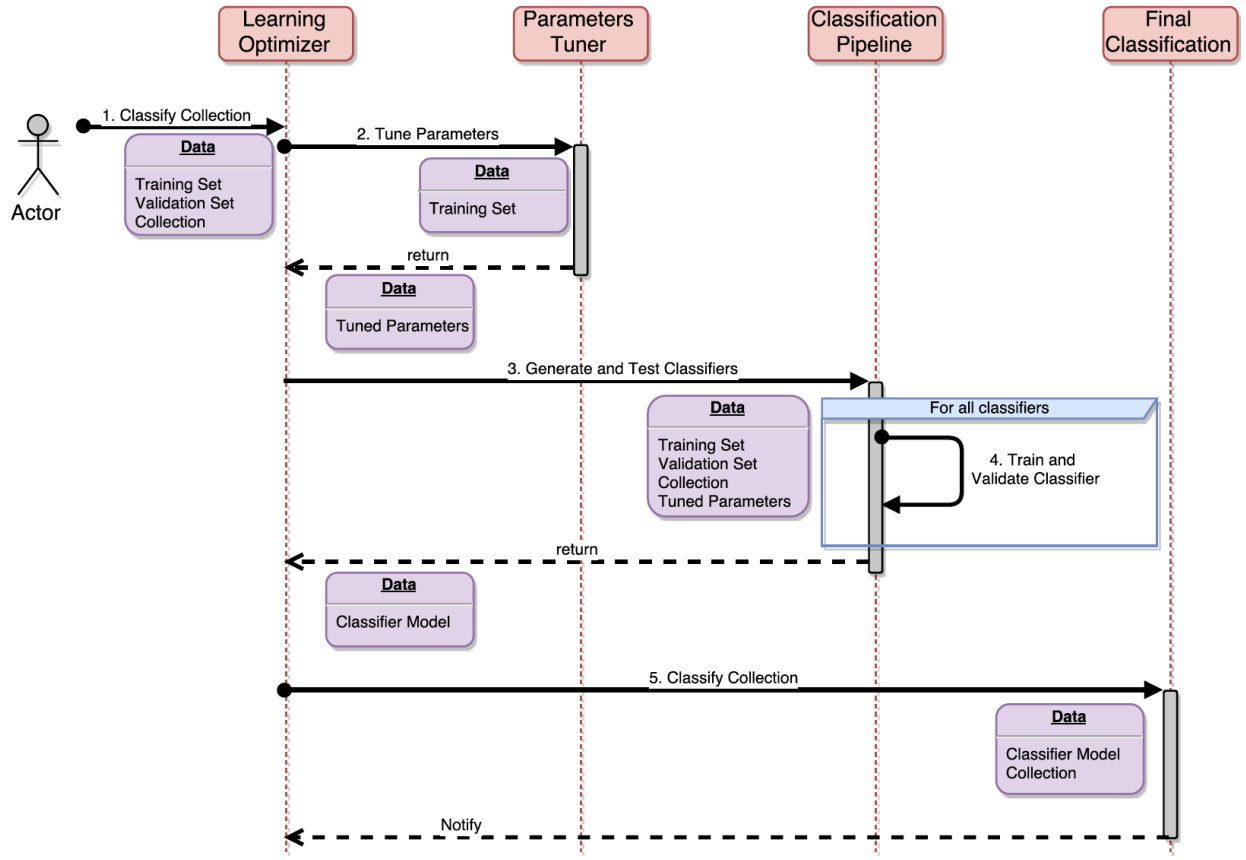

Figure 13: Sequence diagram for the Learning Optimizer methodology

The sequence diagram for the Learning Optimizer methodology is shown in Figure 13. The different steps are as follows.

- **Classify Collection** - The human actor annotates a small amount of training and validation data. The human actor also specifies the collection from which the auxiliary data can be sourced; this is also the collection that has to be finally classified. We assume that all the data that we process in this workflow has been pre-processed as per the design discussed in Section 3.3.1.
- **Tune Parameters** – During this step the parameters for the pre-classification steps are tuned. These include identifying the cosine similarity threshold and the window size as described in Section 3.4.4.
- **Generate and Test Classifiers** – During this step, the classification pipeline orchestrates the training and testing of all the classifiers.
- **Train and Validate Classifier** – This is a sub-step of the previous step. During this step, a given classifier is chosen and is trained. Once the training is complete as per the design covered in Section 3.4.3, we retain the model file. Once all the classifiers have been evaluated, the classifier with the best F1 score on the validation data is chosen as the one used for classifying the collection finally.
- **Classify Collection** – During this step, the classifier chosen in the previous step is used to classify all the tweets in this collection.

4 Datasets

This chapter describes the datasets used in our case studies.

4.1 Preliminary Case Study

This case study is used to help identify the choice of feature selection and classification method for classifying the tweets. We would use the best performing feature selection and classification methods identified in the preliminary case studies for our final large collection learning optimizer framework as described in Section 3.6.

4.1.1 The Dataset

The data for this case study is drawn from 3 broad categories, with 3 sub-categories for each broad category:

- Shootings
    a. Kentucky Shooting
    b. Newton Shooting
    c. Firefighter Shooting
- Hurricanes
    a. Hurricane Arthur
    b. Hurricane Sandy
    c. Hurricane Isaac
- Explosions
    a. China Factory Explosion
    b. Texas Plant Explosion
    c. Manhattan Explosion

The matching of the real-world events to class labels can be found in Table 2. The distribution of the specific classes for the sample data is shown in
Figure 14. This distribution shows the class imbalance in the sample set. Such an imbalance was intentionally created to reduce the bias of the classifier [30].

Table 2: Real World Events

Class Label	Real World Event	Description
0	Firefighter Shooting	A gunman ambushed firefighters at a house fire in the Rochester suburb of Webster, N.Y. on 14-Dec-2012
1	China Factory Explosion	A series of explosions in a container storage location on 12-Aug-2015 in Tianjin, China.
2	Kentucky Shooting	A school shooting that took place in Louisville, Kentucky on 30-Sep-2014.
3	Manhattan Explosion	An explosion that happened in New York City on 18-Sep-2016.
4	Newtown Shooting	A school shooting that happened in a Newtown, Connecticut on 14-Dec-2012.
5	Hurricane Sandy	A hurricane that devastated the East coast of the USA, formed on 22-Oct-2012.
6	Hurricane Arthur	A hurricane that hit the East coast of the USA, formed on 21-July-2014.
7	Hurricane Isaac	A hurricane that hit the US state of Louisiana, that formed on 21-Aug-2012.
8	Texas Plant Explosion	On April 17, 2013, an ammonium nitrate explosion occurred at the West Fertilizer Company storage and distribution facility in West Texas

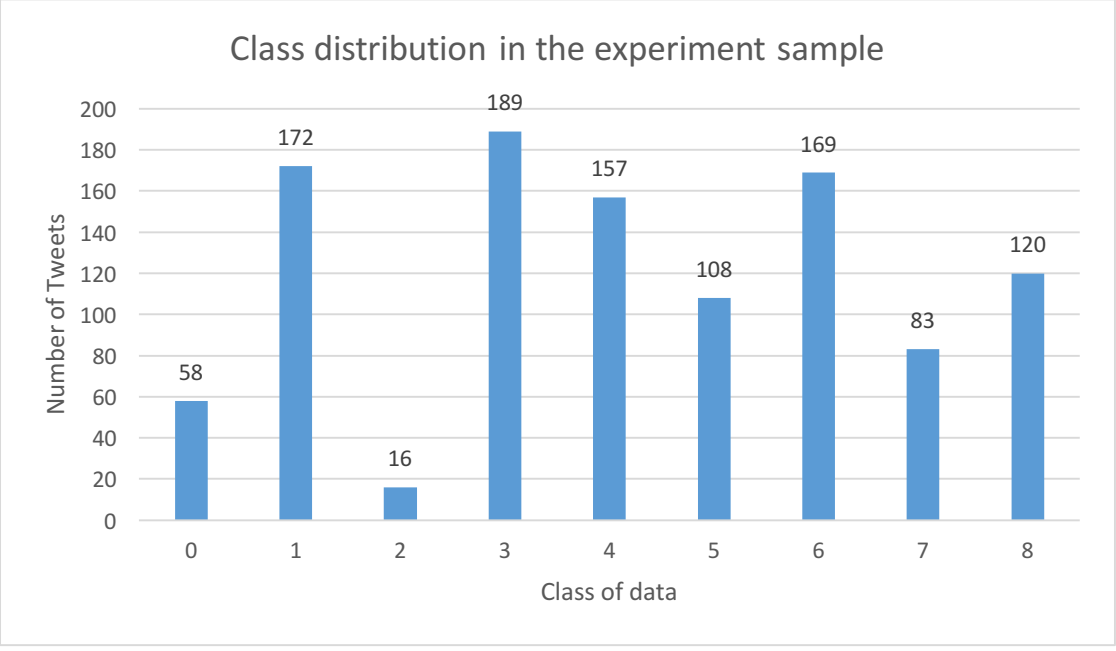

Figure 14: Class distribution in the experiment sample

4.2 Augmented Training Case Study

The high-level goal of the auxiliary experiments dataset is to measure the efficacy of the classification methodology across different categories that are not related to each another and are based on events that vary from being very specific to something very broad.

As part of our initial case studies, we use 5 datasets to perform empirical evaluation to ascertain whether the augmented training methodology results in any improvement in classification results. The next step in our case studies is to validate the consistency of our results. For this purpose, we curate 3 additional training sets for each of the 5 discrete real-world events and perform empirical evaluation of the classification results. The final step in our case studies pertains to using the large collection learning optimizer framework to classify another 2 real-world datasets in an automated way. The following sections describe each of the datasets in detail. Appendix B lists the location where the file artifacts related to this case study can be found.

4.2.1 The Dataset

We choose 5 datasets across different categories from our collection. Table 3 lists the different categories and the keywords that led to the respective collections. We randomly select about 500 tweets for each category from these collections for our experiments.

Table 3: The 5 datasets for the augmented training experiments

Category	Collection Keywords
Egyptian Revolution	#egypt
Ebola Outbreak	#ebola
Obesity	#obesity
Winter Storm	#winterstorm
Severe Weather	#severeweather

Once we have collected the tweets for these 5 datasets, we need to further divide the tweets -- for training, validation, and auxiliary data -- for each category. Table 4 shows the details of the different sets for each category. For each category, the distribution of the positive and negative examples for training and validation are shown in parentheses. We perform the pre-processing on the tweets as described in Section 3.3.1. For some of the tweets, the cleaning process clears all the content of the tweet since they contain only noise. These tweets are omitted from the training and auxiliary sets.

Table 4: The training, validation, and auxiliary datasets for each category

	Winter Storm	Severe Weather	Obesity	Ebola Outbreak	Egyptian Revolution
Training	13 (7+, 6-)	16 (7+, 9-)	13 (7+, 6-)	16 (7+, 9-)	22 (8+, 14-)
Validation	88 (52+, 36-)	91 (40+, 51-)	184 (79+, 105-)	51(25+, 26-)	38 (23+, 15-)

4.2.2 Classification Consistency Case Study

To perform our classification consistency experiments, we create three additional training datasets to ensure that we have variation in the training data and are still able to evaluate the classification performance for consistency.

4.2.3 The Dataset

Table 5 shows the distribution for each of the training sets. The validation and auxiliary data for each category comes from the same validation and auxiliary datasets as in Section 4.2.1.

Table 5: The distribution of the 3 training sets for each category

	Winter Storm	Severe Weather	Obesity	Ebola Outbreak	Egyptian Revolution
Training Set 1	21 (15+, 6-)	14 (10+, 4-)	25 (8+, 17-)	11 (6+, 5-)	44 (12+, 32-)
Training Set 2	16 (3+, 13-)	22 (4+, 18-)	18 (11+, 7-)	11 (5+, 6-)	15 (6+, 9-)
Training Set 3	20 (8+, 12-)	23 (12+, 11-)	15 (7+, 8-)	11 (5+, 6-)	21 (6+, 15-)

4.2.4 The Large Collection Optimizer Case Study

We choose 2 datasets from our collection that we have not used so far. The purpose of this case study is to classify a collection using very little training data. The approach described in Section 3.6 generates a classifier to be used to classify the whole collection and then classifies all the data in the collection. We randomly select about 800 tweets for each category from these collections for our experiments.

Table 6: The 2 datasets for the augmented training experiments

Category	Collection Keywords
Greece	#greece
Environment	#environment

Table 7 shows the distribution of the training, validation, and auxiliary datasets for this experiment.

Table 7: The training, validation and auxiliary datasets for each category

	Greece	Environment
Training	10 (10+, 7-)	20 (10+, 10-)
Validation	57 (18+, 39-)	42 (19+, 23-)
Auxiliary	600	691

5 Preliminary Case Studies

The goal of the preliminary case studies is to identify the best performing feature selection and classification methods.

5.1 Pre-processing the Tweets

As described in Section 3.3.1 we start with the case studies related to the pre-processing of the tweets. This step helps us decide whether pre-processing of the tweets improves the classification accuracy.

5.1.1 Setup

To conduct this experiment, we divided the hand labeled data into a training and a test dataset. The split used was 70% for the training and 30% for the test data. The same split data was run on both classifiers with the raw text and cleaned text. The dataset is described in Section 4.1. The pre-processing of the tweets was based on the Standford CoreNLP Toolkit [21].

The cleaning that was done for this experiment was:

- Lemmatization
- Stop word removal
- Hashtag removal
- Non-English character removal

The motivation behind cleaning in this way is discussed in Section 3.3.1.

5.1.2 Results

We can see the summary of the results in Table 8. Cleaning of the data yielded a large reduction in the misclassifications for both the Association Rules classifier as well as the Logistic Regression classifier. To be able to best classify the tweets in the collections to specific real world events we will employ pre-processing both on the training tweets as well as the tweets whose class we are predicting.[4]

Table 8: Cleaning experiment results

Classifier	% reduction in misclassifications
Word2Vec with Logistic Regression	28.6
Association Rules	50.8

Based on the results, we would pre-process all the text data that we process for classification purposes, going forward.

5.2 Association Rules

As described in Section 3.3.2, we use a classification method based on Association Rules. This case study tunes the Association Rules based classifier using the support threshold and observes

[4] A comparison of the accuracy of these classifiers can be found in section 5.3

the number of tweets that are predicted by rules and distance for a given support threshold. By looking at this we are able to infer the correct support threshold value that the rules in the Twitter domain need, to classify using Association Rules. The reason for wanting to classify by the rules instead of cosine similarity is that the rules can be applied much faster.

