Classification

Information Storage and Retrieval

CS 5604 - Fall 2017

Final Report

Team Members:
Ahmadreza Azizi
Deepika Kishore Mulchandani
Amit Pandurang Naik
Khai Thoi Ngo
Suraj Satish Patil
Arian Vezvaee
Robin Sheng Yang

Project Advisor:
Prof. Edward A. Fox

1/3/2018

Blacksburg, VA 24061

Abstract

This report describes work to classify webpages and tweets about important global events and trends. Since most of the Twitter data is in text format, classification of these tweets as to events discussed would prove to be useful for finding patterns and relations. Classification of this data would allow users to analyze, visualize, and explore content related to crises, disasters, human rights, inequality, population growth, shootings, violence, etc. To limit the amount of information posted by an individual, Twitter established a maximum length of 140 characters, and then doubled that on 7 November 2017. This resulted in tweets consisting of short forms, slang words, hashtags, and other non-standard use of language. The traditional methods of text classification will not work efficiently on such data. For webpages, since most of the data is in text and it is following the standard usage of the language, the main aim is to extract information about a particular event by selecting the correct features.

Once we received the cleaned data from the CMT and CMW teams in CS5604, we used that data to perform machine learning techniques for classification. During this process, we faced some challenges like hand-labeling the data so that we had sufficient training data. We also faced some challenges while selecting the models for webpage data due to the size of webpages. We tested different models on the hand-labeled webpage data. We were able to select a model with good performance metric scores and efficiency. We integrated this model into our framework and then had a tweet and webpage classification framework ready.

We learned the importance of feature selection before building the classification model. We trained a Word2Vec model on the entire corpus available in the HBase table. In the future, Google's or Stanford's pre-trained Word2Vec models could be compared, possibly making the feature selection process more robust.

We have implemented binary classification, which can be used in the future to support hierarchical classification. This is more flexible than the multi-class classification model from the 2016 team. It allows more models to be trained and added efficiently into our framework, as more global events occur.

Contents

1 Introduction **7**

2 Literature Review **9**

3 Requirements, Design And Implementation **14**
3.1 Requirements 14
3.2 Design 15
3.2.1 Classifying Webpages 15
3.2.2 Tuning Tweet Classification Model 15
3.2.3 Original Framework 16
3.2.4 New Framework 18

4 Implementation **23**
4.1 Hand Labeling Process 23
4.1.1 Tweets 23
4.1.2 Webpages 24
4.2 Class Cluster Classification 25
4.2.1 General Improvements 25
4.2.2 Word2Vec Model Training Data 25
4.2.3 Logistic Regression Model Training 26
4.2.4 Predicting Document Classes 27

5 Experiments **30**
5.1 Classifying Webpages with Tweet Framework 30
5.2 Model Selection for Webpage Classification 32
5.3 Initial Classification of Tweets on Class Cluster 37
5.4 Classification of Documents on getar-cs5604f17 40
5.5 Classifier Accuracy 43
5.6 Classifier Examples 44

6 User Manual **46**

6.1 Dependencies 46
6.2 Project File Structure 47
6.3 Data Files 48
6.4 Calling the Code 49

7 Developer Manual **51**
7.1 File Naming Convention 51
7.2 Source Code Files 52
7.3 HBase Table Scanning 53
7.4 Hand Labeling Tweet Training data 53

8 Timeline **56**

9 Conclusion **58**

10 Future Work **59**
10.1 Hierarchical Framework 59
10.2 Cron task and YAML Configuration File 60
10.3 Hand Labeling Process 61
10.4 Overriding Spark Library Word2Vec 61
10.5 More Cleaned Text for Tweet Documents 62
10.6 Webpage Classifier Performance 62

11 Acknowledgements **63**

Bibliography **64**

List of Figures

1.1 Schematic role of classification team 7

3.1 Training and predicting processes of the previous framework [1] . . . 17
3.2 New training process of the Word2Vec model 18
3.3 New training process of the tweet Logistic Regression model 19
3.4 New training process of the webpage Logistic Regression model . . . 20
3.5 New class prediction process for tweets 21
3.6 New class prediction process for webpages 21

4.1 Example of the hand labeling process after running the script 24
4.2 Data flow of a RDD of a batch of documents during prediction process 28

5.1 Results for Word2Vec with Logistic Regression for initial webpage classification for 2009SchoolShootingDunbar and North Illinois University Shooting 31
5.2 Example of common substrings in documents; each line is one document 32
5.3 2010SchoolShootingUniversityAlabama collection. The webpages belonging to the event and not belonging to the event are 71 and 180, respectively. The quality and number of instances were enough to conclude that the accuracies of different combinations are comparable. Word2Vec with SVM and Logistic Regression have better performance. 33
5.4 2008SchoolShootingNIU collection. The webpages belonging to the event and not belonging to the event are 1 and 14, respectively. The number of instances for each class is not sufficient, and hence the results are not reliable. 34
5.5 2014SchoolShootingReynoldsHighSchool collection. The webpages belonging to the event and not belonging to the event are 34 and 192, respectively. The instances were insufficient and were of poor quality. Accordingly, we believe the results are not comparable. 34

5.6 2012SchoolShootingSandyHook collection. The webpages belonging to the event and not belonging to the event are 31 and 303, respectively. For this collection we believe that even though the instances are few, they are good enough such that we can conclude Word2Vec with Logistic Regression has better performance. 34
5.7 2015SchoolShootingUmpqua collection. The webpages belonging to the event and not belonging to the event are 9 and 408, respectively. The numbers of instances are very unbalanced and their quality is not good enough either. Thus no conclusion about the performance can be made. 35
5.8 2009SchoolShootingDunbar collection. The webpages belonging to the event and not belonging to the event are 7 and 211, respectively. Over fitting can be seen in the results of this collection and it might be related to the low number of 'No' instances. We believe the results are not comparable. 35
5.9 Solar Eclipse webpage collection 36
5.10 Las Vegas Shooting webpage collection 36
5.11 Trained model accuracy of kept models 38
5.12 Sample of classification results for the 'solareclipse' tweet collection . 39
5.13 Metrics of saved 2017EclipseSolar2017 tweet Logistic Regression model 41
5.14 Metrics of saved 2017EclipseSolar2017 webpage Logistic Regression model 41
5.15 Metrics of saved 2017ShootingLasVegas webpage Logistic Regression model 42
5.16 Example tweet classified in the 'getar-cs5604f17' table. 44
5.17 Example of another tweet classified in the 'getar-cs5604f17' table . . 45
5.18 Example of a tweet not belonging to the event classified in the 'getar-cs5604f17' table 45

List of Tables

3.1 HBase schema 22

6.1 Project folder structure 47
6.2 Data files in the ./data/ directory 48
6.3 Useful log files in the ./oldLogs/ directory 48
6.4 Included bash scripts 50

7.1 Scala source code files. 52

8.2 Timetable of tasks 57

Chapter 1

Introduction

This chapter is dedicated to describe the role of the classification (CLA) team in the overall class project. The CLA team's input is the data collected by the Collection Management Tweets (CMT) team and Collection Management Webpages (CMW) team. The CLA team then outputs the classified data. This data is useful to the SOLR and CTA teams. The main function of the CLA team can be outlined as follows: it receives cleaned raw webpage and tweet data from the two mentioned teams, and it classifies the data into a specific class of events using advanced machine learning schemes. This classified data is put in HBase for other teams. For instance, the SOLR team will index the classified data. A schematic diagram of the CLA team role is shown in Fig. 1.1.

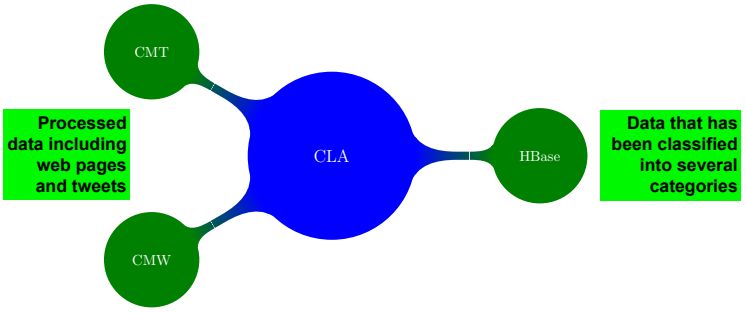

Figure 1.1: Schematic role of classification team

The CLA team of Fall 2016 has already developed a framework for classifying tweets. This framework gives good accuracy and uses good techniques. However, no work has been done on webpage classification. The goal of our team is to enhance the

implemented approach of classifying tweets and to extend this model for webpage classification.

Chapter 2

Literature Review

The CLA team from the Fall 2016 class has made headway with classifying tweets [1]. Their classification process begins with the training phase. They first read raw tweets from HBase and pre-process these raw tweets. This pre-processing includes cleaning the raw tweets, removing stop words, and performing lemmatization. The cleaning in the pre-processing step involves eliminating non-English words, URLs, and emojis. They have made use of the Word2Vec feature selection model to generate word vectors. After the feature selection phase using Word2Vec, they have used the multi-class Linear Regression classifier for classifying the tweets into nine classes. To find the best linear regression model they performed 10-fold cross validation. Both their Word2Vec and Linear Regression models persisted in the Hadoop Distributed File System (HDFS). After they trained the Word2Vec and Classifier models and saved them in HDFS, they performed the prediction step. This prediction step runs periodically (once every 4 hours) using a cron job in Linux. The tweets which need to be classified are read from the database and pre-processed. The trained Word2Vec and Classifier models are loaded from HDFS and this data is passed through the models. The output of the classifier is one of the nine class labels for each tweet. The label is written back in HBase in the column 'real-world-events' which is a part of the column family 'clean-tweet'.

To figure out the best classifier model, the CLA team also used an 'Association rules' classifier model and compared its results with the Word2Vec with Linear Regression classifier. They computed the F-1 score for comparison and found that Word2Vec with Linear Regression gave an F-1 score of 0.96 whereas the Association Rule model gave a score of 0.90 [1].

In the textbook 'An Introduction to Information Retrieval', the authors have included a chapter on Text Classification [2]. This chapter begins with the authors

defining the need for classification as 'the ongoing need of information by different users and applications'. They inform about standing queries and hand-crafted rules and also state the problems associated with these methods. They mention the approach of machine learning based classification and describe it as the approach where 'the set of rules or, more generally, the decision criterion of the text classifier, is learned automatically from training data'. The documents that need to be classified are generally represented in high dimensional spaces. Hence, the authors describe the use of feature selection for improving the efficiency of the classifiers.

Feature selection is defined as the process of picking a subset of terms as features and discarding the rest of the terms. This reduces the overall vocabulary for training the classifier and improves the efficiency. It also helps in removing noise features, i.e., features that lead to misclassification due to insufficiency or inaccuracy of training data available. As described in the book, feature selection methods are based on the principle of computing a utility matrix of all the terms and then selecting those terms which have high value in the utility matrix. The Mutual Information, Chi-Square, and Frequency based feature selection methods are introduced in the book [2].