5.2.1 Setup

This experiment is performed with hand-labeled training data from the 3 different classes of shooting events. The data represents the following real world events. The dataset is a subset of that described in Section 4.1.:

- New York Firefighter shooting
- Connecticut School shooting
- Kentucky Accidental Child Shooting

The Association Rules classifier was run on the same dataset with different support value thresholds. The support threshold parameter dictates the Association Rules that will be used in prediction. Only rules that have support greater than the threshold will be used to predict tweets.

5.2.2 Results

Our implementation of Association Rules [11] uses rules that have 100% confidence. Also, for the cases that we don't have a rule, we use the cosine similarity for classification. The cosine similarity based classification in the Association Rules implementation has lower classification efficacy compared to the classification using the rules. Table 9 shows the percentage of the test set samples that were classified using Association Rules and cosine similarity, respectively. We can see that when there is no threshold on the rule values, 83.3% of the tweets are predicted by the Association Rules. As expected, when the support threshold is increased to 0.05 the percentage classified by rules decreased to 33.3% because many of the rules that had very small support, were excluded. We observe that the rules created have low support because the tweets themselves did not have a significant number of common terms even after cleaning. We want the classification to be done using Association Rules as much as possible. To this effect, used a support threshold very close to 0 for our case studies.

Table 9: Number of tweets classified by support thresholds

Support threshold	Percent predicted by rules	Percent predicted by cosine similarity
0.0	83.3	16.7
0.05	33.3	66.7
0.1	15.6	84.4
0.15	14.4	85.6
0.2	14.4	85.6

5.3 Classification using Word2Vec based Logistic Regression

The purpose of this case study is to determine which classifier performs better from an accuracy perspective out of the Word2Vec based Logistic Regression (LR) classifier and the Association Rules(AR) classifier on our hand-labeled data.

5.3.1 Setup

This experiment uses the same hand-labeled data for the [31] learning experiment as described in Section 4.1.

To fairly judge the performance of these classifiers we split up the hand-labeled data into 10 different 70% train - 30% test sets. This was accomplished by first splitting the data into 10 equal sets, then placing 7 of those into training and 3 into test. The remaining splits were generated by rotating the sets between training and testing so that each set was in both train and test for at least one of the splits. This procedure is explained pictorially in Figure 15.

Each of the splits was then run on each classifier to compare their results on the exact same training and test data.

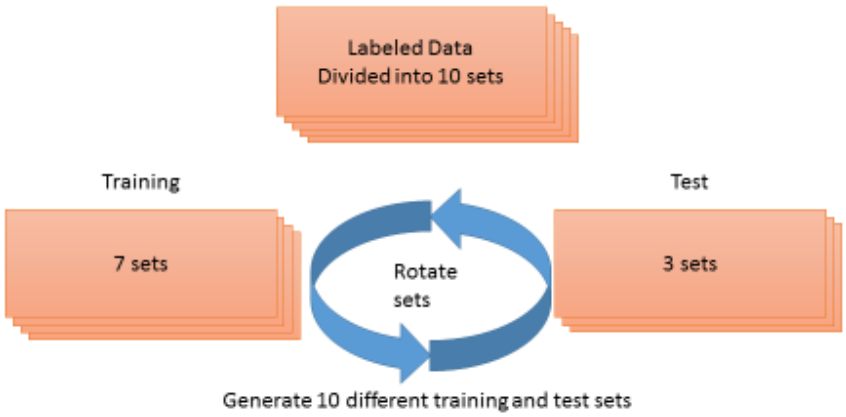

Figure 15: Accuracy experiment data generation

5.3.2 Results

Each classifier was run on all 10 experimental splits and the metrics Weighted F-Measure, Weighted Precision, and Weighted Recall were recorded. Table 10 shows the comparisons of the AR and LR classification results comparisons. The AR based classifier predicted 306 samples correctly and 14 samples incorrectly. The LR based classifier predicted 312 samples correctly and 8 samples incorrectly.

Table 10: AR and LR classification results comparison

AR \ LR	Correct	Wrong	total
Correct	299	7	306
Wrong	13	1	14
Total	312	8	320

The results from each classifier on a specific set as well as the averages are shown in Table 11. The averages calculated are shown in Figure 16. From these results, we can see that for all the metrics tested, the averages were higher (or equal in one case) for the Word2Vec with Logistic Regression classifier.

Table 11: Comparative experiment results

	Word2Vec with Logistic Regression			**Association Rules**		
Dataset	Weighted F1	Weighted Precision	Weighted Recall	Weighted F1	Weighted Precision	Weighted Recall
10	0.97	0.97	0.97	0.95	0.96	0.96
9	0.96	0.96	0.96	0.91	0.92	0.92
8	0.94	0.95	0.94	0.89	0.90	0.90
7	0.97	0.97	0.97	0.97	0.97	0.97
6	0.96	0.97	0.96	0.88	0.90	0.89
5	0.96	0.96	0.97	0.90	0.91	0.91
4	0.97	0.98	0.97	0.85	0.88	0.87
3	0.95	0.95	0.95	0.88	0.88	0.90
2	0.95	0.96	0.95	0.91	0.91	0.92
1	0.96	0.96	0.96	0.86	0.88	0.87

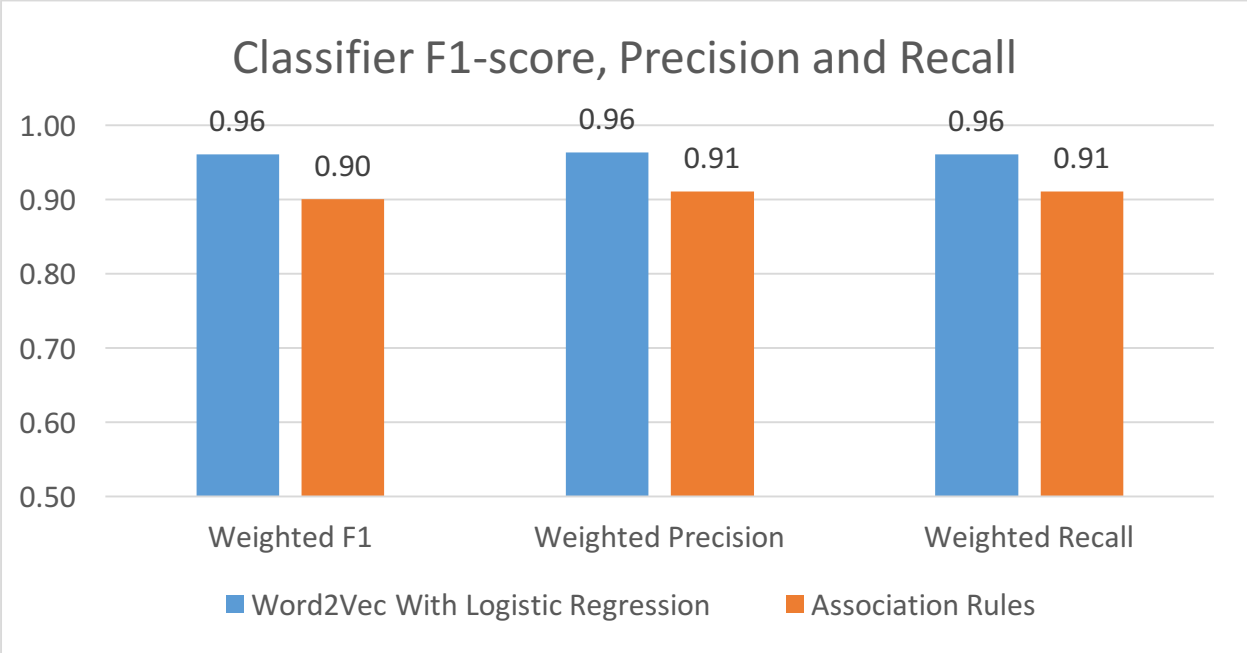

Figure 16: Average classifier F1-score, Precision and Recall

Test of Significance

We performed a test of significance to determine if the results from the accuracy experiment reported in Table 11 were statistically significant. This test is used to determine if two sets of data are from a same or different distribution. The null hypothesis for this test is that the average weighted F1 scores are equal. The result of this test is a p-value, which is the probability of obtaining a difference at least as large as what was observed assuming the null hypothesis. A higher p-value indicates that the null hypothesis is likely to be true, while a lower p-value indicates that the null hypothesis is likely to be rejected. A normal p-value that will determine if the null hypothesis should be rejected is less than 0.01.

We performed a 2-tailed t-test and found that the resulting p-value was 0.0006. Given the results of this test we reject the null hypothesis and conclude that the difference between the average weighted F1 scores between the two classification techniques is statistically significant.

5.4 Probability Experiment

The purpose of this experiment is to analyze the distribution of probabilities for the Word2Vec based Logistic Regression classifier predictions on the test set. This will let us see how well it performs on sub-categories that have the same parent category.

5.4.1 Setup

For this experiment, we used one of the sets from the same training test data split from the cleaning experiment discussed in Section 4.1. The data used was the pre-processed data, as it had produced more accurate predictions for the Logistic Regression classifier.

5.4.2 Results

The distribution of the probabilities of the test set can be found in Figure 17. One use of these probabilities is to set a threshold where only the tweets with probabilities above the threshold will be classified into their respective classes. Any tweet below the probability threshold will be ignored for labelling purposes. The first 3 bars in the figure represent the number of examples that were classified as belonging to either of 4, 3, or 2 classes, respectively. This means that the classifier is not absolutely certain which one class the tweets belong to. It makes a random choice between the probable classes that have the same probabilities. Table 12 shows the distribution of the probabilities along with the percentage of samples falling within a certain probability interval. If we set a threshold to be 0.3, we will be able to classify about 87% of the tweets in the test set, but 76% of them are of the nature where the classifier believes that they can belong to more classes than what has been labelled. The choice of this threshold represents the precision-recall tradeoff. If we set a high threshold such as 0.9 the system will label only 74 tweets out of a total of 321, while the other tweets will be ignored for labelling. This would lead to high precision as the tweets with a probability higher than the threshold will only be assigned a classification label. But, on the other hand, the tweets with a probability lower than the threshold will not be labelled and hence result in lowering the recall to 0.23.

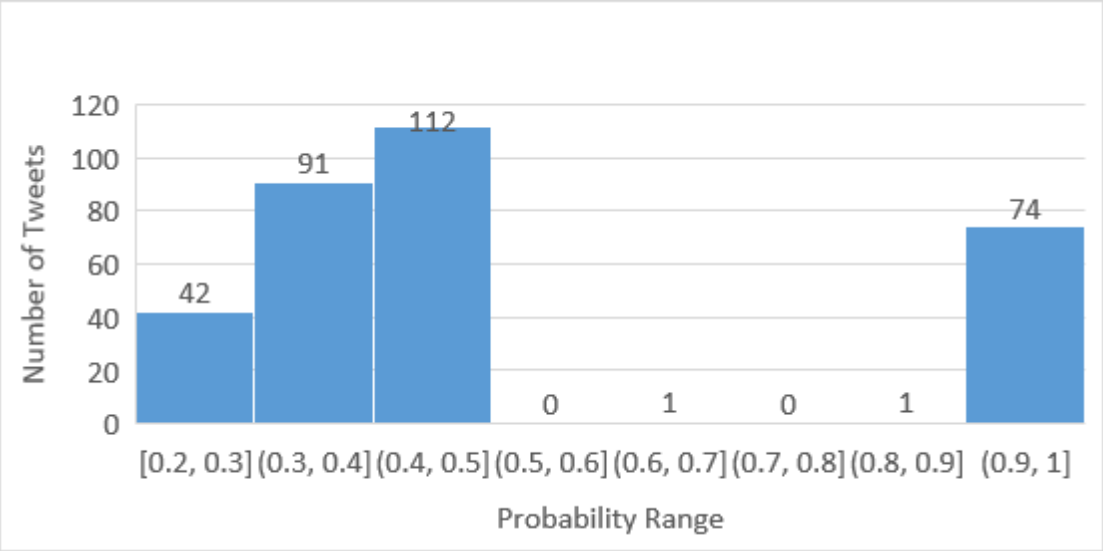

Figure 17: Probability distribution of predicted tweets

Table 12: Distributions of classification probabilities

Range	Number of instances	% of samples
0.2 – 0.3	42	13
0.3 – 0.4	91	28
0.4 – 0.5	112	35
0.5 – 0.6	0	0
0.6 – 0.7	1	0
0.7 – 0.8	0	0
0.8 – 0.9	1	0
0.9 – 1.0	74	23
Total	**321**	

We believe that the distribution across the different probability ranges is happening because of two reasons.

- **Limited number of word vectors** – We trained our word vectors using about 750 training tweets as described in Section 3.3.3. During classification of test data, we omit all the words that the Word2Vec model has not trained on. This results in the culling of words that would have otherwise contributed to the probability for some class. We observed instances where a tweet related to hurricane was talking about hurricane in general and these tweets were assigned equal probabilities for all classes related to hurricanes. Having a classifier trained on a richer corpus will be able to assign a high probability to one class.
- **Training on broad categories** – In our experiments, we observe that the probabilities indicated a tweet was more likely to belong to multiple classes than belong to a specific class. For example, a tweet that just had the presence of the word "hurricane" was classified with a probability of 0.33 across all the three specific hurricane classes that we had. This happens because the classifier thinks that the 3 classes are equally probable. Training a class specific to a broad category like "hurricane" would ensure that a tweet that is not specific to a particular hurricane gets a probability score close to 1. This would help classify

a tweet that could be of a broad nature into the appropriate broad category, instead of the probability scattering equally across specific classes.

If a lower threshold is used, the classifier is not sure what label to assign to a given tweet. The label in this case can be any of the equally probable classes. If we select a high threshold, we are very confident about our prediction, but we are not getting a lot of coverage in our classification. Figure 18 shows the distribution of the number of classes based on the percentage distribution in Table 12. We have only shown the probability ranges where the percentage is more than 10% of the whole sample size. If we choose a probability threshold of 0.9, then we only get a coverage of 23%. These predictions are very precise, but this precision comes at the cost of recall. If we select a threshold of 0.3, we get a coverage of 87%. The increase in recall comes at the cost of precision. We believe that the user requirements should be the best factor in defining what the ideal threshold should be.

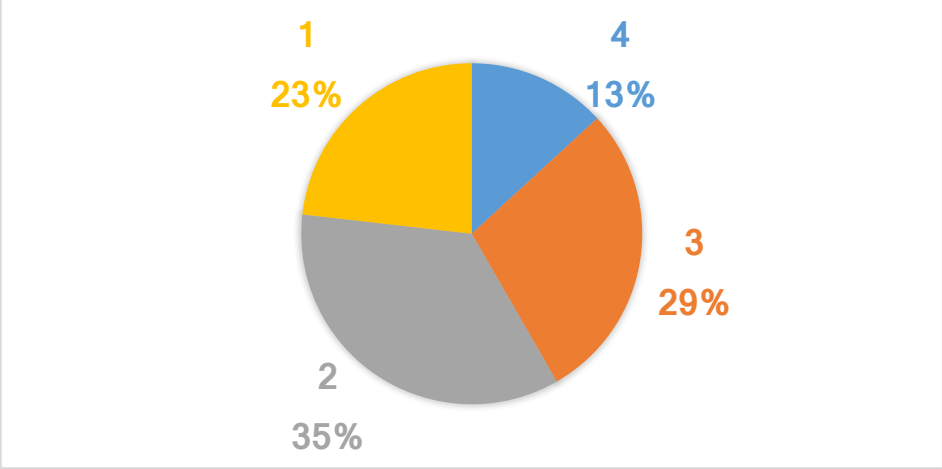

Figure 18: Multi-class assignment distribution

5.5 Inter-Classifier Mutual Agreement

In this experiment, we measure the level of agreement between the classifiers that we are evaluating. We use the results of the accuracy experiments in Section 5.3.2 to compare the classification results between our logistic regression classifier (LR) and the Association Rules based classifier (AR). We use the Kappa statistic [3] to calculate the inter-classifier agreement for our experiment.

Table 13: Kappa inter-classifier agreement

AR \ LR	Correct	Wrong	total
Correct	300	7	307
Wrong	13	1	14
total	313	8	321
P(A)	0.94		
P(Correct)	0.97		
P(Wrong)	0.03		
P(E)	0.93		
Kappa	0.06		

$$kappa = \frac{P(A) - P(E)}{1 - P(E)}$$

Figure 19: Formula to compute Kappa

Table 13 shows the distribution of the classification results. The LR and AR classifier agree on 299 correct classifications and 1 misclassification. They disagree on 7 and 13 misclassifications for LR and AR, respectively. We compute the marginal values P(A) and P(E) and use the formula in Figure 19 to compute the Kappa value. Table 14 shows the different ranges of the Kappa values and the definition of agreement for the different range of Kappa values. For our computed value of 0.06, it signifies that there is only a slight agreement between the two classifiers.

Table 14: Kappa agreement definitions

Kappa	Definition of agreement
> 0.8	Good
0.67 – 0.8	Fair
< 0.67	Slight

We believe the reason for slight agreement between the classifiers is because of the fact that the classifiers agreed only once during misclassification compared to 299 correct classifications. This imbalance skews the Kappa statistic. A Kappa computation on a larger set would definitely give us more data for the Kappa statistic to be of significance.

For our case studies, we observe that there are a very few misclassified samples owing to the high accuracies achieved by the classifiers. Owing to this fact, there are even fewer misclassified samples where the classifiers have an agreement on. Due to this very fact, the imbalance between the agreed correct classifications and misclassifications causes the Kappa statistic to be skewed. We decide not to perform any analysis related to inter-annotator agreement for the classifiers in any of the case studies going forward.

5.6 Runtime Comparison

In addition to the accuracy of the classifiers, another property to consider is the time that the classifier takes to predict the classes for large sets of data. This is an important consideration because we have a very large number of tweets to be classified, and it is imperative for us to efficiently classify these into their respective classes within a reasonable time.

Being able to classify efficiently means that we can reclassify large parts of our collection as we add additional training data. The classifiers must also be able to classify at a rate faster than the production of tweets so that they can be run on future tweets.

5.6.1 Setup

For this experiment, we reuse the pre-processed hand-labeled data described in Section 4.1. Instead of splitting the set into training and test we used all of the labeled data as training for the classifiers. We then selected a large set of 640,000 unlabeled tweets from the IDEAL collection for prediction. Each classifier is instrumented so that only the prediction time is tested and the training of the classifiers is not included in this measurement. Both classifiers performed predictions on the same datasets.

5.6.2 Results

The results can be seen in Figure 20. We observe that the Logistic Regression based classifier is relatively faster for the smaller collections, but the Association Rules based classifier performs much better on larger collections. As the collections for this project will have millions of tweets, the performance on large datasets is of top concern.

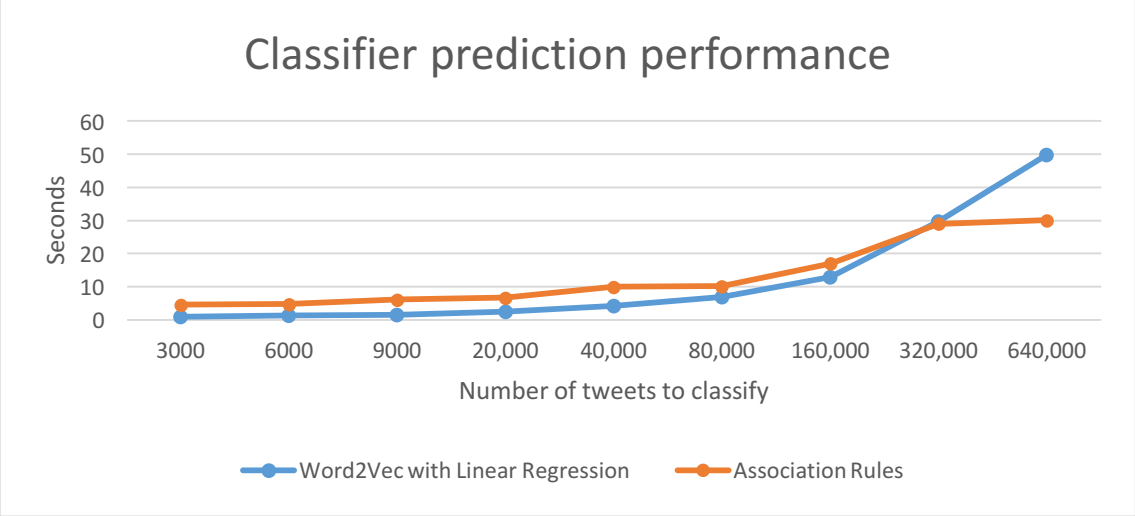

Figure 20: Performance of the classifiers on different number of tweets

To be able to use the Logistic Regression classifier effectively we must reduce the runtime it experiences over large datasets. We optimize the classifier using the partitioning and caching methods mentioned in Appendix A9. The results of the experiment when run with the optimized classifier can be seen in Figure 21. Applying the optimization shows an improvement, with 57%

less time spent in prediction than the original classifier, as well as a 14% faster prediction time than the Association Rules classifier when the number of tweets is 640,000.

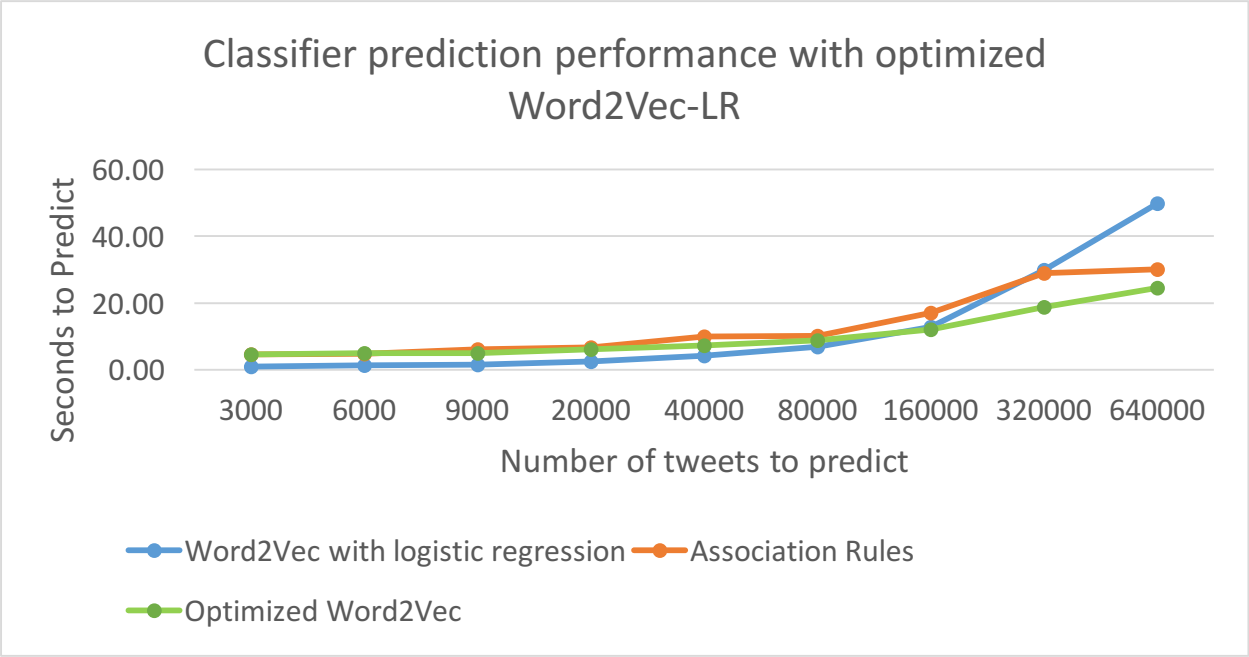

Figure 21: Performance of the classifiers including optimization

Table 15 shows the time taken by the Logistic Regression and Association Rule based classifier and optimized version of the Logistic Regression classifier. The optimized version does have a bit of overhead costs related to the caching and partitioning and this is apparent in the classification involving small datasets as shown in the figure above.

Table 15: Time taken in seconds by the classifiers for the different collection types

Number of tweets in collection	Word2Vec with Logistic Regression (seconds)	Association Rules (seconds)	Optimized Word2Vec (seconds)
3000	0.92	4.60	4.63
6000	1.25	4.71	4.96
9000	1.59	6.13	4.97
20000	2.48	6.66	6.10
40000	4.23	9.99	7.27
80000	6.95	10.17	8.88
160000	12.85	17.06	12.09
320000	29.80	28.97	18.77
640000	49.87	30.10	24.53

6 Augmented Training Case Studies

6.1 Similarity Measure – Cosine Similarity

In this case study, we observe the cosine similarities within the training data of the dataset to identify the similarity threshold and window size parameters as described in Section 3.4.4.

6.1.1 Setup

We use the datasets described in Section 4.2. We choose the "Egyptian Revolution" and the "Obesity" datasets for our preliminary experiments with identifying the parameters. We integrate the code with our classification pipeline to incorporate the automatic discovery of the threshold and window size parameters for all our collections at runtime. The results are shown for the Word2Vec vectors based on the Google News corpus [32]. We have to set the cosine similarity thresholds and window sizes for all the data sets and vector types.

6.1.2 Results

Table 16 and Table 17 show the average cosine similarities values of the positive and negative training examples with the positively labelled training examples, for the "Egyptian Revolution" dataset. We observe that the average cosine similarities of all the positive examples are above 0.44, and the negative examples are below this value. We want to ignore any examples that is at this border of this threshold. We would ideally want to label the examples that are well above and below the cosine similarity value of 0.44 as positive and negative, respectively. We set a window size around this border threshold and discard any auxiliary examples for training that fall in this window.

Table 16: Egyptian Revolution – Cosine similarities for the positive training examples

Cosine similarity	Tweet Text
0.44	égyptien vote divisive vote tahrir
0.52	egypt sultan alqassemus incredible photo anti morsy protester tahrir via
0.46	egypt get check manufacture day
0.45	Another Egypt army clear presidential palace islamist muslim brotherhood
0.54	muslim brotherhood continue kill coptic christian cant say surprise redo insane egypt
0.48	moftasa graph rise albersaber bishoykamel database arrest threaten blogger egypt
0.54	egyptian challenge morsi nationwide protest via reuter egypt
0.52	egypt morsi call dialogue say violence solution political impasse

Table 17: Egyptian Revolution – Cosine similarities for the negative training examples

Cosine similarity	Tweet Text
0.4	fukushima come clean editorial unthinkable think britain plan new generation
0.39	fukushima harm commerce russium reject radioactive japanese car httptconrbejeuqsc via healthranger
0.35	japan brace tepco prepare critical phase fukushima cleanup youtube news
0.33	gigaoktopus durch atom katastrophe fukushima transatlantisch fre via youtube
0.3	fukushima business usual japan nuclear industry turkey purchase new plant
0.31	ebola salta capital petrolera nigeria con caso
0.36	government publish fact sheet zmapp anti ebola drug
0.37	new ebola case may slow sierra leone say
0.44	people worry bring back american africa ebola remember flight ban impose
0.28	cuban doctor return home ebola treatment

Table 18: Obesity – Cosine similarities for the positive training examples

Cosine similarity	Tweet Text
0.68	heart failure hypertension med may help decrease obesity type drug normally use obesity news
0.65	great article nancihellmich doctor encourage treat obesity like disease
0.6	kennethhorpe pfcd overweight obesity cost company billion lose productivity
0.65	omics group medical conference fitness weight loss obesity conference fight fat healthy
0.64	right call course list numerous obesity relate health unhealthy
0.63	portion size cat could share one extra large meal satisfy obesity fat

Table 18 and Table 19 show the average cosine similarities of the positive and negative training examples with the positively labelled training examples, for the "Obesity" dataset. We observe that the average cosine similarities for the positive training examples are all above 0.63 and they are less than 0.39 for the negative training examples. There is a clear demarcation between the positive and negative training examples in this case.

Table 19: Obesity – Cosine similarities for the negative training examples

Cosine similarity	Tweet Text
0.35	structural basis ebola viral pathogenesis talk erica ebola
0.3	hundred tornado could hit midwest south week weather storm
0.36	arrest expect baltimoreriots recent raymond carter face federal offense
0.28	thegrio pace university quarterback suspend confederate flag nazi salute photo
0.36	watch live hurricane sandy cause partial crane collapse midtown manhattan
0.18	hamradionow hurricane sandy
0.39	climate change cause hurricane

Table 20 lists all the cosine similarity threshold and window size values for all the datasets and their corresponding vector types. The original column denotes the cosine similarity threshold and window size values for the data that is described in Section 4.2.1. The columns 1, 2, and 3 denote the cosine similarity threshold and window size values for the data that is described in Section 4.2.2.