The feature selection techniques that we researched are described below:

- TF-IDF: Term frequency–inverse document frequency is a statistic that measures the importance of a word for a document. This value increases as the term appears more frequently but is scaled by how often the term appears in the entire corpus. This makes a term that is common throughout the text, such as "the", to not be weighted heavily even though it would appear many times in each specific document [1].
- Word2Vec: Word2Vec creates word embeddings to create a representation of words that capture meaning and semantic relationships and different contexts. Word embeddings mean generating numerical vectors for words using a dictionary. This conversion is done for ease of using these features in machine and deep learning models. The context of the word is defined as the word and its surrounding neighbors. For example, a word might have words preceding it and succeeding it. The word along with its surrounding neighbors forms the context. Two techniques that identify the context of a work are the CBOW (continuous bag of words) and the skip-gram method. The skip-gram method predicts a context given a word. The CBOW technique predicts the probability of a word in a context. 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 for text classification. From a feature selection perspective, since each word generates a vector, we average all 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.

- Doc2Vec: Paragraph vectors were proposed to extend the Word2Vec model. Embeddings are formed by learning from sentences instead of learning from words as is the case with Word2Vec. Doc2Vec has two models: 'DBOW' and 'DMPV'. DBOW works in the same way as skip-gram of Word2Vec. DMPV works as CBOW of Word2Vec. The authors mention that with a large corpus of data, Doc2Vec performs better. They ran Doc2Vec, Word2Vec, and n-gram techniques for a document duplicate identification task. They compare the results using ROC curves and conclude that Doc2Vec performs better for some classes and is comparable for others. The DBOW technique performs better than DMPV [3].

The authors of 'An Introduction to Information Retrieval' also mention improving 'classifier effectiveness' as one of the areas of machine learning research. This research has led to development of advanced classifiers like support vector machines, boosted decision trees, regularized logistic regression, and neural networks. The authors state that many machine learning algorithms have been used for classification and SVMs are very prominent among them. It is worth mentioning that SVMs perform remarkably well. The general idea behind SVMs is designing a large margin classifier. The decision boundary has to be such that it is maximally far from any point of any class in the training data. The decision boundary is specified by a subset of points which defines the position of the separator. These points are the support vectors. Maximizing the margin helps because if we have to put a fat separator between the classes, we have fewer choices of where to put it [2].

SVMs are designed as two-class classifiers. However, many techniques exist for constructing a multi-class classifier from SVMs. One of the basic approaches includes building a one-versus-all classifier and choosing the class which this classifier classifies with the greatest margin. Another approach is to build a set of $\{|C|(|C|-1)/2\}$ one-versus-one classifiers and pick the class that is selected by the most classifiers [2].

A logistic regression classifier model 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.

The Naïve Bayes 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.

We plan on exploring SVMs, Logistic Regression, and Naïve Bayes classifier models for webpage classification along with the above mentioned feature extraction techniques.

Hierarchies are becoming ever more popular for the organization of text documents, particularly on the Web. Web directories and Wikipedia are two examples of such hierarchies. Along with their widespread use comes the need for automated classification of new documents to the categories in the hierarchy. As the size of the hierarchy grows and the number of documents to be classified increases, a number of interesting machine learning problems arise. In particular, it is one of the rare situations where data sparsity remains an issue, despite the vastness of available data: as more documents become available, more classes are also added to the hierarchy, and there is a very high imbalance between the classes at different levels of the hierarchy. Additionally, the statistical dependence of the classes poses challenges and opportunities for new learning methods.

In our literature review we studied a similar Kaggle problem posed in 2014 named 'Large Scale Hierarchical Text Classification' [4] and gained insights upon the different approaches to go about it. The winning solution consists mostly of an ensemble of sparse generative models extending Multinomial Naïve Bayes [5]. The base-classifiers consist of hierarchically smoothed models combining document, label, and hierarchy level Multinomials, with feature pre-processing using variants of TF-IDF and BM25.

The ensemble algorithm optimizes the macro F-score by predicting the documents for each label, instead of the usual prediction of labels per document. Scores for documents are predicted by weighted voting of base-classifier outputs with a Feature-Weighted Linear Stacking. The number of documents per label is chosen using label priors and thresholding of vote scores.

The macro-average method [6] is straightforward. Just take the average of the precision and recall of the system on different sets. For example, the macro-average precision and recall of the system for the given example is,

$$\text{Macro-average precision} = \frac{P1 + P2}{2}$$

$$\text{Macro-average recall} = \frac{R1 + R2}{2}$$

where precision (P) and recall (R) are

$$P = \frac{\text{True Positive}}{\text{True Positive+False Positive}},$$

and

$$R = \frac{\text{True Positive}}{\text{True Positive+False Negative}},$$

respectively. The Macro-average F-Score will be simply the harmonic mean of these two figures.

A hierarchical classifier has been designed in [7] for a medical diagnosis application. This hierarchical classifier includes a training phase and a classification phase. In the training phase, the data are clustered into sub-populations and these are relabeled as sub-classes. A reduced feature set is generated from these results. A super-classifier is then constructed from cluster information and new data. We know that the objective of classifiers is to generate a decision boundary to separate data and put it into classes. Classification algorithms work on the premise of minimizing the error for training data. However, this may lead to overfitting of the model over the training data. Hence, the decision boundaries need to be generalized to some extent. Hierarchical classifiers help in this case. With sub-classes being present, the estimated boundaries can be close to the actual boundary. This improves the accuracy of the classifier.

In the model described in [7], the data is clustered and each cluster is a sub-population. Here, the distribution of each cluster is different than that of the original data. The distribution is based on sub-classes and the decision boundary is calculated. By finding the decision boundary, the classifiers for sub-classes are designed. Every data cluster's feature's mean and covariance also are preserved. Each classifier generates the probability and all these probabilities are added to give the next level (super-class) classification model. This model maintains the strength of the multiple-classifier approach and also solves some of the problems of the multi-classifier approach. This model was run on two large data sets, and sensitivity and specificity were measured to validate the model.

We plan to explore hierarchical classifiers to identify if they would help with text classification of webpages and tweets.

Chapter 3

Requirements, Design And Implementation

3.1 Requirements

In the big picture, when you think of the CLA team as a function which requires input and output data, the input data is provided by the CMT and CMW teams, and the output data is delivered to the SOLR and CTA teams. Therefore it is important for our team to actively communicate with associated teams to have a common expectation about the input/output data.

To provide acceptable training data for classifiers, the input data should be cleaned from characters that are not the subject of classification. For example, there can be non-English words or characters like (<?/=> ...), which should be removed from the input data. The CMW and CMT teams need to clean the data and provide us with clean raw text for webpages and tweets, respectively.

The major goal of our team is to create a classifier model which is trained by the input data. We will explore TF-IDF, Word2Vec, and Doc2Vec feature extraction techniques before passing the features to train our classifier model. We plan to evaluate Logistic Regression and Support Vector Machines classifier models for our data sets. Since the Apache Spark framework provides us with machine learning libraries, we will use them to pursue our goal. Beside these classification models, we would like to evaluate if hierarchical clustering suits for the input data. We will go through chapter 17 of the textbook (hierarchical clustering) to get more theoretical perspectives about possible choices (such as top-down or bottom-up) for our data sets [2].

In the final stage of our work, the output of our model will be a column having class labels and a probability-list column having the probability values for the classes predicted. This output will be written into an HBase column family for the use of other teams.

3.2 Design

3.2.1 Classifying Webpages

One major goal for this semester is to classify webpages provided by the CMW team. We plan to test various models for webpage classification. We will first apply the existing tweet classifier code to webpages, and then we will transition to trying different models and frameworks. We plan on executing these models on the shooting data provided at first and then we will use other data as and when provided by the CMW team. We will do various experiments to find an optimal and efficient model and framework. From there, we will optimize the implementation and finalize the code for the next CLA team.

3.2.2 Tuning Tweet Classification Model

The Fall 2016 CLA team, as explained in Chapter 2, has developed a framework that cleans and pre-processes the tweets, trains the tweet classifier and feature selection models, and uses these models to find features and classify new tweets. The team has tested and compared various feature selection techniques, and they have concluded to use the Word2Vec model for feature selection. The previous team has also tested and optimized the tweet cleaning methodology. The previous team chose Logistic Regression as the classifying model. Their framework uses Apache Spark jobs to minimize the training and prediction time taken by the classifier model.

We will attempt to improve the tweet classification framework by implementing a hierarchical classification framework. This way, we can allow users to search something like Hurricanes > Hurricane Harvey, allowing a broader and more in-depth search. We will also try out other models and compare their performance, selecting and implementing the best one.

3.2.3 Original Framework

The tweet classifier framework from the Fall 2016 CLA team will mostly still be used for tweet classification as it is accurate and fast. However, modifications need to be done to fit it into the new framework. Fig. 3.1 shows the original framework that was used to classify tweets. The design is divided into two main phases: training and prediction. Here we will provide a brief overview of each phase.

Training Phase:

- First, the framework takes the raw training and testing tweets stored in HBase and pre-processes them into 'clean tweet data'.
- The clean tweet data in the overall course HBase table is used to train the Word2Vec model which produces features that are used to train the linear regression classifier.
- Once the Word2Vec and classifier models are trained and tested, they are stored in HDFS.

Prediction Phase:

- Cron, a Linux utility, allows scheduling of parsing new tweets every 4 hours, and handing off to the pre-processing step.
- Similar to the training phase, the pre-processing step cleans the tweets.
- The Word2Vec and classifier models are loaded from HDFS and are used to predict which class (real world event) each tweet belongs to.