Table 20: Cosine similarities thresholds and window sizes – All datasets

Dataset Categories	Vector Types	Training Dataset Thresholds and Window Sizes			
		Original	1	2	3
Winter Storm	**Google**	0.45, 0.20	0.45, 0.20	0.57, 0.24	0.50, 0.20
	Local	0.38, 0.25	0.33, 0.20	0.55, 0.35	0.41, 0.30
Severe Weather	**Google**	0.60, 0.30	0.60, 0.25	0.60, 0.25	0.55, 0.20
	Local	0.34, 0.15	0.40, 0.20	0.45, 0.30	0.44, 0.25
Obesity	**Google**	0.60, 0.25	0.50, 0.25	0.50, 0.20	0.60, 0.25
	Local	0.33, 0.25	0.48, 0.20	0.45, 0.25	0.45, 0.30
Egypt	**Google**	0.49, 0.15	0.50, 0.20	0.60, 0.35	0.55, 0.20
	Local	0.50, 0.20	0.45, 0.25	0.55, 0.30	0.50, 0.18
Ebola	**Google**	0.55, 0.15	0.55, 0.20	0.55, 0.20	0.55, 0.20
	Local	0.32, 0.12	0.45, 0.30	0.45, 0.20	0.45, 0.30

6.2 Training a Classifier with Auxiliary Training Data

As part of this case study, we train a classifier supplemented by additional training data that is sourced from the auxiliary dataset. We add training tweets in sets of 20 in each iteration. We perform 20 iterations in total. We exit out of the iterative process if the classifier achieves an F1 score of 0.98. The labelling of the auxiliary data tweets being added has been described in detail in Section 3.4.4.

6.2.1 Setup

We use the datasets described in Section 4.2. We choose a dataset from each of the 5 classes to compare and contrast the classifier performance with the following combination of feature selection and classification technique.

- Logistic Regression with Word Vectors generated from the Google News corpus.
- Logistic Regression with Word Vectors generated from the Auxiliary dataset of the same class.
- CNN with Word Vectors generated from the Google News corpus.

The purpose of this experiment is to ascertain what combination of word vector type and classification method works the best across datasets. We also want to answer the question whether

it is possible to have one universal choice of word vector type and classification technique. We use the technique described in Section 3.4.3 to perform this case study.

6.2.2 Results

Table 21 shows the results obtained through classification using the various combinations of word vectors and classification techniques. The values in the table show the initial F1-score obtained without any auxiliary training data and the maximum F1-score obtained from the whole run of 20 iterations. If the maximum F1-score was obtained via the use of auxiliary training data, the iteration number is mentioned. The results in bold for each dataset highlights the best result achieved.

Table 21: Classification results across 5 datasets

Methodology	Winter Storm	Severe Weather	Obesity	Ebola Outbreak	Egyptian Revolution
Logistic Regression with Local Word Vectors	Initial F1: 0.58 Max F1: 0.94 (Iteration: 6)	Initial F1: 0.76 Max F1: 0.78 (Iteration: 4)	Initial F1: 0.9 Max F1: 0.93 (Iteration: 12)	Initial F1: 0.86 Max F1: 0.9 (Iteration: 5)	Initial F1: 0.6 Max F1: 0.83 (Iteration: 14)
Logistic Regression with Google Word Vectors	Initial F1: 0.96 Max F1: 0.96	Initial F1: 0.93 Max F1: 0.98 (Iteration: 2)	Initial F1: 0.92 Max F1: 0.95 (Iteration: 8)	Initial F1: 0.86 Max F1: 0.9 (Iteration: 4)	**Initial F1: 0.91 Max F1: 0.97 (Iteration: 4)**
CNN with Google Word Vectors	**Initial F1: 0.84 Max F1: 0.99 (Iteration: 4)**	**Initial F1: 1.0 Max F1: 1.0**	**Initial F1: 0.96 Max F1: 0.96**	**Initial F1: 0.94 Max F1: 0. 94**	Initial F1: 0.88 Max F1: 0.89 (Iteration: 3)

As is apparent from the above results, the CNN classifier with Word Vectors derived from the Google News corpus provides the best results in 4 of the 5 datasets.

6.2.3 Test of Significance

Based on the results in Section 6.2.2, we clearly observe that there are improvements provided by the augmented data based classification methodology. We want to perform a test of significance on the improvements. We perform a pair-wise 1-tailed t-test to ascertain whether the augmentation based methodology results in any statistically significant improvements in classification. We use the comparison of the initial F1-score to the best F1-score for each of the classifiers across all the 5 datasets. Table 22 shows the result of this test.

Table 22: Test of Significance results for the effect of augmented training

	p-value	Interpretation
LR-Local	0.06	Not quite statistically significant
LR-Google	0.01	Statistically significant
CNN-Google	0.17	Not statistically significant

We can interpret from the above results that only the classifier based on Logistic Regression coupled with the Google Word2Vec feature vectors shows significant F1-score improvements

across all datasets. The p-value for the LR-Local classifier is just above 0.05 but from the results in Section 6.2.2, it is very evident that the improvements in the F1 scores for most of the datasets were more than marginal. We can attribute this to the low sample size since we have only 5 such observations. We will investigate this further in Section 6.4.3 as we can perform this test with an additional 3 observations.

We also perform a test of significance on the classifiers in an inter-classifier basis. That is, we compare the different classifiers against each other in pairs and compare their best F1-scores across all the 5 datasets.

Table 23: Test of Significance results for the 3 classifiers on an inter-classifier basis

	p-value	Interpretation
LR-Local vs. LR-Google	0.06	Not quite statistically significant
LR-Google vs. CNN-Google	0.43	Not statistically significant
LR-Local vs. CNN-Google	0.04	Statistically significant

We can clearly see in Table 23 that the classification result differences between LR-Local and CNN-Google are statistically significant, which correlate well with the results that we observe in Section 6.2.2. Also, the inter-classifier p-value between the LR-Local and LR-Google classifiers is marginally above 0.05. We cannot make a strong assertion about the statistical significance currently, but we believe that a few more observations might help us make a stronger assertion about it. We will investigate this further in Section 6.4.3 as we can perform this test with an additional 3 observations.

6.3 Significance of Word Sequencing

We want to ascertain whether the CNN classification technique performs any better when we use the bi-gram or tri-gram representations for the tweet texts.

6.3.1 Setup

We use the datasets described in Section 4.2. We choose one dataset from each of the 5 classes and set up the CNN to use the unigram, bigram, and trigram representation of the features.

6.3.2 Results

Table 24 shows the results of the n-gram feature representation across the 5 datasets. It is apparent that the bigram and trigram representation don't provide any improvement over the unigram representation in 4 of the 5 datasets. The unigram representation had the best results in 4 datasets. We will only use the unigram representation with CNN in our future experiments.

Table 24: n-gram feature representation results across 5 datasets

Methodology	Winter storm	Severe Weather	Obesity	Ebola Outbreak	Egyptian Revolution
CNN with Google Word Vectors – Unigram Model	**Initial F1: 0.84 Max F1: 0.99 (Iteration: 4)**	**Initial F1: 1.0 Max F1: 1.0**	**Initial F1: 0.96 Max F1: 0.96**	**Initial F1: 0.94 Max F1: 0. 94**	Initial F1: 0.88 Max F1: 0.89 (Iteration: 3)
CNN with Google Word Vectors – Bi-gram Model	Initial F1: 0.81 Max F1: 0.88 (Iteration: 3)	Initial F1: 0.99 Max F1: 0.99	**Initial F1: 0.96 Max F1: 0.96 (Iteration: 5)**	**Initial F1: 0.94 Max F1: 0.94**	Initial F1: 0.91 Max F1: 0.91
CNN with Google Word Vectors – Tri-gram Model	Initial F1: 0.77 Max F1: 0.95 (Iteration: 7)	**Initial F1: 1.0 Max F1: 1.0**	Initial F1: 0.94 Max F1: 0.95 (Iteration: 1)	**Initial F1: 0.94 Max F1: 0.94**	**Initial F1: 0.89 Max F1: 0.94 (Iteration: 8)**

6.4 Consistency of Results

We want to ensure that it is not by chance that we have obtained the results described in Section 6.2.2. We want to measure the consistency of the results across different training datasets for all the real-world events.

6.4.1 Setup

We use the 3 additional training datasets described in Section 4.2.2 along with the same auxiliary and validation datasets described in Section 4.2.1.

We have the Training (3 independently curated sets), Validation, and Auxiliary data across the 5 categories. We want to ascertain whether improvements provided by the augmented training methodology are consistent across the categories for all the 3 independently curated datasets. We perform classification across the 5 categories for all the 3 training sets and compare the results for each category independently. For the significance tests, we do a pair-wise 1-tailed t-test comparing the initial and best F1 scores for each classification technique for a given real-world event. This test is done to measure the statistical significance of the improvements in classification accuracy for the augmented training methodology.

Table 25: Consistency case study dataset

	Winter Storm	Severe Weather	Obesity	Ebola Outbreak	Egyptian Revolution
Training Set 1	21 (15+, 6-)	14 (10+, 4-)	25 (8+, 17-)	11 (6+, 5-)	44 (12+, 32-)
Training Set 2	16 (3+, 13-)	22 (4+, 18-)	18 (11+, 7-)	11 (5+, 6-)	15 (6+, 9-)
Training Set 3	20 (8+, 12-)	23 (12+, 11-)	15 (7+, 8-)	11 (5+, 6-)	21 (6+, 15-)
Validation	88 (52+, 36-)	91 (40+, 51-)	184(79+, 105-)	51(25+, 26-)	38 (23+, 15-)
Auxiliary	500 (25%+)	500 (25%+)	500 (25%+)	500 (25%+)	500 (25%+)

6.4.2 Results

6.4.2.1 Ebola Outbreak

Table 26: Ebola dataset case study results

	Training Dataset 1	Training Dataset 2	Training Dataset 3
LR with Local vectors	Initial F1: 0.42 Max F1: 0.84 (Iteration: 1)	Initial F1: 0.86 Max F1: 0.9 (Iteration: 18)	Initial F1: 0.82 Max F1: 0.86 (Iteration: 14)
LR with Google vectors	Initial F1: 0.69 Max F1: 0.88 (Iteration: 6)	Initial F1: 0.9 Max F1: 0.9	Initial F1: 0.8 Max F1: 0.86 (Iteration: 1)
CNN with Google vectors	**Initial F1: 0.86 Max F1: 0.92 (Iteration: 1)**	**Initial F1: 0.94 Max F1: 0.94**	**Initial F1: 0.88 Max F1: 0.94 (Iteration: 1)**

Ebola outbreak case study observations

- CNN with Google vectors has the best results. This is consistent with our previous experiment.
- The Max F1 obtained in the previous experiment in Section 6.2.2 for the Ebola dataset was 0.94, which is consistent with the results we have here.
- Overall, we observe very slight difference between the maximum F1 scores achieved for the 3 independent training sets.

Table 27: Test of Significance – Initial vs. Max F1 score comparison between the 3 datasets

	p-value	Interpretation
LR with Local vectors	0.16	Not statistically significant
LR with Google vectors	0.14	Not statistically significant
CNN with Google vectors	0.09	Not statistically significant

All the p-values that we obtain by comparing the initial and best F1 scores for each classifier as shown in Table 27 are not significant. These correlates well with our case study results that we observe in Table 26 as the improvement using the augmented training methodology was very marginal. But this cannot be strongly asserted either since we have a very small sample size.

6.4.2.2 Egyptian Revolution

Table 28: Egyptian Revolution dataset case study results

	Training Dataset 1	Training Dataset 2	Training Dataset 3
LR with Local vectors	Initial F1: 0.78 Max F1: 0.86 (Iteration: 2)	Initial F1: 0.56 Max F1: 0.66 (Iteration: 4)	Initial F1: 0.86 Max F1: 0.88 (Iteration: 1)
LR with Google vectors	**Initial F1: 0.87 Max F1: 0.92 (Iteration: 16)**	**Initial F1: 0.86 Max F1: 0.89 (Iteration: 2)**	**Initial F1: 0.84 Max F1: 0.89 (Iteration: 12)**
CNN with Google vectors	Initial F1: 0.76 Max F1: 0.86 (Iteration: 1)	Initial F1: 0.76 Max F1: 0.85 (Iteration: 9)	Initial F1: 0.83 Max F1: 0.86 (Iteration: 2)

Egypt case study observations

- LR with Google vectors has the best results. This is consistent with our previous experiment.
- The Max F1 obtained in the previous experiment was 0.94, whereas we have a slightly lower F1 across all the 3 different datasets.
- Overall, we observe no significant difference between the maximum F1 scores achieved for the 3 independent training sets.

Table 29 Test of Significance – Initial vs. Max F1 score comparison between the 3 datasets

	p-value	Interpretation
LR with Local vectors	0.05	***Statistically significant***
LR with Google vectors	0.01	***Statistically significant***
CNN with Google vectors	0.04	***Statistically significant***

All the p-values that we obtain by comparing the initial and best F1 scores for each classifier as shown in Table 29 are significant. They correlate well with our case study results that we observe in Table 28 as we see considerable improvement using the augmented training methodology. But this cannot be strongly asserted either since we have a very small sample size.

6.4.2.3 Obesity

Table 30: Obesity dataset case study results

	Training Dataset 1	Training Dataset 2	Training Dataset 3
LR with Local vectors	Initial F1: 0.87 Max F1: 0.93 (Iteration: 2)	Initial F1: 0.8 Max F1: 0.92 (Iteration: 15)	Initial F1: 0.8 Max F1: 0.89 (Iteration: 4)
LR with Google vectors	Initial F1: 0.93 Max F1: 0.93	Initial F1: 0.75 Max F1: 0.95 (Iteration: 5)	Initial F1: 0.9 Max F1: 0.93 (Iteration: 16)
CNN with Google vectors	**Initial F1: 0.94 Max F1: 0.96 (Iteration: 8)**	**Initial F1: 0.87 Max F1: 0.97 (Iteration: 4)**	**Initial F1: 0.94 Max F1: 0.95 (Iteration: 2)**

Obesity case study observations

- CNN with Google vectors has the best results. This is consistent with our previous experiment.
- The Max F1 obtained in the previous experiment was 0.9, whereas we have a higher F1 across the 3 different datasets.
- Overall, we observe no significant difference between the maximum F1 scores achieved for the 3 independent training sets.