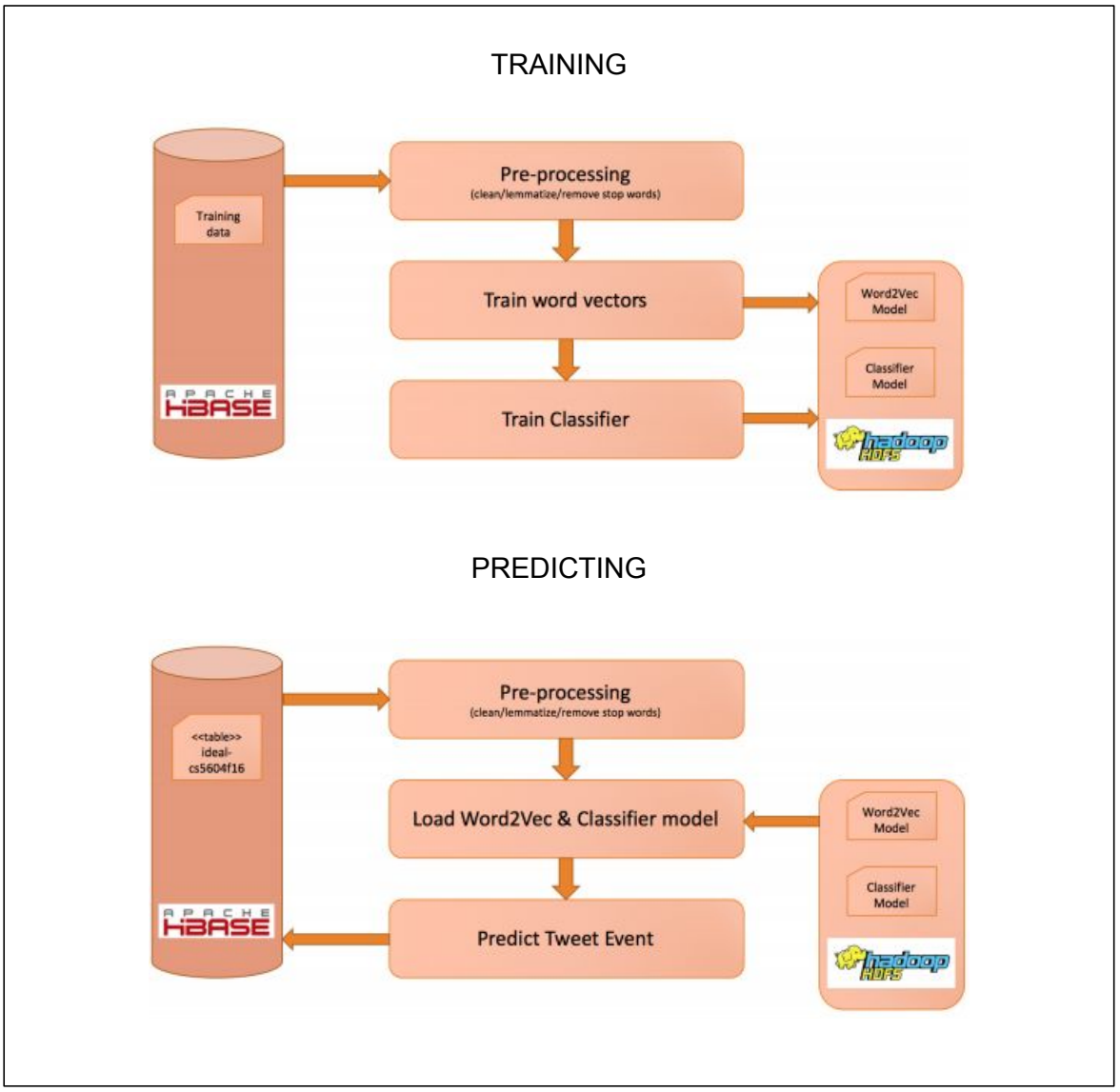

Figure 3.1: Training and predicting processes of the previous framework [1]

From understanding and testing the previous team's provided git repository code, a few noticed drawbacks of the old framework are listed below:

- The code has hard coded classification ID values for classification-labels.
- The process assumes that all data in a table was a single collection with expected table settings.
- The process does not account for handling empty tweets post-cleaning.

- Word2Vec training corpus limited to only the training data set.

3.2.4 New Framework

Our team plans to utilize the previous framework with modifications to fix some of the existing drawbacks. Also, these modifications aim to add webpage classification and support the new class HBase schema. A majority of cleaning done in the pre-processing step will be done by the CMT and CMW teams.

First, the training of Word2Vec models is now separated. This was done because the old process limited the Word2Vec model to a vocabulary only present in the training data. This resulted in a number of words not present in the Word2Vec model's vocabulary when processing documents on the real table. With this training process separated, the Word2Vec model can now be trained off a different data set. It was decided to train off all the documents present in a defined source table. Due to the classification process reading from various columns, all columns that the classification process could look at will be used to train the Word2Vec model. This new training process guarantees that all possible input words from all documents in a table will be in the trained Word2Vec model's vocabulary. Figure 3.2 shows the new Word2Vec model training process.

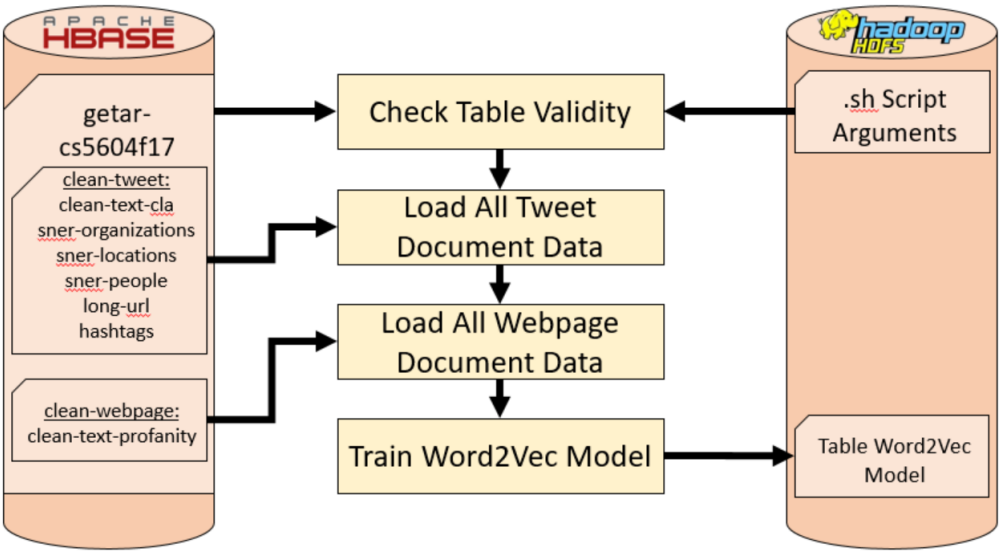

Figure 3.2: New training process of the Word2Vec model

Training of the Logistic Regression models for tweet documents remains relatively the same, though with some changes. First, the input arguments list all of the

classes of the data set, and the training process checks the input training file for a matching number of classes. This is done because hard coded classes will no longer be used. From there, the training and testing data set will be split 80% and 20% respectively with splitting and randomization per class, guaranteeing proper 80% and 20% of training documents per class. Figure 3.3 shows the new training process of tweet Logistic Regression models.

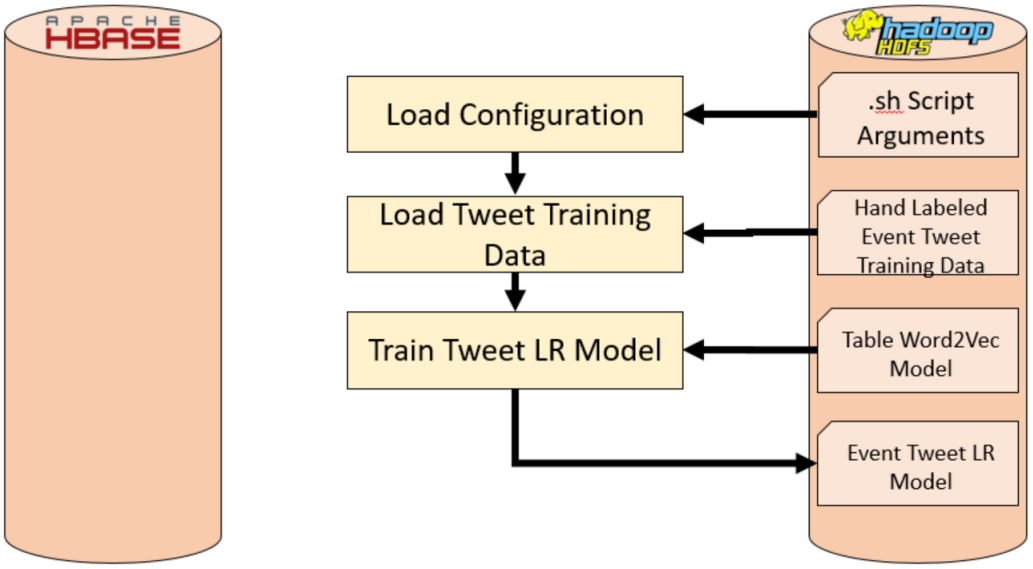

Figure 3.3: New training process of the tweet Logistic Regression model

Training a webpage Logistic Regression model is similar to training the tweet Logistic Regression model. However, the only difference is that data is read from an HBase table rather than from the local training file. This was done because early efforts did not have clean webpage data available for training purposes. When cleaned webpage data became available, it was easier to quickly retrain with the up-to-date cleaned data off the table rather than re-export a new training file with hand labels. Figure 3.4 shows the new webpage Logistic Regression model training process.

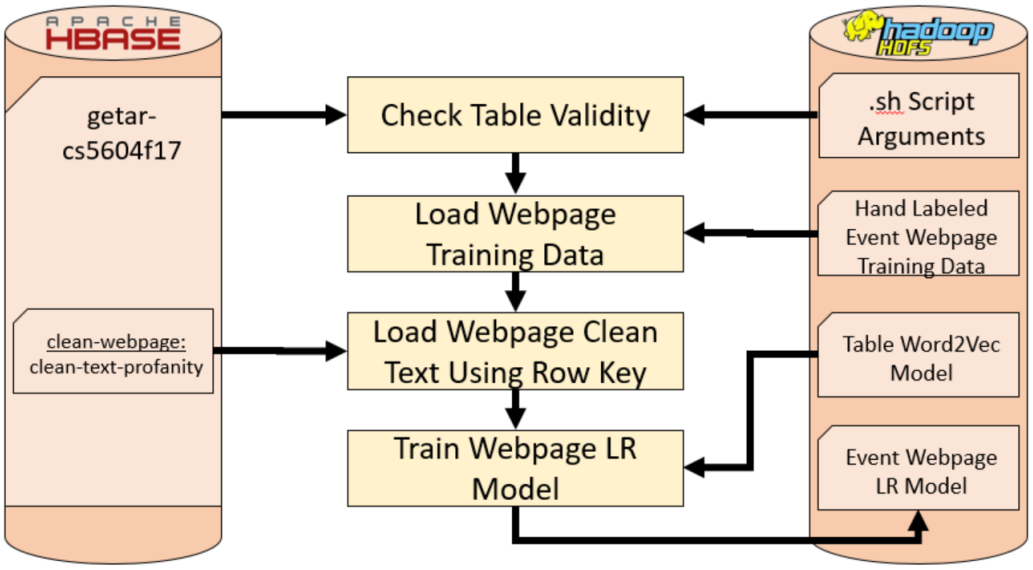

Figure 3.4: New training process of the webpage Logistic Regression model

Classification has been redesigned to support the HBase configuration with scan filtering to read the desired collection and document type from the source data table. For tweet classification, tweets are restricted to 140 (and later 280) characters. This makes tweets have very low information per document. Also, the CMT team generated clean tweet data has a lot of data removed such as hashtags. Therefore, tweet document data had to first be retrieved from multiple columns and concatenated. From there, the tweet data was cleaned for items such as hashtag and URL syntax characters and then classified. Figure 3.5 depicts our new tweet prediction process.

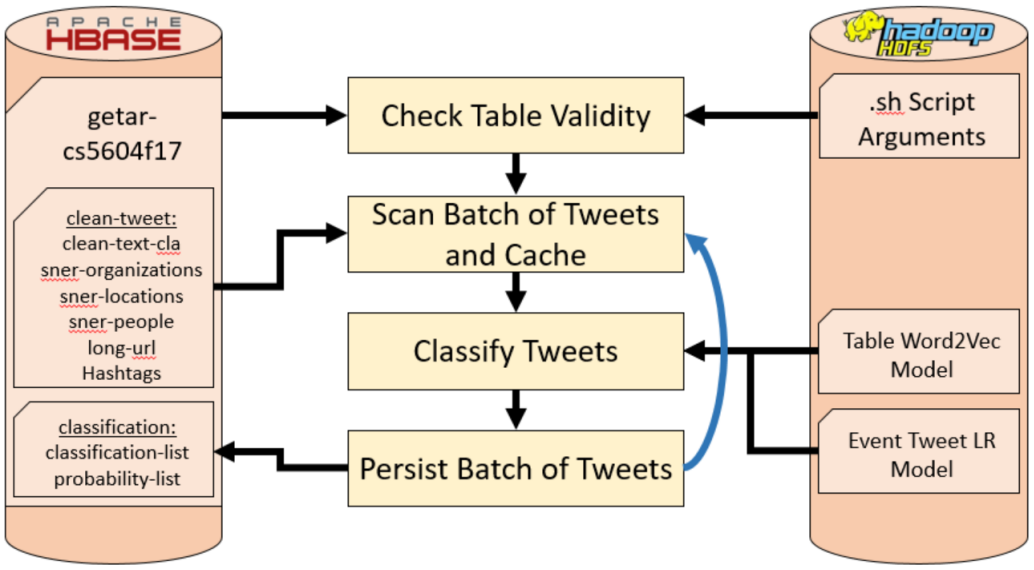

Figure 3.5: New class prediction process for tweets

From experimentation (see Section 5.2), using Word2Vec with Logistic Regression had high accuracy while being very fast. Therefore, a similar process used for classifying tweets was used to classify webpages. Due to webpage clean text being relatively abundant and high in word count, only the webpage clean text was needed to classify webpages. Figure 3.6 depicts our new webpage prediction process.

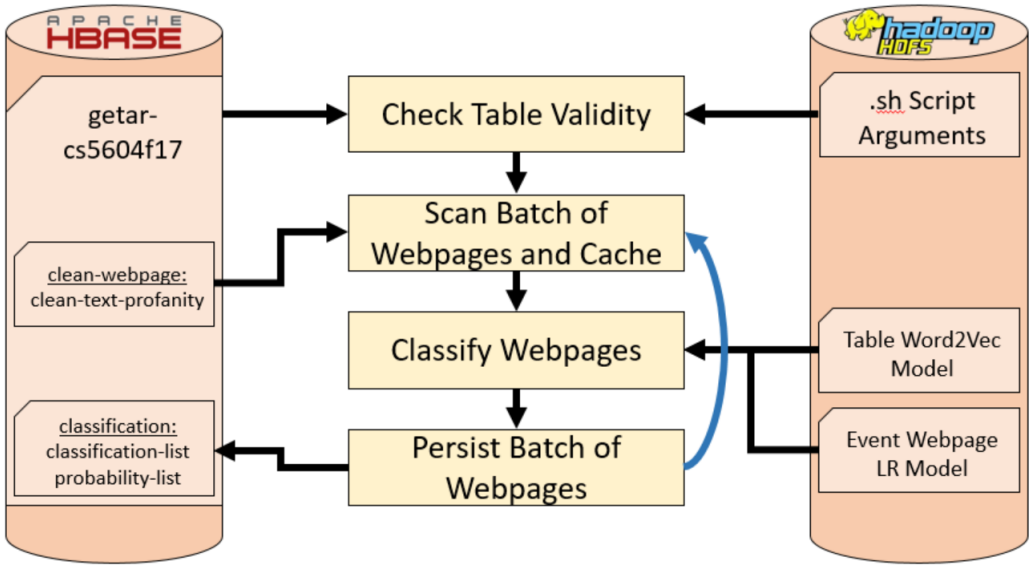

Figure 3.6: New class prediction process for webpages

The HBase schema for the class's shared table was designed and documented this semster. Table 3.1 shows which HBase table columns that the entire classification process will read from or write to.

Column Family	Column	Usage	Example
metadata	collection-name	Input collection filter	"#Solar2017"
metadata	doc-id	Input tweet/webpage filter	"tweet"
clean-tweet	clean-text-cla	Input clean tweet text	"glasses stare sun hurts eyes"
clean-tweet	long-url	Input URL in tweet	"https://www.cnn.com/faulty_glasses/"
clean-tweet	hastags	Input tweet hashtags	"#NASA;#Solar2017"
clean-tweet	sner-organizations	Input SNER text	"NASA"
clean-tweet	sner-locations	Input SNER text	"Virginia"
clean-tweet	sner-people	Input SNER text	"Thomas Edison"
clean-webpage	clean-text-profanity	Input clean webpage text	"glasses dangerous chemicals"
classification	classification-list	Output classification names	"glasses;chemicals"
classification	probability-list	Output class probabilities	"0.899;0.101"

Table 3.1: HBase schema

Chapter 4

Implementation

The main() function of the code has been modified to take in input arguments to determine the runtime configuration. It determines if hand labeling, training, or classification is being done, webpage or tweet data is being processed, which HBase tables to input and output data from, the processed table collection name, and classification classes to use. Currently, bash shell .sh scripts with user defined inputs are used to run the code, and multiple different runtime parameters can be defined at once to, for example, classify tweets for a set of collection names. Eventually, most of these inputs will be loaded from a configuration file and work off of a separate HBase table that maps event names to collection names.

4.1 Hand Labeling Process

4.1.1 Tweets

Our team provides code to hand label tweets in the class cluster, called by a bash script. The code will access directly to the main HBase table, filter out the unrelated tweets based on several predefined filters, and retrieve important columns for classification. Our team filters the tweets based on the correct collection name, tweet document type, false retweet flag, and non-empty clean text. For tweets, our team identified the important input columns as the following:

- clean-tweet:clean-text-cla
- clean-tweet:sner-people
- clean-tweet:sner-locations

- clean-tweet:sner-organizations
- clean-tweet:hashtags
- clean-tweet:long-url

After gathering all the columns, the script will display the raw tweet and ask the developer to label it. Figure 4.1 shows an example of what the script would display after being executed. For our implementation, our team also provides flexibility to choose how many tweets are to be labeled. After all the tweets have been labeled, the tweet row key, tweet clean text and other column information, and classification label, will be stored in a .csv directory on the Hadoop file system. A "hadoop fs -get "$\langle generated\_file\_name \rangle$"" command will pull it to the local file system. This output will be a folder of parts. Use "cat part-* > trainingFile.csv" to merge the parts into a single .csv file for training usage. For our team's objective to classify data for the Solar Eclipse in 2017, we have hand labeled 500 tweets from the "#Eclipse" collection in the "getar-cs5604f17" table.

```
[cloudera@quickstart ISRProject]$ ./labelSolarEclipseCluster.sh

NUMBER OF LABELED TWEETS: 0 / 500
Combinedwords: clean tweet number 1 people1  org1 11 www.1
Label this tweetID: 1 | RAW: raw text 1
```

Figure 4.1: Example of the hand labeling process after running the script

4.1.2 Webpages

Hand labeling the webpage data was more challenging than for tweets. One of the main challenges we faced with the webpage data was the "size" of contents; it could vary from one line to multiple paragraphs. The size difference in data entries causes problems in the process of classification. For example, a spam instance in the data could be fairly middle size (which was good) but a non-spam page might have few words which means it is not high quality as an input for the feature selection models. On the other hand, some of the instances had only non-text contents such as images, advertisements, and videos.

Apart from the obstacles we pointed out, the actual contents of the webpages also held challenges. First, we had to read the entire webpage content rather than just the first few lines. This was because we found that some of the webpages had just the first line about the event and then the rest of the content was not related to the event at all. For example, in the 'Dunbar High School Shooting' data the suspect 'Amy Bishop' was involved in a few other events and not just the shooting. During

our hand labeling we found webpages starting with her name but that would detail some other event which she was a part of.

As we had to read entire webpages, this task took some time. However, we hand labeled about 550 webpages related to the 'Solar Eclipse' event and 800 webpages related to the 'Las Vegas Shooting' event. We also hand labeled six 'School Shooting' collections to test different classification models for the framework.

4.2 Class Cluster Classification

For tweet classification, our team has the tweet classifier Scala code used last year. The problems with this framework were mentioned in Section 3.2. In addition, the provided code repository had code commits made ahead of the previous team's final report submission, deprecating some sections of the previous team's final report. The tweet classification code has been modified to reflect the new framework design described in Section 3.2.4.

4.2.1 General Improvements

Time was taken to remove unused or test code from the master branch of the code such as incomplete functions and injected return nulls. In addition, a lot of hard coded values such as table names, class labels, and file names were included in local function scopes or local object scopes. These local scope variables now refer to higher scope variables so that constants such as column family names do not need to be defined in multiple locations. Also, the code no longer needs to be recompiled to change table names, input training data sets, and model files. This is thanks to using more input argument fields when calling the main() function.

4.2.2 Word2Vec Model Training Data

For classification model training, a Word2Vec model must be trained first. Currently, the Word2Vec model naming scheme is: '<SrcTableName>_table_w2v.model'. Word2Vec models are named after the source data table since its entire corpus was used for training, making it applicable to all events, document types, and collections stored in the table.

To train the Word2Vec model, the code first collects column data of all documents in the table. This column data is gathered through a table scanner, and the columns

read from are only the ones used as inputs to the classification process defined in Table 3.1. To train a Spark library Word2Vec model, the entire corpus must be input, as one massive data structure, in one go. Iterative training of the Spark library Word2Vec model is not possible due to the training function's re-initialization of internal class values. Therefore, a massive (RDD) data structure containing the entire corpus of the table is built first. Large tables may take up to hours to compile the entire corpus, and it is recommended to only train the Word2Vec model on high system memory computers (i.e., >8GB). As a performance metric for estimating how long this process will take, one hour was needed to train the Word2Vec model off of the 3.3 million documents in the class cluster table. This table is composed of mixed tweet and webpage documents, mostly tweets (>75%). After compiling the data into an RDD, the model is trained and saved. Due to the long training times, it is not recommended to retrain the Word2Vec model often.

4.2.3 Logistic Regression Model Training

After a Word2Vec for the source table is trained, Logistic Regression models may be trained for each event in that table. Initially, hand labeled training documents are fed in through a .csv training file. The formatting of the training file should be three comma separated values in the order of: row key, concatenated classification text, and class label. Note that the webpage training code may use a .tsv training file with different layout due to how the initial unlabeled webpage data was pulled. The name of the training file should be: '<EventName>_<DocumentType>_training.csv'. A training and testing document set is needed, so the training document data set must be split.