Table 31: Test of Significance – Initial vs. Max F1 score comparison between the 3 datasets

	p-value	Interpretation
LR with Local vectors	0.01	***Statistically significant***
LR with Google vectors	0.17	Not statistically significant
CNN with Google vectors	0.13	Not statistically significant

Two out of the three p-values that we obtain by comparing the initial and best F1 scores for each classifier as shown in Table 31 are not significant. They correlate well with our case study results that we observe in Table 30 as the improvement using the augmented training methodology was very marginal for the LR-Google and the CNN classifiers. But this cannot be strongly asserted either since we have a very small sample size.

6.4.2.4 Severe Weather

Table 32: Severe Weather dataset case study results

	Training Dataset 1	Training Dataset 2	Training Dataset 3
LR with Local vectors	Initial F1: 0.58 Max F1: 0.74 (Iteration: 2)	Initial F1: 0.58 Max F1: 0.7 (Iteration: 2)	Initial F1: 0.71 Max F1: 0.77 (Iteration: 2)
LR with Google vectors	Initial F1: 0.72 Max F1: 0.96 (Iteration: 14)	**Initial F1: 0.81** **Max F1: 0.93** **(Iteration: 2)**	**Initial F1: 0.69** **Max F1: 0.94** **(Iteration: 8)**
CNN with Google vectors	**Initial F1: 0.41** **Max F1: 0.97** **(Iteration: 1)**	Initial F1: 0.55 Max F1: 0.79 (Iteration: 10)	**Initial F1: 0.94** **Max F1: 0.94**

Severe Weather case study observations

- LR and CNN with Google vectors have the best results in 2 of 3 sets, with a tie in set 3. This is somewhat consistent with our previous experiment, since the LR and CNN accuracy differs by only 0.01 in the first set.
- The Max F1 obtained in the previous experiment was 1.0, whereas we have a lower F1 across all of the 3 different datasets.

Table 33: Test of Significance – Initial vs. Max F1 score comparison between the 3 datasets

	p-value	Interpretation
LR with Local vectors	0.03	***Statistically significant***
LR with Google vectors	0.02	***Statistically significant***
CNN with Google vectors	0.12	Not statistically significant

Two of the three p-values that we obtain by comparing the initial and best F1 scores for each classifier as shown in Table 33 are not significant. They correlate somewhat with our case study results that we observe in Table 32 as there was considerable improvement using the augmented training methodology for the LR-Local and LR-Google classifiers. Also, there was a considerable improvement observed with the CNN classifier in two of the datasets, but the differences are observed as not statistically significant. This anomaly can be attributed to the small sample size and the fact that the initial F1 score in training dataset 3 did not improve at all. The p-value becomes very sensitive to even one instance of an outlier because of the small sample size.

6.4.2.5 Winter Storm

Table 34: Winter Storm dataset case study results

	Training Dataset 1	Training Dataset 2	Training Dataset 3
LR with Local vectors	**Initial F1: 0.7** **Max F1: 0.88** **(Iteration: 9)**	Initial F1: 0.64 Max F1: 0.81 (Iteration: 5)	Initial F1: 0.88 Max F1: 0.93 (Iteration: 6)
LR with Google vectors	Initial F1: 0.66 Max F1: 0.82 (Iteration: 8)	**Initial F1: 0.85** **Max F1: 0.89** **(Iteration: 8)**	Initial F1: 0.84 Max F1: 0.91 (Iteration: 5)
CNN with Google vectors	Initial F1: 0.40 Max F1: 0.85 (Iteration: 10)	Initial F1: 0.44 Max F1: 0.88 (Iteration: 3)	**Initial F1: 0.98** **Max F1: 0.98**

Winter Storm case study observations

- There is no clear winner here. It is surprising to see LR with Local vectors outperform other classifiers based on the Google Word2Vec vectors. This could be attributed to the collection being very generic, and the classifier cannot classify based on a few distinguishing words.
- The Max F1 obtained in the previous experiment was 0.94, whereas we have a slightly lower F1 across all of the 3 different datasets.

Table 35: Test of Significance – Initial vs. Max F1 score comparison between the 3 datasets

	p-value	Interpretation
LR with Local vectors	0.04	***Statistically significant***
LR with Google vectors	0.06	Not quite statistically significant
CNN with Google vectors	0.09	Not statistically significant

Only one of the p-values that we obtain by comparing the initial and best F1 scores for each classifier as shown in Table 35 is significant. They don't correlate well with our case study results that we observe in Table 34 as there is considerable improvement using the augmented training

methodology for the CNN classifier in two of the three datasets. This anomaly can be attributed to the small sample size and the fact that the initial F1 score in training dataset 3 did not improve at all. The p-value becomes very sensitive to even one instance of an outlier because of the small sample size. For the LR-Google classifier, there were considerable improvements in two of the 3 datasets and the difference was very close to 0.05.

6.4.3 Test of Significance – Summarization of Results

Many of the anomalies observed in significance tests in the sections 6.4.2.16.4.2.5 happen because of small sample size and the same was evident in our case studies too. To empirically validate with a larger sample size, we select the LR-Local classification results from the 5 datasets in Section 6.2.2 and combine them with the results of the 3 datasets in Section 6.4.2. This gives us a sample size of 8 observations and the p-value obtained indicates statistical significance as shown in Table 36. This suggests that we need to have more observations to conclusively assert the statistical significance for our results. The small sample sizes make the tests of significance very sensitive to a few outliers. Also, even a slight overlap between the distributions can result in a p-value of more than 0.05 and hence indicating the difference in the distributions being not statistically significant.

Table 36: Test of Significance with 8 observations

Initial vs. Best	
LR-Local	
0.58	0.94
0.76	0.78
0.9	0.93
0.86	0.9
0.6	0.83
0.42	0.84
0.86	0.9
0.82	0.86
p-value	**0.04**

6.5 Classifying Using the Large Collection Learning Optimizer

As described in Section 3.6, we develop a large collection learning optimizer framework that can generate and choose the best classifier for the user and label a given collection. We choose the "Greece" and "Environment" datasets from our collections for this purpose. The "Greece" dataset contains tweets related to the financial crisis in Greece and the "Environment" dataset is more general with tweets talking about environment and the various aspects related to it.

6.5.1 Setup

Table 37 shows the distribution of the training, validation, and auxiliary datasets for the Greece and Environment categories.

Table 37: Greece and Environment datasets

	Greece	Environment
Training	10 (10+, 7-)	20 (10+, 10-)
Validation	57 (18+, 39-)	42 (19+, 23-)
Auxiliary	600	691

6.5.2 Results

Table 38: Greece and Environment dataset classification results

	Greece	Environment
LR with Local vectors	Initial F1: 0.42 Max F1: 0.46 (Iteration: 19)	Initial F1: 0.58 Max F1: 0.67 (Iteration: 1)
LR with Google vectors	Initial F1: 0.68 Max F1: 0.77 (Iteration: 20)	**Initial F1: 0.78** **Max F1: 0.83** **(Iteration: 8)**
CNN with Google vectors	**Initial F1: 0.65** **Max F1: 0.83** **(Iteration: 7)**	Initial F1: 0.66 Max F1: 0.66

Table 38 outlines the results of the classification based on the learning optimizer. The learning optimizer chose the CNN with Google word vectors and LR with the Google word vectors for the Greece and the Environment datasets, respectively. The choice of feature vectors and classification method was based on the classification F1 score obtained during the execution of the learning optimizer.

We observe a substantial gain in the F1 score for the Greece dataset but the gain achieved on the Environment dataset is not that substantial. This could be attributed to the fact that the Environment category is very general and a classifier cannot rely on a few keywords to distinguish the positive examples from the negative ones. For the Greece dataset, the classifier relies on keywords like bailout, crisis, finance, etc., to assist it with classification, and is able to improve upon the initial classification results.

7 Conclusion

We performed 2 case studies consisting of Preliminary and Augmented Training experiments. The goal of the Preliminary case studies was to identify what feature representation and classification methods would work the best for classifying tweets. The Augmented Training case studies focused on identifying how to build a classifier using very little training data and augment it with training examples from an auxiliary source. We further developed a framework to build an optimized classifier for an unseen collection and classify the collection using the optimized classifier.

In Sections 7.1 and 7.2, we discuss the results of the Preliminary and Augmented Training case studies and validate the hypotheses outlined in Section 1.4.

7.1 Preliminary Case Studies

7.1.1 Pre-Processing of Tweets

We hypothesized that pre-processing the text contained in the tweets can help improve classification accuracy. The pre-processing step gave us improved results with both the Association Rules and Logistic Regression based classifiers. This proved that cleaning the text after understanding the domain (Twitter) does result in better F1-scores for both the classifiers; we conclude this hypothesis is valid.

7.1.2 Word2Vec Feature Representation and Transformation

We hypothesized that a rich feature representation like Word2Vec can result in higher accuracy in classification of tweets than methods based on Association Rules. The choice of the Word2Vec feature selection and transformation technique gave us good results in terms of classifier accuracy performance. We believe this is because of its ability to capture the semantic and syntactic relations of the words.

The classifier that we have developed based on the Word2Vec and Logistic Regression gives us very good accuracy (0.96 F1-score) when applied to a sample of 9 classes. This result also performs better than the AR classifier (0.90 F1-score) and reduces the error rate by 60%. Primarily, this is attributed to the various steps in our classification pipeline including pre-processing, feature transformation, and classification. We conclude that this hypothesis is valid.

7.1.3 Spark Partitioning and Caching

We hypothesized that utilizing a distributed computing infrastructure using Apache Spark can result in increase in run-time performance of the Logistic Regression based classifier. Using the partitioning and caching operations provided by the Apache Spark library gave us significant run-time performance improvements. The Spark/Hadoop development platform lets us monitor the performance metrics in great detail. The platform documentation [1] on task optimization enabled us to identify the various performance bottlenecks visually and design the corresponding optimization strategy accordingly. We conclude that this hypothesis is valid.

7.2 Augmented Training Case Studies

7.2.1 Augmented Training

We hypothesized that higher accuracy can be achieved by augmenting a base classifier trained on a small amount of training data with additional examples to learn from. As part of the case studies with the 5 datasets, we observed substantial increase in classifier accuracy with augmented training data for the Logistic Regression based classifiers that we developed.

However, the CNN based classifier does not register much improvement with augmented training data. For 3 of the 5 sets it does not register any improvement. This could be attributed to the fact that it is learning the most distinguishing words from the initial training data itself. It only showed improvement with the "WinterStorm" dataset where its F1 score improved from an initial 0.84 to the maximum of 0.99. Any dataset that does not lend itself to be classified with a few distinguishing keywords does not let CNN attain a high accuracy on the initial dataset. This is confirmed by our final learning optimizer based experiment where CNN could not improve at all on the Environment dataset and it took 7 iterations of augmented training on the Greece dataset for it to reach its best F1 score of 0.83.

We conclude that this hypothesis is valid for the Logistic Regression based classifiers, but not for the CNN based classifier. We also conclude that, based on the experimental results in Section 6.4, the Logistic Regression based classifiers show consistent improvement by using the Augmented Training methodology, and the improvement is not just by chance.

7.2.2 Labeling the Auxiliary Data Samples

We hypothesized that Word2Vec based feature representation can also help identify what tweets in the auxiliary data source are similar and dissimilar to the positively labelled training data. The training examples that were labelled using the cosine similarity measure on the feature vectors did improve the accuracy of the classifier. This is attributed to the fact the feature representation is very rich and tweet text projected in higher dimension are conducive for the cosine similarity operation to distinguish the positive examples from the negative ones. We conclude that this hypothesis is valid.

7.2.3 Word Vector Source Corpus

We hypothesized that Word2Vec based feature vectors for tweets trained from the auxiliary data source can help achieve better classification accuracy than the word vectors generated from the Google News corpus since these vectors are trained on words from the same domain that we are classifying.

The preliminary case studies used Word2Vec word vectors generated from the auxiliary data source. The augmented training based on these word vectors showed accuracy gains using the Logistic Regression classifier, but did not work with the CNN based classifier. In fact, it performed poorly on all datasets with the CNN based classifier, and labelled all examples as either belonging or not belonging to the class.

The Logistic Regression classifier using word vectors generated from the auxiliary data source performs poorly for the Greece and Environment datasets even with augmented training data. The datasets are very generic in nature and cannot be classified based on a few distinguishing keywords. This poor performance can be attributed to the local word vectors built using very limited vocabulary, and words that are not in the vocabulary of the Word2Vec model being ignored during classification. In all our experimental results shown in Section 6.2.2, we observed that the

classifier trained on the word vectors generated from the auxiliary data source never achieved a better performance than classifiers trained on the word vectors generated from the Google News corpus. We therefore conclude that this hypothesis is false.

7.2.4 CNN Classifier as the Universal Best Choice

We hypothesized that the CNN based classifier would consistently outperform simple classifiers like Logistic Regression, Naïve Bayes, etc. Based on the experimental results shown in Section 6.2.2, the CNN based classifier outperformed the Logistic Regression based classifier in 4 of the 5 datasets. Though the gains are not statistically significant. In the one instance where the Logistic Regression based classifier had better performance than CNN, the difference was quite stark (0.98 vs. 0.89).

For the final experiment with the "Greece" and "Environment" datasets, the CNN based classifier had the best performance on the "Greece" dataset while the Logistic Regression based classifier had the best performance on the "Environment" dataset. The "Environment" dataset is very general in nature and cannot be classified by using just the presence of a few distinguishing keywords and hence CNN does not get the best performance on this.

Based on our case studies, we are not able to conclusively assert that a single classification methodology would perform the best for most of the real-world events. We therefore conclude that this hypothesis is false.

We even wonder whether it is possible to make such an assertion given the challenges associated with language and text classification. We plan to address this in our future work.

7.2.5 N-gram based Feature Representation for CNN

We hypothesized that a CNN classifier constructed using a bi-gram or tri-gram feature representation would perform better than a unigram representation. Based on the results of our experiments in Section 6.3, we see no strong evidence in favor of either of the bi-gram or tri-gram feature representations. Based on our experimental results, we conclude that the hypothesis is invalid.

We believe that combining the feature representation similar to [30] might help us make a strong assertion on this. We plan to address this in our future work.

7.3 Research Questions

In Section 1.5, we posed several research questions. Several of those have been answered in the preceding two sections. In this section, we attempt to answer the remaining one based on what we have observed in our case study results.

7.3.1 Performance of the Classifier on a Completely Unseen Collection

As we observe in Section 6.5.2, the Greece and Environment datasets were classified using the Large Collection Learning Optimizer. The framework does identify the best classifier to be used for each collection. The framework uses the generated classifier model to classify the whole collection.

For the system administrators that are managing digital library systems similar to Virginia Tech's DLRL, the Large Collection Learning Optimizer is instrumental in reducing the human effort to classify the collections.

7.4 Research Contributions

In Section 1.6, we outlined the research contributions of our work. In the following discussion, we revisit the contributions and summarize them.

- This work identified and highlighted the intricacies involved in classification of tweets, and defined the rules related to feature engineering, that included feature selection and transformation along with the selection of the classifiers.
- This work provided an approach to run the classification of tweets using Logistic Regression and Word2Vec on the Apache Spark platform using the full features of the platform, including but not limited to distributed computing and fast execution. This work involved researching the performance implications of Word2Vec based Logistic Regression classifier and utilized the features of the Apache Spark platform to optimize the runtime performance of the classification pipeline. The optimization based on caching and repartitioning of the data across the cluster did result in increased runtime performance of the Logistic Regression classifier.
- This work described an approach on how to extract additional training from an auxiliary data based on a very small initial training set. It also evaluated the efficacy of using Google News and the auxiliary source data based word vectors, and compares the results generated by the various classifiers using these vector representations.
- This worked involved constructing a CNN based classifier and measuring its effectiveness using unigram, bi-gram and tri-gram word context representations. We have laid out the foundation for pursuing this extensively in the future and come up with more concrete guidance on what representations would work the best for what kind of real world events or categories.
- We have devised a framework that would let us build a classifier with very little training data, based on the methodology described in Section 3.6. Most of the steps of this process are automated. This will be very useful for the Digital Library community as this would let them focus on more important things than manually building a classifier from scratch each time they want a new event or trend to monitor and classify.

8 Future Work

To fully explore the problem of classifying tweets as well as the results of our experiments we have identified the following future work:

1. The GETAR project scope includes identifying events and trends from webpages and tweets. This work was based on classifying tweets. It would be interesting to modify the techniques used, to classify webpages, and to tune it to achieve a satisfactory classification accuracy for webpages.
2. We have used the cosine similarity measure to label the examples from the auxiliary dataset. [28] proposes a methodology based on the edit distance of the words, termed as Word Mover Distance (WMD). It uses the cost function based on the vector distance based on Word2Vec vectors between the words that are needed to be substituted, replaced, or added, to transform one sentence to the other. This technique is hyper-parameter free and is highly interpretable, as the distance between two tweets can be explained as the edit distances between a few individual words. This method has shown to outperform 7 state-of-the-art alternative document distance techniques. Using a WMD based threshold and window size to label the positive and negative examples in auxiliary data has the potential to find better examples for augmenting the training process.
3. As part of our case studies, we observed that the Word2Vec vectors based on the auxiliary data performed better than CNN in a few cases. We propose building local word vectors based on the Word2Vec technique from a collection of about 2000 tweets randomly sampled from all the collections in the IDEAL database. This would ensure that we have built a substantial vocabulary for the word vector model. Having a Word2Vec model trained on just tweets might be able to represent the words more effectively in the vector space based on the Twitter orthography, including non-standard English words present in the Twitter universe. Performing classification using the feature representation based on these word vectors can potentially improve the results with either of the Logistic Regression or CNN based classification methods. If results are promising, samples larger than 2000 could be utilized.
4. Work in [30] performed an extensive sensitivity analysis on the different parameters that are used to build the CNN classifier for text classification. This involves using word vector representations other than Word2Vec, filters with different region sizes, using different activation functions, and more. Performing an extensive sensitivity analysis using tweets and finding the optimal values for these parameters can potentially result in better classification efficacy for a classifier based on CNN.
5. Work in [13] involves techniques that combine a CNN and a Recurrent Neural Network (RNN) to form a Recurrent CNN. It utilizes a bi-directional recurrent structure to capture word features that contain the contextual information to the greatest extent possible when learning representation of text sentences. The CNN max pooling layer allows identifying which features play a key role, i.e., what words are the most distinguishing for a given class. Performing classification on the tweets based on this work can help achieve better classification results.

6. We posed a question in the discussions section whether we can make an assertion as to the possibility of a single classification methodology that would perform the best for most real-world events. To conclusively answer this question, we propose choosing about 20 categories of different types and performing case studies on them using the learning optimizer framework. Performing a detailed analysis on the results obtained would allow us to answer this question.

9 References

[1] "Apache Spark is a fast and general engine for large-scale data processing," October 2016. [Online]. Available: http://spark.apache.org/.

[2] Christian S. Perone, "Cosine Similarity for Vector Space Models," http://blog.christianperone.com/2013/09/machine-learning-cosine-similarity-for-vector-space-models-part-iii/.

[3] C. D. Manning, P. Raghavan and H. Schütze, An Introduction to Information Retrieval, vol. 1, Cambridge: Cambridge University Press, 2008.

[4] P. Meesad, P. Boonrawd and V. Nuipian, "A chi-square-test for word importance differentiation in text classification," *Proceedings of International Conference on Information and Electronics Engineering,* pp. 110-114, 2011.

[5] S. R. Singh, H. A. Murthy and T. A. Gonsalves, "Feature Selection for Text Classification Based on Gini Coefficient of Inequality," *FSDM,* vol. 10, pp. 76-85, 2010.

[6] Y. Bengio, R. Ducharme, P. Vincent and C. Janvin, "A neural probabilistic language model," *The Journal of Machine Learning Research,* vol. 3, pp. 1137-1155, 2003.

[7] T. Mikolov, K. Chen, G. Corrado and J. Dean, "Efficient Estimation of Word Representations in Vector Space," *eprint arXiv:1301.3781,* 2013.

[8] P. Jin, Y. Zhang, X. Chen and Y. Xia, "Bag-of-Embeddings for Text Classification," *International Joint Conference on Artificial Intelligence,* no. 25, pp. 2824-2830, 2016.

[9] S. K. Pal and S. Mitra, "Multilayer perceptron, fuzzy sets, and classification," *IEEE Transactions on Neural Networks,* vol. 3, pp. 683-697, 1992.

[10] D. A. Pereira, E. E. Silva and A. A. Esmin, "Disambiguating publication venue titles using association rules.," *Proceedings of the 14th ACM/IEEE-CS Joint Conference on Digital Libraries,* pp. 77-85, 2014.

[11] E. P. S. Castro, S. Chakravarty, E. Williamson, D. A. Pereira and E. A. Fox, "Classifying Short Unstructured Data using the Apache Spark Platform," *Joint Conference in Digital Libraries,* 2017.

[12] Y. Kim, "Convolutional neural networks for sentence classification," arXiv preprint arXiv:1408.5882, 2014.

[13] L. Siwei, L. Xu, K. Liu and J. Zhao, "Recurrent Convolutional Neural Networks for Text Classification," in *AAAI*, 2015.

[14] "VTech Works," 1 12 2016. [Online]. Available: https://vtechworks.lib.vt.edu/handle/10919/19081. [Accessed 1 12 2016].

[15] L. Li, A. Pillai, Y. Wang and K. Tian, "CS5604 Fall 2016 Solr Team Project Report," [Online]. Available: https://vtechworks.lib.vt.edu/handle/10919/73710. [Accessed 06 05 2017].

[16] M. J. Wagner, F. Abidi and S. Fan, "CS5604: Information and Storage Retrieval Fall 2016 - CMT," [Online]. Available: https://vtechworks.lib.vt.edu/handle/10919/73739.

[17] T. Dao, C. Wakeley and L. Weigang, "Collection Management Webpages - Fall 2016," [Online]. Available: https://vtechworks.lib.vt.edu/handle/10919/76675.

[18] E. R. Williamson and S. Chakravarty. [Online]. Available: https://vtechworks.lib.vt.edu/handle/10919/73713.

[19] A. Bartolome, M. Islam and S. Vundekode, "Clustering and Topic Analysis in CS 5604," [Online]. Available: https://vtechworks.lib.vt.edu/handle/10919/73712.

[20] R. Kohler, R. Tasooji and P. Sullivan, "CS 5604 INFORMATION STORAGE AND RETRIEVAL Front-End Team Fall 2016 Final Report," [Online]. Available: https://vtechworks.lib.vt.edu/handle/10919/73711.

[21] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, B. S. J. and D. McClosky, "The Stanford CoreNLP Natural Language Processing Toolkit," *Association for Computational Linguistics: System Demonstrations,* pp. 55-60, 2014.

[22] Frontline Solvers, "Association Rules," 2017. [Online]. Available: http://www.solver.com/xlminer/help/association-rules. [Accessed 24 04 2017].

[23] V. Rijsbergen and C. Joost, Information retrieval, 1979.

[24] C. McCormick, "Word2Vec Tutorial," 19 April 2016. [Online]. Available: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/ .

[25] Z. Chase Lipton, C. Elkan and B. Narayanaswamy, "Thresholding Classifiers to Maximize F1 Score," *eprint arXiv:1402.1892,* 2014.

[26] Z. Lu, Z. Yin, S. J. Pan, E. W. Xiang, Y. Wang and Q. Yang., "Source Free Transfer Learning for Text Classification," *AAAI,* pp. 122-128., 2014.

[27] S. Chakravarty, E. R. Williamson and E. A. Fox, "Classification of Tweets using Augmented Training," *WADL,* 2017.

[28] M. J. Kusner, Y. Sun, I. N. Kolkin and K. Weinberger, "From Word Embeddings To Document Distances," ICML, 2015.

[29] Stanford University, "Convolutional Neural Networks for Visual Recognition," [Online]. Available: http://cs231n.github.io/neural-networks-1/. [Accessed 23 04 2017].

[30] Y. Zhang and B. Wallace, "A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification.," *arXiv preprint 1510.03820,* 2015.

[31] J. Brownlee, "How To Choose The Right Test Options When Evaluating Machine Learning Algorithms," http://machinelearningmastery.com/how-to-choose-the-right-test-options-when-evaluating-machine-learning-algorithms/, 2014, February 19.

[32] Google Inc., "Word2Vec Implementation," Google, 29 July 2013. [Online]. Available: https://code.google.com/archive/p/word2vec/. [Accessed 18 April 2017].

[33] JetBrains, "Intellij IDEA," [Online]. Available: https://www.jetbrains.com/idea/. [Accessed 11 October 2016].

[34] "What is Apache Hadoop?," 2016. [Online]. Available: http://hadoop.apache.org/.

[35] "Welcome to Apache HBase," 09 October 2016. [Online]. Available: http://hbase.apache.org/.

[36] "APACHE HADOOP HDFS," 2016. [Online]. Available: http://hortonworks.com/apache/hdfs/.

[37] "MLlib is Apache Spark's scalable machine learning library," 2016. [Online]. Available: http://spark.apache.org/mllib/.

[38] R. Jindal, R. Malhotra and A. Jain, "Techniques for text classification: Literature review and current trends," *webology,* vol. 12, no. 2, pp. 1-28, 2015.

[39] Emit Classifier Probability in Spark for Logistic Regression, http://stackoverflow.com/questions/30391399/predicting-probablities-in-logistic-regression-model-in-apache-spark-mlib/36238801#36238801.

[40] American Press Institute, "Twitter and the News: How people use the social network to learn about the world," 1 September 2015. [Online]. [Accessed 2016].

[41] J. Han, J. Pei and Y. Yin, "Mining frequent patterns without candidate generation," *ACM SIGMOD international conference on Management of data,* pp. 1-12, 2000.

[42] D. D. Lewis, "Reuters-21578 text categorization test collection, distribution 1.0," 1997. [Online]. Available: http://www. research. att. com/~ lewis/reuters21578. html.

[43] C. Sherman, "Humans Do It Better: Inside the Open Directory Project," July 2000. [Online]. Available: http://www.infotoday.com/online/OL2000/sherman7.html. [Accessed 2016].

[44] M. Oakes, R. Gaaizauskas, F. H, A. Jonsson, V. Wan and M. Beaulieu, "A method based on the chi-square test for document classification," *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval,* pp. 440-441, September 2001.

[45] N. Japkowicz, "Assessment Metrics for Imbalanced Learning," in *Assessment Metrics for Imbalanced Learning, in Imbalanced Learning: Foundations, Algorithms, and Applications (eds H. He and Y. Ma)*, John Wiley & Sons, Inc. Hoboken, NJ, USA.

Appendix

Appendix A: Implementation

In this appendix, we describe the implementation details of our work.

A1. Environment

To rapidly develop the code needed for the project we opted for a hybrid environment between the DLRL cluster and our local machines. Fast iterative development was done on our local machines where we could take advantage of tools such as the IntelliJ IDEA [33] to quickly develop our Scala code. We could verify that the code is working on small datasets on our local machines, and then move the code up to the cluster when we ran classifications on the large datasets. This allowed us to take advantage of the processing power of the cluster when we needed to run our experiments. From the cluster, we were able to communicate with the database to store and retrieve data from the IDEAL and GETAR project collections.

The following technologies and frameworks were used for the implementation.

- **Apache Hadoop** – This is the base layer of the distributed computing framework that we used. [34]
- **Apache Spark** – This is an optimized RDD based framework built on top of Hadoop. [1]
- **HBase** – This is the distributed database from the Apache Hadoop stack that is widely used as a NOSQL database in many implementations. [35]
- **HDFS** – This is the distributed file system of the Apache Hadoop stack. [36]
- **Spark MLlib** – This is a machine learning library [37] that is based on the Apache Spark framework. For all the classification work related to our project, we used this library.

A2. Training Data Curation

To be able to build classifiers we had to create training and test sets to train and evaluate our classifiers. To begin, we took data from the database present in the column family "cleantext" and assumed that that data had been cleaned of profanity and unreadable text. For our initial studies, we took 200 random samples of the documents (tweets) from each of the collections to form the basis for different classes. We then hand labeled the documents with the class that they belong to. As per the description in Section 1.1.3, we label the tweet as to its specific category and not its parent category.

A3. HBase Access

HBase is the database that is being used to store all of the tweets that our classifiers will run on. For our classification to operate satisfactorily we must read the tweets from HBase efficiently and correctly. Spark provides many ways to read records from HBase such as Scan [34] that allows iteration over an HBase table, and HadoopRDD [34] that reads HBase data into an RDD for further processing.

To use the HadoopRDD API we have to first specify the table name from which we want to generate an RDD. The API only supports making an RDD out of a single table; to get multiple tables we will have to create multiple RDDs. The next list of parameters that must be specified are the data type information that each row-key is stored in the HBase table. Finally, we can specify the columns that we want to retrieve from HBase.