The training documents are randomly split into 80% training and 20% testing set. The splitting is done randomly on a per class label basis. This means that for every class, 80% of that class's training documents are for training and 20% for testing. This ensures that each class has an adequate amount of training and testing data. However, this does not balance the class data set size against other classes. For example, a training data set has 100 'Solar Eclipse' and 200 'Spam' classified documents. Then the training data sets will each have 80 'Solar Eclipse' and 160 'Spam' documents. This split was used because of a low count of hand labeled training documents, which means it is better to train as much as possible, and further testing for optimal splitting of training to testing data sets is needed. It may also be a good idea to have dedicated training and testing sets that are both hand labeled and hand picked.

Note that, when training the tweet Logistic Regression models, the tweet document

data is read from the .csv training file. However, for training the webpage Logistic Regression models, the webpage document data is read from the specified source table, but the hand labeling and row key are provided from a .csv training file. This was done because clean webpage data was not initially available when the hand labeling of classes was done. Ideally, in the future, all training will read the latest cleaned document data off the table due to possible collection cleaning process adjustments, and the training files may only have the document row key and hand labeled class. Finally, training Logistic Regression models is very quick, only taking around 15 seconds per model for around 600 hand labeled documents of either type.

4.2.4 Predicting Document Classes

The prediction process first checks the source and destination HBase tables for the expected column family names. It will throw an exception if the required column families are missing. See Section 3.2.4 for the HBase Schema. Next, the Word2Vec and Logistic Regression models are loaded. Afterwards, a Scan object and many ColumnValueFilter objects are generated to pull the desired data from the source HBase table. These filters check for empty or missing cleaned input text, as well as for the desired collection and document type. See Section 7.3 for more information on these filters. The scan pulls the clean text and other desired column data for concatenation of the document data to classify. The scan results are pulled in cached batches to not steal too much processing time of the table from other class cluster processes. A batch of tweet scan results includes up to 20,000 tweets per batch. A batch of webpage scan results includes up to 2,000 webpages. Scanning an HBase table is a serial operation done by the driver (main) process, and the results are stored into an RDD. After pulling the filtered batch of documents, the driver process partitions the batch of documents into RDD partitions for parallel class prediction via parallel task processes. The driver process spawns these task processes, where Spark distributes the task to cluster nodes for parallel computing, speeding up the process. Figure 4.2 shows an overview of this RDD partition processing.

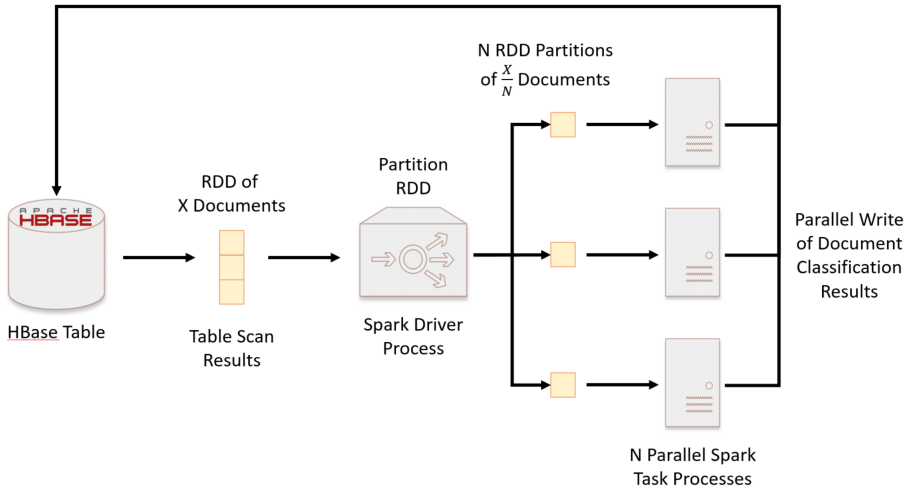

Figure 4.2: Data flow of a RDD of a batch of documents during prediction process

Prediction consists of a quick basic cleaning of some punctuation and symbols as some column data fields that can be accumulated may not be cleaned. See Section 10.5 for Future work that the CMT team can do to help remove this cleaning process from our end to integrate into their cleaning process. Most parts of this process is done in RDD partitions to apply parallel task processes to portions of the batch of documents at the same time. For each RDD partition, the Word2Vec model transforms all document word strings into features, filtering for any words not in the Word2Vec model vocabulary. If none of the document words are in the vocabulary, that document is filtered and marked as class '0.0' for the 'CouldNotClassify' class with a probability of 100%. Then, the document data is classified by the Logistic Regression model using an overridden predictPoint() function in ClassificationUtilities.scala. This is done because the original Logistic Regression API outputs only the final predicted class. The probabilities of each class are normalized over all probabilities. The output of predicting is an RDD with a tuple of the document and an array of probabilities per class. The probabilities are of type double and the index in the array is the class label ID number.

After predicting, the probabilities and classes of the documents in each partition are output to the destination HBase table. This partition-based writing can be done in parallel as each document per partition should always have a unique row key, naturally preventing concurrent writing to the same row. The probability array index is not the class name, rather the class label. The class label ID is mapped to the class name string. The probabilities are the elements in the array. The outputs are lists that are semi-colon (';') separated. The lists are indexed such that the first class label in the 'classification-list' corresponds to the respective first

probability in the 'probability-list'. All classes with a probability of 0 are omitted. A higher probability threshold may be used in the future for omitting low probability labels. After writing the classification information to the destination HBase table, the predict phase will loop over the next batch of documents until there are no more documents matching the scan filter requirements.

Classifying tweets and webpages have some slight differences. The scans that return document data are different. This is because of the different table columns being read. Also, tweet clean text data is sparse, so more columns of data need to be read per tweet document to maximize classification accuracy.

Chapter 5

Experiments

5.1 Classifying Webpages with Tweet Framework

As no work had been done with webpage data previously, it was decided to adapt the original Word2Vec with Logistic Regression framework of classifying tweets to classify webpage data as a functionality test. Several data sets were given to our team to run our tests and benchmarks by the GTA. Two shooting collections were picked. These were the '2009SchoolShootingDunbar' and the '2008SchoolShootingNIU' collections. As the original framework had multi-class classification, '2009SchoolShootingDunbar' was encoded as instances of class 0.0 and the '2008SchoolShootingNIU' as class 1.0.

The '2009SchoolShootingDunbar' instances were all from Facebook webpages and so a lot of spam contents, non-English text, and other unnecessary data were included. On the other hand, though the '2008SchoolShootingNIU' data set was small, it had good document quality. The tweet cleaning code performed a basic cleaning for the provided webpage data. Since the CMW team had planned to provide a better quality cleaned data set, it was decided to run this experiment on the above data set only as a quick test. With that, it was possible to evaluate how well the tweet classifier worked on the webpage classification. The data sets were broken up randomly with about a 50:50 ratio for training:testing purposes. This was done as the smaller 2008SchoolShootingNIU collection would have few test documents. The data had 125 training and 132 testing documents. The models were retrained multiple times, and the best scoring model was kept. Figure 5.1 shows the results obtained.

```
Took 22.595 seconds for the training.
<---- done
Took 0.022 seconds for Prediction.

***********   Class:0.0   *************
F1 Score:0.9692307692307692
True Positive:0.9545454545454546
False Positive:0.2857142857142857
Precision:0.984375
Recall:0.9545454545454546

***********   Class:1.0   *************
F1 Score:0.5555555555555556
True Positive:0.7142857142857143
False Positive:0.045454545454545456
Precision:0.45454545454545453
Recall:0.7142857142857143

Confusion Matrix
126.0  6.0
2.0    5.0

***********   Classifier Results for LRW2VClassifier   *************
Weighted F1-Measure = 0.9483982045133124
Weighted Precision = 0.957692936559843
Weighted Recall = 0.9424460431654677
Macro F1 = 0.7623931623931623
Micro F1 = 0.9424460431654677

***********   End of Classifier Results for LRW2VClassifier   *************
```

Figure 5.1: Results for Word2Vec with Logistic Regression for initial webpage classification for 2009SchoolShootingDunbar and North Illinois University Shooting

The results showed that the class 0.0 (2009SchoolShootingDunbar) classification performance is really high, which was not expected due to the high spam count of the data set. Class 1.0 (2008SchoolShootingNIU) classification performed very poorly as expected due to the small data set. This data set had only 14 instances. This shows that a large training data set is important to increase classification performance. Analyzing the '2009SchoolShootingDunbar' data set in both the original and cleaned content strings showed some patterns of how spam data was treated as classified correctly into the collection event. The Facebook webpage had common substrings such as "facebook" in all the entries, both English and non-English. This is shown in Figure 5.2.

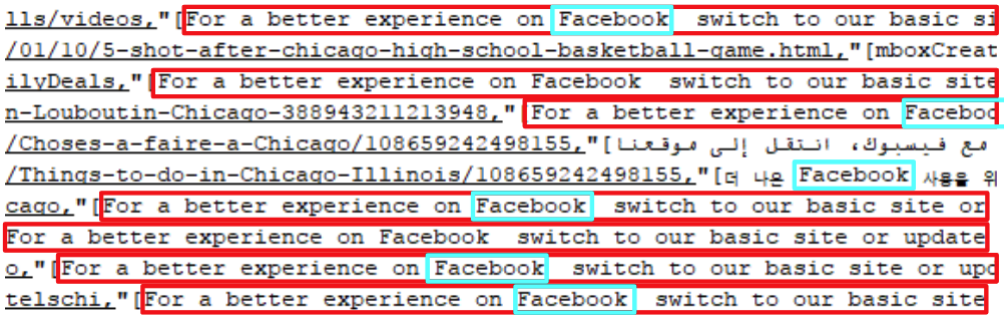

Figure 5.2: Example of common substrings in documents; each line is one document

To avoid an issue of a spam instance from a collection being classified as belonging to that event, a few methods of fixing this were devised: switch Word2Vec for another word embedding algorithm such as Doc2Vec which puts less emphasis on the common substrings, clean the document with specific types of stop words such as the webpage's website name, or provide better training data for the Logistic Regression model. Most of the non-English entries were cleaned with only the English words such as "facebook" left, leaving the Logistic Regression model too little data to accurately classify the documents with. Also, because all of the 2018SchoolShootingDunbar webpages had the common Facebook substrings after cleaning, the Logistic Regression model became trained to recognize Facebook webpages as belonging to only the 2018SchoolShootingDunbar class. Better training data which does not enable the Logistic Regression model to associate the Facebook substring word embeddings to a class is needed. A spam class data set for training could also be added with the Facebook substrings to encourage the Logistic Regression model to associate documents with only the Facebook substrings after cleaning as spam rather than an event class. In the end, this analysis shows that quality of the training data is also as important as quantity.