The HadoopRDD API works by streaming the data to the driver node, then partitioning it across the cluster so each node can get a different piece of the data. This means that we can execute operations on that data across the cluster. Example code for how a HadoopRDD will be created can be seen in Figure 22.

```
// This will store any configuration for what we read from HBase
val conf = HBaseConfiguration.create()

// configure the name of the HBase table to read
conf.set(TableInputFormat.INPUT_TABLE, "Table Name")

// the RDD to be retrieved from HBase
val hBaseRDD = sc.newAPIHadoopRDD(conf, classOf[TableInputFormat],
  classOf[org.apache.hadoop.hbase.io.ImmutableBytesWritable], // key type
  classOf[org.apache.hadoop.hbase.client.Result]) //result type
```

Figure 22: HadoopRDD usage example

The Scan API works similarly to the HadoopRDD API. It requires the table name to be specified along with the columns to read and any filtering of records that you want to do. The Scan API returns a Result object. This object allows us to iterate over these records in small batches without committing a lot of memory. This operation is not parallelized and would be executed on the driver node, so to achieve parallelism for the later operations we have to take some records and parallelize them into an RDD. An example framework for showing how this is set up can be seen in Figure 23.

```
// a scan object that will specify the columns to scan
val scan = new Scan()

// a reference to the specific HBase table we will be reading from
val table = new HTable(HBaseConfiguration.create(),"table name")

// add the specific column to read from the table
scan.addColumn(Bytes.toBytes("column family"), Bytes.toBytes("column"))

// get an iterator over the records in the table matching the scanner
val resultScanner = table.getScanner(scan)

val BatchSize = 5000;
while (!complete){
    // read one batch into memory to process
    val results = resultScanner.next(BatchSize)
    // process results
}
```

Figure 23: Scan usage example

One challenge that we faced is that there are collections of tweets that have millions of records. Because of this, our read code must be able to still work efficiently when operating on large record counts. We found empirically that the HadoopRDD API did not finish reading after several hours when run on a collection of size greater than 1 million records. This led us to use the Scan API and read in small sections of the large collection at a time. By reading in a small block, parallelizing it across the cluster, then only reading the next one once we had finished processing the previous block, we were able to keep our code running in parallel while eliminating memory errors that large collections generate.

A4. Cleaning

Twitter limits the number of characters in each message. This means that each document we want to classify has a limited amount of information. To be able to correctly classify these tweets we clean the data of non-discriminative stop words, perform lemmatization, and remove symbols such as hashtags '#' and URLs.

RT: @AssociationsNow A Year After Texas Explosion Federal Repourt Outlines Progress on Fertilize... http://t.co/8fDbMu9asU #meetingprofs

Figure 24: Example raw tweet

An example raw tweet (i.e., not yet cleaned) can be seen in Figure 24. This tweet has a short URL that is irrelevant to the class it is a part of. The words in this tweet are also capitalized, and we would like them to still match with words contained in other tweets that are lowercase. It is also important to lemmatize words, so that the features that the classifier will train on, will consist of the word lemmas [38]. An example of the raw tweet from Figure 24 after being cleaned is shown in Figure 25.

year texas explosion federal report outline progress fertilize meetingprof

Figure 25: Example of a cleaned tweet

We utilized these cleaning methods on our training and test data for our experiments. The specific details on how we accomplished each cleaning method can be found in Section 3.3.1. The accuracy gain that the classifiers experienced when they ran on cleaned data can be found in Section 5.1.

A5. Association Rules Classifier

Association rules is a technique for data mining that provides relationships between elements through the form of rules. The rules take the form of implications with a certain confidence value. For use in classification, if the text contains a collection of terms, it will belong in the rules' respective class. If there is no rule that will classify the record, the record will be classified using similarity measures such as cosine similarity to determine the class it is closest to in vector space. Association rules has been shown to be effective at matching publication venue title variations to their actual titles [10].
An example rule would be:

{<KY>, <Bluegrass State>} --> Kentucky

The presence of the token KY, or the phrase "Bluegrass State," implies that this document is talking about the state Kentucky.

Association rules are useful when classifying large collections because the prediction operation can be performed efficiently in constant time when a hash table is used to provide a lookup for the rules. One specific parameter of interest for the Association Rules classifier will be the support threshold that is required to use the rule in prediction. A lower support threshold will allow more predictions to be made by the rules, speeding up the overall prediction time. However, predictions with rules with a lower support will not be as accurate as the ones with high support. The effect of different support thresholds on the number of tweets classified with rules and similarity is explored in Section 5.2.

A6. Classifier Training and Prediction

The definition of classification is to determine which class(es) a given sample belongs to, given a set of pre-defined classes. As for the problem defined for the project, we have to classify a given tweet to a real-world event as explained in Sections 1.1.31.2.

The process of classification involves the following three steps.

1. Feature selection

2. Feature representation
3. Choosing a classification algorithm

We started our first implementation with Logistic Regression and considering that we have a lot of training data at our disposal, the choice of classifier [3] in our case is not significant. After performing some initial experiments with a small set of test data, the accuracy results were quite promising. More details related to this experiment can be found in Section 5.3.2. Thus, we continued with Logistic Regression as our classification algorithm of choice. We use a multi-class classifier for our project for the preliminary case studies and a binary classifier thereafter, including for the large scale learning optimizer case studies.

Figure 26 shows an example on how to use a Word2Vec model to transform a tweet into a feature vector and train a Logistic Regression classifier and save it into HDFS.

```
1  //Initialize the variables
2  val numberOfClasses = 9
3  val trainingDataFile = "data/trainingData.txt"
4  val word2VecModelFilename = "data/word2VecModel.model"
5  val lrclassifierModelFilename = "data/lrClassifierModel.model"
6
7  //Load the training data into an RDD
8  val trainTweetsRDD = sc.textfile(trainingDataFile)
9
10 //Load the word2vecModel object from HDFS
11 val word2vecModel = Word2VecModel.load(sc,word2VecModelFilename)
12
13 //Define a delegate function to perform the transform using the word2vecModel
14 def transformTweetToWordVector(words:Iterable[String]):Iterable[Vector] =
15 words.map(w=>Try(word2vecModel.transform(w)))
16 .filter(_.isSuccess)
17 .map(x=>x.get.sum)
18
19 //Apply the transform to all the tweets in the training data
20 val trainTweetsFeaturesRDD = trainTweetsRDD mapValues transformTweetToWordVector
21
22 //Train a model using the transformed RDD of feature vector
23 val logisticRegressionTrainedModel = new LogisticRegressionWithLBFGS()
24 .setNumClasses(numberOfClasses)
25 .run(trainTweetsRDD )
26
27 //Save the model to HDFS
28 logisticRegressionTrainedModel .save(sc,lrclassifierModelFilename)
```

Figure 26: Logistic Regression training example

Figure 27 shows an example on how to load a logistic regression model from HDFS and generate predictions on data.

```
1  //Initialize the variables
2  val numberOfClasses = 9
3  val testDataFile = "data/testData.txt"
4  val word2VecModelFilename = "data/word2VecModel.model"
5  val lrclassifierModelFilename = "data/lrClassifierModel.model"
6
7  //Load the test data into an RDD
8  val testTweetsRDD = sc.textfile(testDataFile)
9
10 //Load the word2vecModel object from HDFS
11 val word2vecModel = Word2VecModel.load(sc,word2VecModelFilename)
12
13 //Define a delegate function to perform the transform using the word2vecModel
14 def transformTweetToWordVector(words:Iterable[String]):Iterable[Vector] =
15 words.map(w=>Try(word2vecModel.transform(w)))
16 .filter(_.isSuccess)
17 .map(x=>x.get.sum)
18
19 //Apply the transform to all the tweets in the data to turn into feature vectors
20 val testTweetsFeaturesRDD = testTweetsRDD mapValues transformTweetToWordVector
21
22 //Load a model using the transformed RDD of feature vector
23 val logisticRegressionTrainedModel = LogisticRegressionModel.load(sc, lrclassifierModelFilename)
24
25 //Generate predictions using the logisticRegressionModel
26 val logisticRegressionPredictions = logisticRegressionModel.predict(testTweetsFeaturesRDD)
27
```

Figure 27: Logistic Regression prediction example

A7. Emitting probability in a multi-class scenario

As described in the previous section, we implemented our solution using a multi-class Logistic Regression classifier. The multi-class implementation for Logistic Regression in the Spark MLLib framework is a one-vs.-all wrapper over the binary Logistic Regression classifier. Also, this implementation only predicts the class of a given tweet. It does not emit the raw probability for each of the classes.

For our classification process to be more effective in classification efficacy and also from the perspective of error analysis, it is imperative that we have a mechanism to generate the probabilities of each of the classes. Having the probability values for each of the classes would help us analyze the classification results in further detail and also to understand what is the relative distribution of probabilities across the classes for a given tweet, since a tweet can belong to multiple classes.

We want to have a system that lets us control how stringent we are about assigning one class to a tweet. If we have the probability of all of the classes, we can set a probability threshold based on the precision we want to have with our classes and balance it with the coverage we want to have for our tweet collections. Having this threshold based mechanism would also allow us to classify a tweet in an "unknown" class, since we are not sure what class it belongs to.

Setting a threshold to a high value would result in high precision but low recall since a lot of classes with low probability will be rather classified as "unknown". Setting a threshold to a low value will get a very good recall or coverage on our collection, but will suffer in precision since a class can belong to multiple categories. If there are multiple classes with the same probability, the choice of class in that case will be the first one with the highest probability.

Since the Spark MLLib does not come with an implementation for generating the probability for a multi-class Logistic Regression, we chose to have a custom implementation by building over a code sample [39] on the web.

Figure 28 shows the example to extend the Logistic Regression method call to start emitting probabilities for a tweet. Especially important to note are line numbers 26 and 28, where the former is the way we were using it in the past and the latter is the new call that we make to the newly implemented *ClassificationUtility* class that emits the probabilities along with the predictions.

```
1  //Initialize the variables
2  val numberOfClasses = 9
3  val testDataFile = "data/testData.txt"
4  val word2VecModelFilename = "data/word2VecModel.model"
5  val lrclassifierModelFilename = "data/lrClassifierModel.model"
6
7  //Load the test data into an RDD
8  val testTweetsRDD = sc.textfile(testDataFile)
9
10 //Load the word2vecModel object from HDFS
11 val word2vecModel = Word2VecModel.load(sc,word2VecModelFilename)
12
13 //Define a delegate function to perform the transform using the word2vecModel
14 def transformTweetToWordVector(words:Iterable[String]):Iterable[Vector] =
15 words.map(w=>Try(word2vecModel.transform(w)))
16 .filter(_.isSuccess)
17 .map(x=>x.get.sum)
18
19 //Apply the transform to all the tweets in the data to turn into feature vectors
20 val testTweetsFeaturesRDD = testTweetsRDD mapValues transformTweetToWordVector
21
22 //Load a model using the transformed RDD of feature vector
23 val logisticRegressionTrainedModel = LogisticRegressionModel.load(sc, lrclassifierModelFilename)
24
25 //Generate predictions using the logisticRegressionModel - OLD WAY
26 val logisticRegressionPredictions = logisticRegressionModel.predict(testTweetsFeaturesRDD)
27 //Generate the predictions along with the probabilities - NEW WAY
28 val (prediction,probabilities) = ClassificationUtility
29 .predictPoint(testTweetsFeaturesRDD, logisticRegressionModel)
```

Figure 28: Spark example to generate probabilities for the tweets along with the predictions

Figure 29 shows the probabilities in the expanded object view for the 9 classes for a given tweet that was classified during our experiments. Note that 5 classes among the nine have the same probability. The empirical analysis and our interpretation for the scenarios is discussed in Section 5.4. The first two values in the screenshot are the true and predicted labels for the tweet, and the last entry is the redacted tweet text.

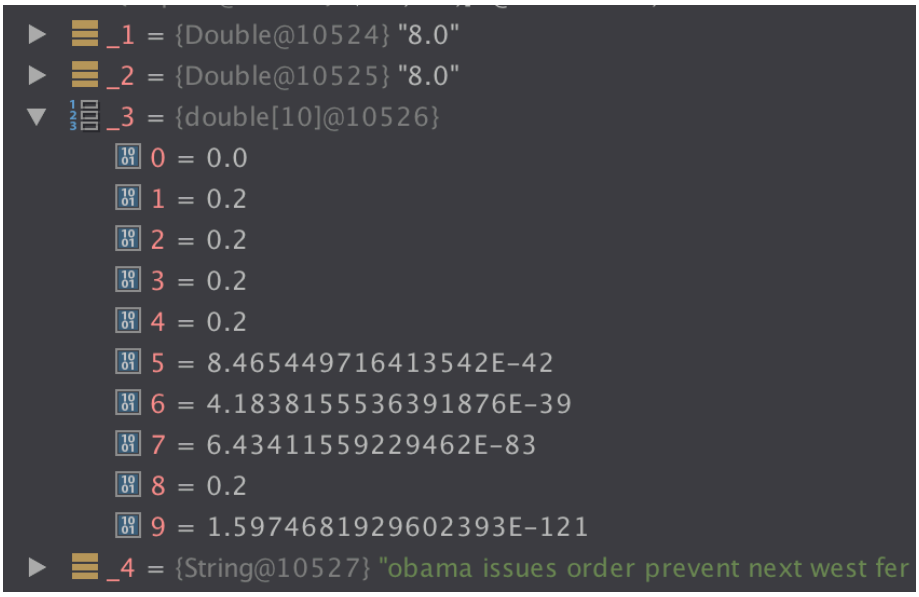

Figure 29: Emission of probabilities along with the prediction for a sample tweet

A8. Spark partitioning and caching

For the tweet collections that we have to classify, we must read and classify tweets ranging in number from a few thousand to tens of millions. It is important for us to ensure fast execution for the classification pipeline and have our implementation optimized to run in a distributed fashion across the cluster. We read data from HBase and partition it accordingly based on our cluster topology. Once the data is distributed across the cluster via partitioning, we perform the classification related processing on the partitioned data. This is described in detail in Section 3.4.1.