5.2 Model Selection for Webpage Classification

Based on the experiment results of Section 5.1 we realized the need of two important things: cleaned and good quantity of webpage data, and a model which would work efficiently.

Before the CMW team provided us with the cleaned data, we used the data for the 'shooting collections' provided by the GTA. We built binary classification models using different feature selection and classification techniques.

In order to build the right framework for webpage data collections, we tried different combinations of TF-IDF and Word2Vec (as feature selectors) with Logistic Regression and SVM (as classifier models)

We used the six shooting collections provided and built binary classification models for each of these. These shooting collections are the '2010SchoolShootingUniversityAlabama', '2008SchoolShootingNIU ', '2014SchoolShootingReynoldsHighSchool', '2012SchoolShootingSandyHook','2015SchoolShootingUmpqua' and the '2009SchoolShootingDunbar' collection. This was done to figure out the optimal combination which produces the highest accuracy and fastest processing time.

For each of the shooting collections we built four combinations of the above models. We randomly split the data into an 80:20 ratio of training and testing data. We then performed tokenization on the body of text to get each separate word. After this we removed stop words. These are words which occur frequently over the whole text. Word2Vec was then performed on this processed data set for feature extraction. Once this was done, Logistic Regression was performed and the threshold was set to 0.5 for logistic regression. This whole process was done using the Pipeline feature in Spark. In Figs 5.3, 5.4, 5.5, 5.6, 5.7, and 5.8 we show the quality of each combination.

In these initial comparisons, we compare the accuracy and the F-1 score. When we had the data provided by the CMW team for the 'Solar Eclipse' and the 'Las Vegas Shooting' collections, we evaluated other performance measures before selecting our final model.

Model Used	Accuracy	F_1 score
TF-IDF+ Logistic Regression	0.924	0.931
TF-IDF+ Support Vector Machine	0.954	0.943
Word2Vec+Logistic Regression	0.969	0.952
Word2Vec+ Support Vector Machine	0.974	0.962

Figure 5.3: 2010SchoolShootingUniversityAlabama collection. The webpages belonging to the event and not belonging to the event are 71 and 180, respectively. The quality and number of instances were enough to conclude that the accuracies of different combinations are comparable. Word2Vec with SVM and Logistic Regression have better performance.

Model Used	Accuracy	F_1 score
TF-IDF+ Logistic Regression	1.0	1.0
TF-IDF+ Support Vector Machine	1.0	1.0
Word2Vec+Logistic Regression	1.0	1.0
Word2Vec+ Support Vector Machine	1.0	1.0

Figure 5.4: 2008SchoolShootingNIU collection. The webpages belonging to the event and not belonging to the event are 1 and 14, respectively. The number of instances for each class is not sufficient, and hence the results are not reliable.

Model Used	Accuracy	F_1 score
TF-IDF+ Logistic Regression	0.50	0.39
TF-IDF+ Support Vector Machine	0.73	0.62
Word2Vec+Logistic Regression	0.64	0.49
Word2Vec+ Support Vector Machine	0.59	0.41

Figure 5.5: 2014SchoolShootingReynoldsHighSchool collection. The webpages belonging to the event and not belonging to the event are 34 and 192, respectively. The instances were insufficient and were of poor quality. Accordingly, we believe the results are not comparable.

Model Used	Accuracy	F_1 score
TF-IDF+ Logistic Regression	0.86	0.78
TF-IDF+ Support Vector Machine	0.83	0.74
Word2Vec+Logistic Regression	0.92	0.88
Word2Vec+ Support Vector Machine	0.90	0.84

Figure 5.6: 2012SchoolShootingSandyHook collection. The webpages belonging to the event and not belonging to the event are 31 and 303, respectively. For this collection we believe that even though the instances are few, they are good enough such that we can conclude Word2Vec with Logistic Regression has better performance.

Model Used	Accuracy	F_1 score
TF-IDF+ Logistic Regression	0.91	0.85
TF-IDF+ Support Vector Machine	0.73	0.60
Word2Vec+Logistic Regression	0.80	0.71
Word2Vec+ Support Vector Machine	0.75	0.68

Figure 5.7: 2015SchoolShootingUmpqua collection. The webpages belonging to the event and not belonging to the event are 9 and 408, respectively. The numbers of instances are very unbalanced and their quality is not good enough either. Thus no conclusion about the performance can be made.

Model Used	Accuracy	F_1 score
TF-IDF+ Logistic Regression	1.0	1.0
TF-IDF+ Support Vector Machine	1.0	1.0
Word2Vec+Logistic Regression	1.0	1.0
Word2Vec+ Support Vector Machine	0.974	0.69

Figure 5.8: 2009SchoolShootingDunbar collection. The webpages belonging to the event and not belonging to the event are 7 and 211, respectively. Over fitting can be seen in the results of this collection and it might be related to the low number of 'No' instances. We believe the results are not comparable.

As we see in many comparisons above, due to insufficiency of data, we can't judge our models accurately. Hence, we evaluated all of the models for the collections obtained by the CMW team.

The data provided by the CMW team had good quality webpages, i.e., webpages with multiple paragraphs. Some of these webpages had appropriate content which related to the event. There were many quality webpages which did not relate to the event as well. The webpage team had cleaned the data along with removal of profane words.

We hand labeled 550 'Solar Eclipse' webpages by reading through the content of each webpage. We performed a 80:20 split on the 550 webpages and built our models. Below are the results of our models for this data.

Type of model used	Precision	Recall	F-1 Score
TF IDF- LR	0.89	0.73	0.80
TF IDF- SVM	0.89	0.8	0.84
Word2Vec- LR	1.0	0.75	0.85
Word2Vec- SVM	1.0	0.75	0.85

Figure 5.9: Solar Eclipse webpage collection

We also hand labeled 800 'Las Vegas Shooting' webpages by reading through the content of each webpage. We performed a 80:20 split on the 800 webpages and built our models. Below are the results of our models for this data.

Type of model used	Accuracy	Precision	F-1 Score
TF IDF- LR	0.68	0.80	0.58
TF IDF- SVM	0.68	0.80	0.58
Word2Vec- LR	0.67	0.82	0.54
Word2Vec- SVM	0.73	0.82	0.64

Figure 5.10: Las Vegas Shooting webpage collection

From Figures 5.9 and 5.10, we concluded that the combination of Word2Vec with Logistic Regression can give us the best performance among the other combinations and so we have planned to use this combination for future cleaned collections. This conclusion was reached as the performance of this model remained consistently good throughout our analysis. You can also observe that Word2Vec with SVM performs better than Logistic Regression in some cases and is on-par with Logistic Regression in other cases. However, SVM is slower than Logistic Regression and considering the amount of data we will have with the addition to more events and more webpages, we decided to go ahead with Word2Vec with Logistic Regression.

5.3 Initial Classification of Tweets on Class Cluster

With the original tweet classification framework refactored to function and meet the class HBase table schema, the framework was executed on the local machine and class cluster. A few unexpected classification result cases were found, because the 'classify tweets' method written by the previous team had some issues. One is that they will not assign any class to tweets with document text that is completely not in the Word2Vec vocabulary. Also, when tweets have the same probability for two classes, the Spark library default classification prediction method will choose the first class with the highest probability. This means that cases with 50%/50% classification will only result in the first 50% class it sees.

To fix these issues, the empty tweets and tweets where all text is not in the Word2Vec vocabulary have been assigned the class 0.0 for 'CouldNotClassify' to account for this edge case. For testing purposes, the previous team used an overridden '.predictPoint()' function defined in the ' ClassificationUtility.scala' to not only return the highest probability class but also the probabilities of all classes. This is because returning only the highest probability class is the default functionality of the '.predictPoint()' function in the Spark library. This overridden function is now used in the normal classification process to return all class probabilities. After making these fixes, all 100 tweets in the 'eclipsedatasample1' table's 'solareclipse' collection documents were classified. This 'eclipsedatasample1' table was generated by the CMT team as a test output of their HBase interactions, but a number of these documents were retweets or were empty after cleaning. Note that the models were trained with all collections in the 'eclipsedatasample1' table for the '2017EclipseSolar2017' event, but only the 'solareclipse' collection was classified. It was initially thought that models should be trained per collection rather than per event, but it was clarified to not be a mistake to train and classify in the way that was done. Multiple training runs were done, generating newly trained models every run. The best performing model in terms of weighted F-1, precision, and recall scores was saved for classification usage. Figure 5.11 shows the saved model performance metrics used for classification.

```
***********   Class:1.0   *************
F1 Score:0.8562300319488819
True Positive:0.8375
False Positive:0.03667953667953668
Precision:0.8758169934640523
Recall:0.8375

***********   Class:2.0   *************
F1 Score:0.9555125725338491
True Positive:0.9536679536679536
False Positive:0.1375
Precision:0.9573643410852714
Recall:0.9536679536679536

Confusion Matrix
134.0  22.0
19.0   494.0

***********   Classifier Results for LRW2VClassifier   *************
Weighted F1-Measure = 0.932083064431202
Weighted Precision = 0.9381201292572552
Weighted Recall = 0.9262536873156342
Macro F1 = 0.9058713022413656
Micro F1 = 0.9262536873156342
```

Figure 5.11: Trained model accuracy of kept models

Figure 5.12 shows the classification done by the trained models on the class cluster. A lot of tweets have 50%/50% probabilities. It was recommended that an "Unknown" default class can be made for such cases if no anomalies were found to cause such results. Further investigation showed that training data splitting was done incorrectly, where some documents under one class were also trained as if under the other class. However, after fixing this issue, some 50%/50% classified documents changed, while some still remained 50%/50%.

```
1001-911328047390916609  column=classification:probability-list, timestamp=1510190063087, value=0.9410470913744431;0.03132303854180851
1001-911329019521585153  column=classification:classification-list, timestamp=1510190064289, value=SolarEclipse;NotSolarEclipse
1001-911329019521585153  column=classification:probability-list, timestamp=1510190064291, value=0.9410470913744431;0.03132303854180851
1001-911332108819402752  column=classification:classification-list, timestamp=1510190064393, value=CouldNotClassify
1001-911332108819402752  column=classification:probability-list, timestamp=1510190064396, value=1.0
1001-911332395751796736  column=classification:classification-list, timestamp=1510190064502, value=SolarEclipse;NotSolarEclipse
1001-911332395751796736  column=classification:probability-list, timestamp=1510190064504, value=0.5;0.5
1001-911332503474057216  column=classification:classification-list, timestamp=1510190064605, value=CouldNotClassify
1001-911332503474057216  column=classification:probability-list, timestamp=1510190064608, value=1.0
1001-911332551809216513  column=classification:classification-list, timestamp=1510190065807, value=SolarEclipse;NotSolarEclipse
1001-911332551809216513  column=classification:probability-list, timestamp=1510190065810, value=0.9998767034658456;6.165586903215289E-5
1001-911334195284279296  column=classification:classification-list, timestamp=1510190067010, value=SolarEclipse;NotSolarEclipse
1001-911334195284279296  column=classification:probability-list, timestamp=1510190067013, value=0.5;0.5
1001-911337688720138241  column=classification:classification-list, timestamp=1510190068208, value=SolarEclipse;NotSolarEclipse
1001-911337688720138241  column=classification:probability-list, timestamp=1510190068211, value=0.5;0.5
1001-911340245328777216  column=classification:classification-list, timestamp=1510190068319, value=SolarEclipse;NotSolarEclipse
1001-911340245328777216  column=classification:probability-list, timestamp=1510190068321, value=0.5;0.5
1001-911340527437611013  column=classification:classification-list, timestamp=1510190068423, value=CouldNotClassify
1001-911340527437611013  column=classification:probability-list, timestamp=1510190068426, value=1.0
1001-911341055928340480  column=classification:classification-list, timestamp=1510190068534, value=SolarEclipse;NotSolarEclipse
1001-911341055928340480  column=classification:probability-list, timestamp=1510190068537, value=0.5;0.5
1001-911348566265868289  column=classification:classification-list, timestamp=1510190069735, value=SolarEclipse;NotSolarEclipse
1001-911348566265868289  column=classification:probability-list, timestamp=1510190069737, value=0.5;0.5
1001-911350231400300544  column=classification:classification-list, timestamp=1510190070948, value=SolarEclipse;NotSolarEclipse
1001-911350231400300544  column=classification:probability-list, timestamp=1510190070951, value=0.5;0.5
1001-911351407000870912  column=classification:classification-list, timestamp=1510190071057, value=CouldNotClassify
1001-911351407000870912  column=classification:probability-list, timestamp=1510190071059, value=1.0
1001-911357015485767680  column=classification:classification-list, timestamp=1510190071162, value=SolarEclipse;NotSolarEclipse
1001-911357015485767680  column=classification:probability-list, timestamp=1510190071164, value=0.8974234701680839;0.05715057230044584
1001-911358365976186880  column=classification:classification-list, timestamp=1510190071262, value=SolarEclipse;NotSolarEclipse
1001-911358365976186880  column=classification:probability-list, timestamp=1510190071265, value=0.5;0.5
1001-911358711100317697  column=classification:classification-list, timestamp=1510190073247, value=SolarEclipse;NotSolarEclipse
1001-911358711100317697  column=classification:probability-list, timestamp=1510190073250, value=0.5;0.5
1001-911359196653699073  column=classification:classification-list, timestamp=1510190074496, value=SolarEclipse;NotSolarEclipse
1001-911359196653699073  column=classification:probability-list, timestamp=1510190074498, value=0.5;0.5
1001-911362518060343296  column=classification:classification-list, timestamp=1510190077305, value=SolarEclipse;NotSolarEclipse
1001-911362518060343296  column=classification:probability-list, timestamp=1510190077308, value=0.5;0.5
1001-911363192554688512  column=classification:classification-list, timestamp=1510190078677, value=SolarEclipse;NotSolarEclipse
1001-911363192554688512  column=classification:probability-list, timestamp=1510190078680, value=0.5;0.5
1001-911364311033040901  column=classification:classification-list, timestamp=1510190080099, value=SolarEclipse;NotSolarEclipse
1001-911364311033040901  column=classification:probability-list, timestamp=1510190080214, value=0.5;0.5
1001-911367065663152128  column=classification:classification-list, timestamp=1510190080711, value=SolarEclipse;NotSolarEclipse
1001-911367065663152128  column=classification:probability-list, timestamp=1510190080713, value=0.5;0.5
1001-911367259955843073  column=classification:classification-list, timestamp=1510190081913, value=SolarEclipse;NotSolarEclipse
1001-911367259955843073  column=classification:probability-list, timestamp=1510190081916, value=0.5;0.5
1001-911367301911449601  column=classification:classification-list, timestamp=1510190083114, value=SolarEclipse;NotSolarEclipse
1001-911367301911449601  column=classification:probability-list, timestamp=1510190083117, value=0.9559086735054924;0.02306252036233582
```

Figure 5.12: Sample of classification results for the 'solareclipse' tweet collection

Checking the clean text data showed that a number of tweets only included hashtags or webpage URLs. The cleaned text in the 'clean-text-cla' column removes all hashtags and webpage URLs out of the clean text. This resulted in empty clean texts despite the tweet having useful data in the text. After this experiment, it was decided that the training data and classification process needed to include data from other columns to maximize the information provided by the tweet document. Therefore, a number of tweets were marked as 'CouldNotClassify' due to empty cleaned tweet text, while a number of tweets were correctly classified (with a number of results still 50%/50%). The training data had a large number of tweets that were empty as well, which means the models had a smaller training data set than the tweet count suggests.

Due to the low quantity of training document data, the 50%/50% results still appearing was suspected to be caused by insufficient training of the Logistic Regression model. In traditional binary Logistic Regression, 50%/50% probabilities translates to a Boolean 0. In the Spark implementation for binary Logistic Regression, the first class it sees in an all equal class probabilities case (such as 50%/50%) is the 0

Boolean result. Therefore, for a multi-class situation resulting in all classes having equal probabilities, the document should be marked as 'Unknown'. However, for binary classification cases such as this one, 50% probability should translate to NOT in the event class, and so the minimum threshold for a document to be related to an event should be greater than 50%.

5.4 Classification of Documents on getar-cs5604f17

After fixing discovered issues from the previous experiment and implementing webpage classification into the framework, the final new framework implementation was tested. This final implementation is what is reflected in the design and implementation chapters of this report. New tweet training data was generated using document data that was accumulated from multiple columns to not have empty tweet text data. A Word2Vec model was trained with the entire corpus on the 'getar-cs5604f17' HBase table as of 6 Dec 2017. This model has a 42,350,232 vocabulary of unique words. Originally, it was desired to use a pre-trained 300 billion word Google News Word2Vec model, but see Section 10.4 as there are memory constraints to the Spark Word2Vec model vocabulary.

For Logistic Regression models, 500 tweets were hand labeled and used for training the '2017EclipseSolar2017' event tweet Logistic Regression model. Specifically, 104 webpages were labeled as "NOT2017EclipseSolar2017", and 396 webpages were labeled as "2017SolarEclipse2017". Figure 5.13 shows the performance metrics of this model. 544 webpages were hand labeled for the '2017EclipseSolar2017' event webpage Logistic Regression model. Specifically, 221 webpages were labeled as "NOT2017EclipseSolar2017", and 323 webpages were labeled as "2017SolarEclipse2017". Figure 5.14 shows the performance metrics of this model. 798 webpages were labeled for the '2017ShootingLasVegas' event webpage Logistic Regression model. Specifically, 466 webpages were labeled as "NOT2017EclipseSolar2017", and 332 webpages were labeled as "2017SolarEclipse2017". Figure 5.15 shows the performance metrics of this model. Let it be noted that, for all of the trained model metrics, the F-1 score is equal to the recall and precision scores by pure coincidence. This is due to the confusion matrix results (shown in the figures). Re-running the current training code again, yielded a different confusion matrix with non-equal F-1, recall, and precision scores. These models were the ones used for class cluster classification as they were the highest performing ones based on these performance metrics after greater than 10 training runs. The solar eclipse models do not have high accuracy, so in the future, more hand labeled data for training is needed to improve accuracy. The Las Vegas shooting model has a high enough score to be considered accurate enough.

Full logs of the training process for these Logistic Regression models will be saved, and these logs are available in the 'oldLogs' folder as described in Section 6.2.

```
**********   Class:1.0   **********
F1 Score:0.8875
True Positive:0.8875
False Positive:0.42857142857142855
Precision:0.8875
Recall:0.8875

**********   Class:2.0   **********
F1 Score:0.5714285714285714
True Positive:0.5714285714285714
False Positive:0.1125
Precision:0.5714285714285714
Recall:0.5714285714285714

Confusion Matrix
71.0  9.0
9.0   12.0

**********   Classifier Results for LRW2VClassifier   **********
Weighted F1-Measure = 0.8217821782178217
Weighted Precision = 0.8217821782178217
Weighted Recall = 0.8217821782178217
Macro F1 = 0.7294642857142857
Micro F1 = 0.8217821782178217
```

Figure 5.13: Metrics of saved 2017EclipseSolar2017 tweet Logistic Regression model

```
**********   Class:1.0   **********
F1 Score:0.8153846153846154
True Positive:0.8153846153846154
False Positive:0.27906976744186046
Precision:0.8153846153846154
Recall:0.8153846153846154

**********   Class:2.0   **********
F1 Score:0.7209302325581395
True Positive:0.7209302325581395
False Positive:0.18461538461538463
Precision:0.7209302325581395
Recall:0.7209302325581395

Confusion Matrix
53.0  12.0
12.0  31.0

**********   Classifier Results for LRW2VClassifier   **********
Weighted F1-Measure = 0.7777777777777777
Weighted Precision = 0.7777777777777777
Weighted Recall = 0.7777777777777777
Macro F1 = 0.7681574239713774
Micro F1 = 0.7777777777777778
```

Figure 5.14: Metrics of saved 2017EclipseSolar2017 webpage Logistic Regression model

```
***********   Class:1.0   *************
F1 Score:0.9253731343283582
True Positive:0.9253731343283582
False Positive:0.05319148936170213
Precision:0.9253731343283582
Recall:0.9253731343283582

***********   Class:2.0   *************
F1 Score:0.9468085106382979
True Positive:0.9468085106382979
False Positive:0.07462686567164178
Precision:0.9468085106382979
Recall:0.9468085106382979

Confusion Matrix
62.0  5.0
5.0   89.0

***********   Classifier Results for LRW2VClassifier   *************
Weighted F1-Measure = 0.9378881987577641
Weighted Precision = 0.9378881987577641
Weighted Recall = 0.9378881987577641
Macro F1 = 0.936090822483328
Micro F1 = 0.937888198757764
```

Figure 5.15: Metrics of saved 2017ShootingLasVegas webpage Logistic Regression model

These models were then used for classification of data on the 'getar-cs5604f17' HBase table. The full classification logs are available at the file location described in Section 6.2. The general classification performance metrics indicate that reading from and writing to the HBase table is the most time consuming process, especially reading from the HBase table. As previously mentioned, the class prediction process is done in batches, and classifying and writing to the HBase table is done in parallel. However, reading from the table can only be done serially due to the nature of the HBase scan library. Generally, predicting (clean document data, vectorize using a Word2Vec model, and Logistic Regression classification) a batch of webpages or tweets will take at most up to 20 milliseconds maximum based on the classification process logs. However, the classification processing speed is orders of magnitude slower, resulting in about 33 webpages per second and 360 tweets per second. Processing speed can vary wildly by more than 50% per batch depending on how busy the table is as well. One possible way to speed up reading from the table is to use asynchronous scanning which involves pipelining the scanning and processing of documents, greatly increasing the throughput of the entire classification process. However, this will put more burden on the servers, so considerations must be taken for server and network workload if this is to be implemented. There is more burden due to the fact that the existing process first reads a set of documents from the table, classifies, and then writes it back to the table in a serial manner, resulting in bursts of activity. If pipelining is used, the previously mentioned 3 steps could be

done continuously and as concurrently as possible.