An example of the partitioning and caching is shown in Figure 30 and Figure 31, respectively.

```
1
2  //Read logs from a log file
3  val logs = sc.textFile("path/to/log-files")
4
5  //Filter the errors and warnings
6  val errorsAndWarnings = logs filter {l => l.contains("ERROR") || l.contains("WARN")}
7
8  //Repartition the errorsAndWarnings across the cluster
9  errorsAndWarnings.repartition(12)
10
11 //Caching ensures that the following two lines don't invoke the read from the log file twice
12 val errorLogs = errorsAndWarnings filter {l => l.contains("ERROR")}
13
14 //This code is executed in parallel across the cluster on partititioned data
15 //Do something with the error logs
16 val errorCount = errorLogs.count
17
18
```

Figure 30: Spark partitioning example

```
1
2  //Read logs from a log file
3  val logs = sc.textFile("path/to/log-files")
4
5  //Filter the errors and warnings
6  val errorsAndWarnings = logs filter {l => l.contains("ERROR") || l.contains("WARN")}
7
8  // Cache the data so that it is not read from again.
9  errorsAndWarnings.cache()
10
11 //Caching ensures that the following two lines don't invoke the read from the log file twice
12 val errorLogs = errorsAndWarnings filter {l => l.contains("ERROR")}
13 val warningLogs = errorsAndWarnings filter {l => l.contains("WARN")}
14
15 //These two lines are the ones that invoke the collect on the RDD.
16 //Till here the code is just lazily evaluated
17 val errorCount = errorLogs.count
18 val warningCount = warningLogs.count
19
```

Figure 31: Spark caching example

Figure 32 shows the partitioning structure for the Spark RDDs and the Word2Vec and Logistic Regression model, respectively. Figure 33 shows an example of Spark code running across a cluster using partitioning in Spark UI. The blue bars in jobs panel show parallel execution.

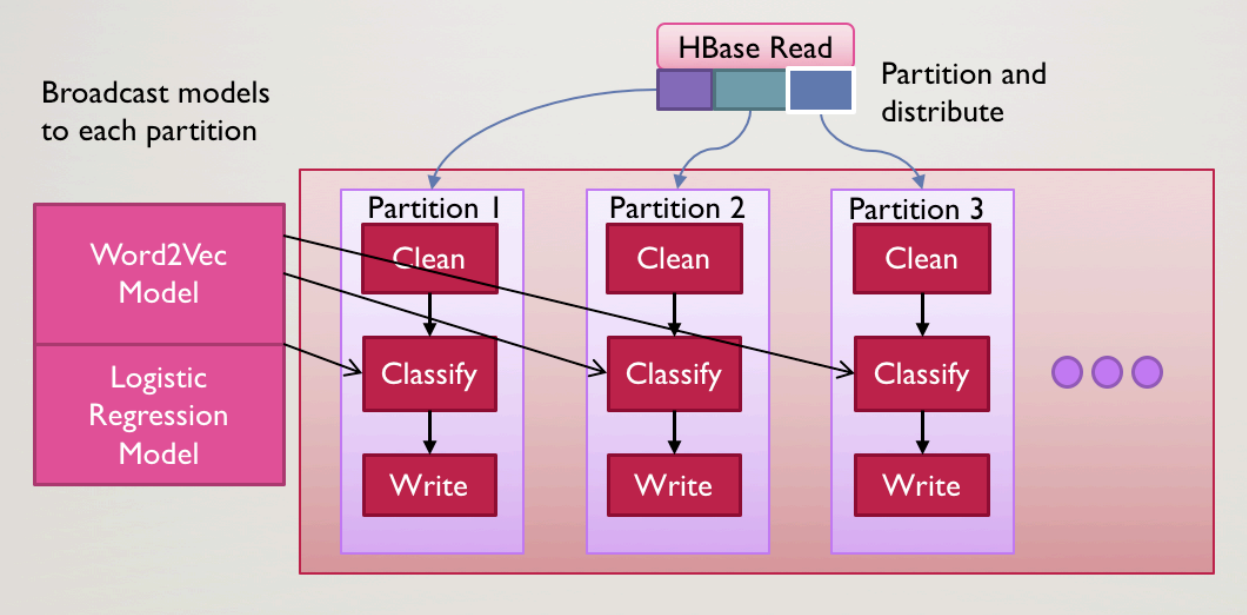

Figure 32: The partitioning structure

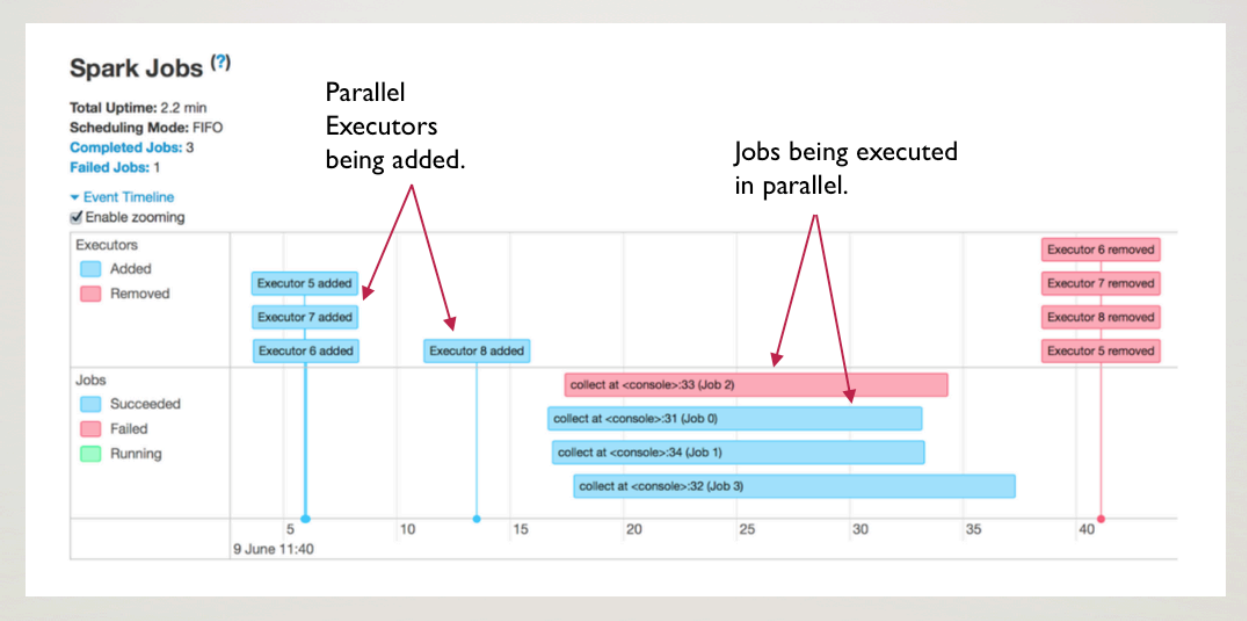

Figure 33: Spark jobs being executed in a parallel fashion

We utilized both repartitioning and caching operations for processing various datasets all through our codebase. The experiments in Section 5.6.2 show the speed gains from the repartitioning and caching based optimization.

A9. Augmenting Training via Auxiliary Training Data

As part of the Augmented Training, we have the following components as part of the execution pipeline.

- **DataRetriever** – This component retrieves the data from a source. For our implementation, retrieve the data from files and HBase.
- **FeatureGenerator** – This component generates the features for a given piece of text. For our implementation, we generate the Word2Vec feature vectors from the Auxiliary and Google News source.
- **Classifier** – This component is used for classification and we have the classifiers for Logistic Regression and Convolutional Neural Network.

For implementing the Augmented Training Framework, we use a factory pattern to instantiate these different components. Having the factory pattern allows us to change the type that gets instantiated. This lets us combine the different DataRetrievers, FeatureGenerators, and Classifiers for our experiments.

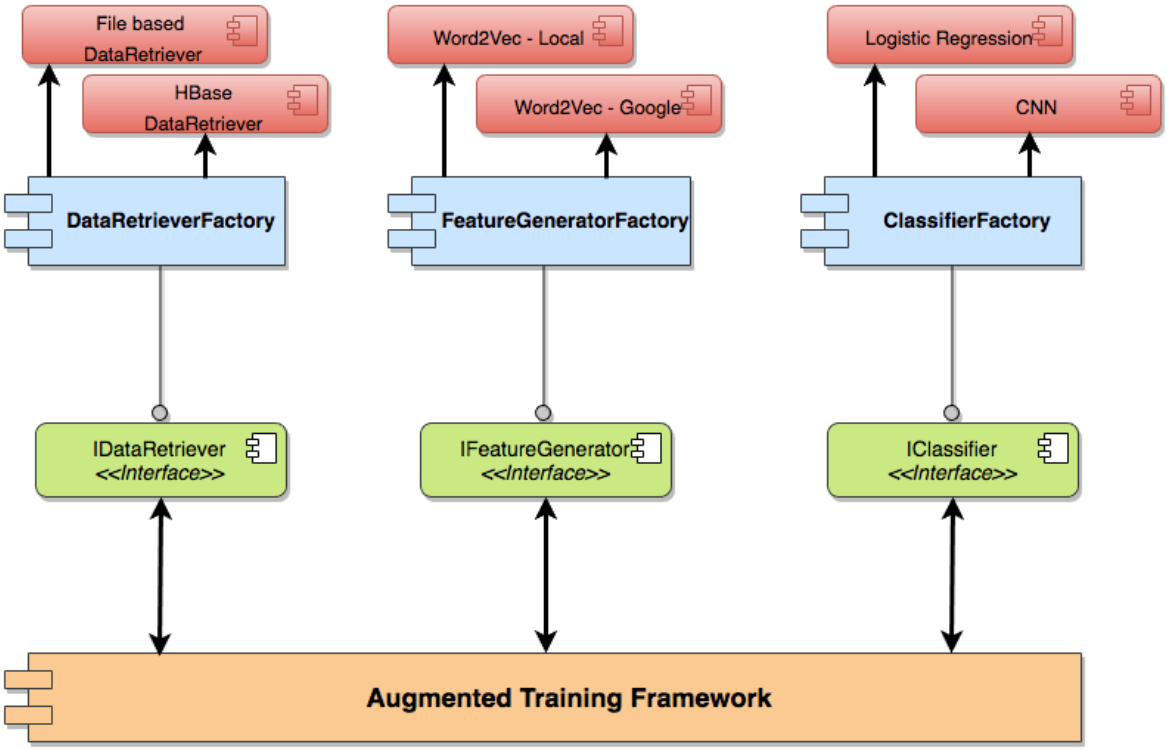

Figure 34: High-level component diagram

Figure 34 shows the high-level component diagram for our implementation. The different factories are connected to the interfaces that they provide an instance for. The factories internally control what type of concrete class instances they provide for the given interface. The Augmented Training framework operates on the interface instances that are provided by the factories.

A10. CNN Implementation

As per the design outlined in Section 3.5, the code snippet shown in Figure 35 is how we constructed the CNN for our work. We used the base implementation of the work in [12].

```
# build the deep neural network model
model = Sequential()
model.add(Convolution1D(nb_filter=self.nb_filters, #Number of filters
                        filter_length=self.n_gram, #Height of the filters
                        border_mode='valid',
                        activation='relu',         #Filter activation function
                        input_shape=(self.maxlen, self.vecsize)))

model.add(MaxPooling1D(pool_length=self.maxlen-self.n_gram)) #Max pooling layer
model.add(Flatten())
model.add(Dense(len(self.classlabels), activation='softmax')) #Output activation

model.compile(loss='categorical_crossentropy', optimizer='rmsprop')
# train the model
model.fit(train_embedvec, indices)

```

Figure 35: Constructing the CNN - Implementation sample

The different components of the CNN classifier are described in Section 3.5. The corresponding implementation reference for the different components in Figure 35 are annotated with a comment.

Appendix B: Implementation Artifacts

B1. Source Repository

Table 39 provides the details of the source code location and repository details.

Table 39: Source Repository details

Source Type	Details	
Large Optimizer Framework	**Location**	*https://github.com/saurabhc123/ThesisPoC*
	Description	This repository contains the code for the Augmented Training and the Large Collection Learning Optimizer framework that leverages Augmented Traning. All the implementation in this repository is in Scala.
Word Vector Generator	**Location**	*https://github.com/saurabhc123/ThesisPython*
	Description	This repository contains the code that generates the word vectors out of the local and Google News based corpus. All the implementation in this repository is in Python.

B2. Training and Validation data

Table 40 provides the details of training and validation dataset that we used for our case studies. The tweets were collected from the DLRL Archive DB. These files can also be retrieved from the repository at *https://github.com/saurabhc123/ThesisPython/tree/master/data.*

Table 40: Training and Validation dataset details

Dataset	Details	
Ebola	Collection Keyword	#ebola, ebola
	Collection Number	224, 225
	Training Data	ebola_training_data.txt ebola_training_data1.txt ebola_training_data2.txt ebola_training_data3.txt
	Validation Data	ebola_validation_data.txt
	Auxiliary Data	ebola_auxiliary_data.txt
Egyptian Revolution	Collection Keyword	#egypt
	Collection Number	1
	Training Data	egypt_training_data.txt egypt_training_data1.txt egypt_training_data2.txt egypt_training_data3.txt
	Validation Data	egypt_validation_data.txt
	Auxiliary Data	egypt_auxiliary_data.txt
Obesity	Collection Keyword	obesity
	Collection Number	377
	Training Data	obesity_training_data.txt

		obesity_training_data1.txt obesity_training_data2.txt obesity_training_data3.txt
	Validation Data	obesity_validation_data.txt
	Auxiliary Data	obesity_auxiliary_data.txt
Severe Weather	Collection Keyword	severe weather
	Collection Number	338
	Training Data	severeweather_training_data.txt severeweather_training_data1.txt severeweather_training_data2.txt severeweather_training_data3.txt
	Validation Data	severeweather_validation_data.txt
	Auxiliary Data	severeweather_auxiliary_data.txt
Winter Storm	Collection Keyword	winterstorm, #winterstorm
	Collection Number	160,161
	Training Data	winterstorm_training_data.txt winterstorm_training_data1.txt winterstorm_training_data2.txt winterstorm_training_data3.txt
	Validation Data	winterstorm_validation_data.txt
	Auxiliary Data	winterstorm_auxiliary_data.txt
Greece	Collection Keyword	greece
	Collection Number	693
	Training Data	greece_training_data.txt
	Validation Data	greece_validation_data.txt
	Auxiliary Data	greece_auxiliary_data.txt
Environment	Collection Keyword	environment
	Collection Number	47
	Training Data	environment_training_data.txt
	Validation Data	environment_validation_data.txt
	Auxiliary Data	environment_auxiliary_data.txt