5.5 Classifier Accuracy

To confirm the classification accuracy of our classifier model and framework, sampling and testing were done on the solar eclipse tweets that were classified on the class cluster. However, the webpage training data included the entire webpage collections, to have enough data for reasonable training. Therefore, the accuracy of the webpage classifiers was not further studied, and is not discussed in this section. This is because all of the webpage data was used for training and testing, and no additional data was available for further classifier quality assessment. Consequently, for now, the performance of the webpage classification models can only be determined based on the testing done during the training process. It is recommended for future work, as mentioned in Section 10.6, to further investigate the performance of the classification models when new webpage data for the event is available.

The tweet classification performance checking procedure is similar to the hand labeling process. The exception is that sampling was done by looking at 1 in every 500 tweets for the event, and 100 tweets were sampled for this test. The 1 in 500 sampling was done to have a wide distribution or variety of tweets across the table and to avoid most retweet documents, which often appear serially in the table. Every time we retrieve a tweet, we look at our classification for the tweet to manually decide whether it is a true positive, false positive, true negative, or false negative. After we checked 100 tweets, we obtained the confusion matrix which lets us calculate our performance metrics.

As previously mentioned, there is an existing issue of 50% belonging to an event and 50% not belonging to an event, where most of these documents appear to be spam when manually checked. Therefore, any tweets that have 50%/50% classifications were marked as NOT relevant to the event in this experiment, along with tweets having <50% probabilities of being relevant to the event. Tweets that have a >50% probability of being relevant to the event were treated as positive or as classified relevant to the event. From the performance checking, we found the following:

- True Positive: 21 out of 100
- False Positive: 4 out of 100
- True Negative: 59 out of 100
- False Negative: 16 out of 100

- Accuracy: .80
- Precision: .84
- Recall: .57
- F-1 Score: .68

The F-1 score is lower than the score measured in the training process. This is largely due to the low recall score. Many of the 50%/50% labeled tweets were indeed true negatives, and there were many spam tweets in this accuracy test set. However, a number of false negatives were found in the 50%/50% tweets, where they should be classified as relevant to the event. Therefore, the recall score went down. Considering the training set with these 50%/50% tweets, a small number of words in the document text were not in the training document set. Also, the words that were in the training set were normally more generic and existed in both relevant and NOT relevant classes. Examples of such words would be 'solar', 'eclipse', 'viewing', etc. This is because spam would often use relevant terms for advertising while relevant tweets would of course also use such terms. In the future, these 50%/50% should definitely be added to the training document set. This should allow the classifier to learn how to handle these documents better.

5.6 Classifier Examples

In this section, we provide a few examples of the tweets that we have classified on the 'getar-cs5604f17' HBase table using our classifier model. Figure 5.16 provides an example of a tweet that is related to the '2017EclipseSolar2017' event and is correctly classified by our classifier with a high probability value.

RT @NASA: #Eclipse2017 happens in less than a month--Aug. 21, 2017! Learn how, where, and when to watch at https://t.co/K29zBFAvh4 https://\xE2\x80\xA6

Classification list-

2017EclipseSolar2017;NOT2017EclipseSolar2017

Probability list-

0.9999982119289905;1.78807100944159E-6

Figure 5.16: Example tweet classified in the 'getar-cs5604f17' table.

Figure 5.17 provides an example of a tweet that is related to the '2017EclipseSolar2017' event but does not contain the usual content that the solar eclipse tweet would have. However, our classifier catches this and is able to classify this tweet as belonging to '2017EclipseSolar2017' event correctly.

Figure 5.17: Example of another tweet classified in the 'getar-cs5604f17' table

Figure 5.18 provides an example of a tweet that is not related to the '2017EclipseSolar2017' event but does contain the hashtag for it. Our classifier catches this and is able to classify this tweet as not belonging to '2017EclipseSolar2017' event with a high probability.

Figure 5.18: Example of a tweet not belonging to the event classified in the 'getar-cs5604f17' table

Chapter 6

User Manual

6.1 Dependencies

Previous work was done on an older Cloudera CDH version. The cluster has been updated to CDH 5.12, and the current code has been proven to work on CDH 5.12 using updated dependencies. Some dependencies were updated due to security updates or Cloudera updates. There are two types of dependencies: compilation dependencies and runtime dependencies.

Compilation Dependencies:

- Python 2.6.6
- SBT 1.0.2
- Java 8 (ver 1.8.0_144-b01)

Run-Time Dependencies:

- Java 7 or later
- Spark 1.5.0 for Hadoop 2.6
- Stanford Core NLP English Model v3.8.0 .jar
- Stanford Core NLP Model v3.8.0 .jar
- HBase-RDD 2.11-0.8.0 .jar

The .jar run time dependencies must be put into the ./jarlib/ folder in the ISRProject folder. These .jar files are not included in the git repository as they are massive files, so they must be manually downloaded and moved to the correct file location upon

cloning a fresh copy of the repository. Note that the .sh bash files used to run the code must have the "-jars" flag arguments for the "spark-submit" command be up-to-date if the .jar model versions are updated.

The run-time dependencies should already be available on the class cluster, but may need to be installed on a local test machine. The code must be compiled on a local machine. This is because the class cluster does not have those compilation dependencies, and Java 8 is a fairly difficult upgrade to have done upon the class cluster. Of course, contact the class cluster administrators for available dependency information as upgrades to the class cluster may be done after the writing of this report.

6.2 Project File Structure

The project git repository is available on GitHub at:

https://github.com/madcat101010/ISRProject

This repository was forked from the previous team's GitHub repository. It is recommended for the next semester team to fork this repository to their own remote repository. The master branch contains the code and some useful data that was used in classifying on the class cluster this semester. The folder structure of the project folder is broken down in Table 6.1.

File Path	Purpose
./	Project Root Directory
./jarlib/	.Jar Dependencies Folder
./data/	Data To Be Pushed To HDFS, Existing Models Saved Here
./data/training/	Training Data Sets
./src/main/scala/	.scala Source Code
./oldLogs/	Logs generated by code execution
./target/	Compiled Files Generated by "sbt package" Command

Table 6.1: Project folder structure

6.3 Data Files

The ./data/ folder should be pushed to the Hadoop file system using "hadoop fs -put ./data/". This is required to run the code. There are also training and test data sets in text files in the ./data/training folder in the git repository. The training and test data sets include some .csv-formatted webpage data for shooting events, and some tweet data for shooting events and the solar eclipse. See Table 6.2 for the list of files in the ./data/ folder. Note that all models are directories, but the internal files of that model directory should not be modified by the user. Loading and saving models should be done using the .model directory name.

File Path (inside ./data/)	Purpose
2017EclipseSolar2017_tweet_lr.model/	Trained Logistic Regression model for 2017EclipseSolar2017 event tweets
2017EclipseSolar2017_webpage_lr.model/	Trained Logistic Regression model for 2017EclipseSolar2017 event webpages
2017ShootingLasVegas_webpage_lr.model/	Trained Logistic Regression model for 2017ShootingLasVegas event webpages
getar-cs5604f17_table_w2v.model	Word2Vec model trained on getar-cs5604f17 complete corpus as of 06 DEC 2017
training/2017EclipseSolar2017_tweet_training.csv	Hand labeled 2017EclipseSolar2017 event training tweets
training/2017EclipseSolar2017_webpage_training.csv	Hand labeled 2017EclipseSolar2017 event training webpages
training/2017ShootingLasVegas_webpage_training.csv	Hand labeled 2017ShootingLasVegas event training webpages

Table 6.2: Data files in the ./data/ directory

Some of the ./data/ files were generated by the classification code such as the trained models. Trained model metrics are logged for future reference as the model data itself doesn't store these metrics. The relevant log files that relate to the current getar-cs5604f17 table classification runs and trained models are in the ./oldLogs/ folder as shown in Table 6.3. There are other log files not listed in the table which are out-of-date and do not reflect any data in the table or provided trained models. The file names should be self-explanatory, and were kept as reference for older reports and presentations.

File Path (inside ./oldLogs/)	Purpose
log_class_eclipse_tweet.txt	Log of 2017EclipseSolar2017 tweet classification run
log_class_eclipse_webpage.txt	Log of 2017EclipseSolar2017 webpage classification run
log_class_vegasshooting_webpage.txt	Log of 2017ShootingLasVegas webpage classification run
log_train_eclipse_tweet.txt	Log of 2017EclipseSolar2017 tweet Logistic Regression model training
log_train_eclipse_webpage.txt	Log of 2017EclipseSolar2017 webpage Logistic Regression model training
log_train_vegasshooting_webpage.txt	Log of 2017ShootingLasVegas webpage Logistic Regression model training

Table 6.3: Useful log files in the ./oldLogs/ directory

6.4 Calling the Code

Bash shell scripts should be used to compile and run the code. In the root directory, there are many .sh bash scripts used to execute the code. Modify a copy or existing script to meet the configuration need of the execution. Existing bash scripts in the repository are shown in Table 6.4. The "sbt package" command is used to compile the code, and the compiled code is stored in a folder called "target" in the local directory. Two scripts are provided to scp the "target" directory with the compiled code from a virtual machine to a local machine and from a local machine to the class cluster. These may be modified if the current virtual machine, local machine, or class cluster ssh login information or file location is changed. A "spark-submit" command, contained within the bash script, is used to actually run the code. The single line call can be broken down into the following segments:

```
spark-submit
```

This is the command to execute the compiled Scala code.

```
--master local
```

This flag sets the local node as the master node. Only need this flag for local machine testing purposes.

```
--driver-memory 15G
```

This flag sets the driver or master node to allocate how much memory for this execution. Sufficient memory is required when processing data sets in large batches. Reduce this memory value when testing on local machines.

```
--jars
jarlib/stanford-english-corenlp-3.8.0-models.jar,
jarlib/stanford-corenlp-3.8.0.jar,
jarlib/hbase-rdd_2.11-0.8.0.jar
```

This specifies the runtime .jar dependency files required to run the compiled code. This does not need to be copied to the Hadoop file system, but must be available at the specified location on the node used to execute this command. This folder is not included in the provided code repository as the .jar files are very large. These dependencies can be downloaded online from their source, and the ./jarlib/ folder needs to be created to hold these .jar files, too. Note: there should be no new lines in between commas, but new lines are added to fit this onto the report page.

```
--class isr.project.SparkGrep target/scala-2.10/sparkgrep_2.10-1.0.jar
<train/classify/label> <webpage/tweet/w2v> <srcTableName>
<destTableName><event name> <collection name> <class1Name> <class2Name>
```

`[class3Name] [...]`

This specifies the project package name and location of the compiled Scala code. The flags in <> are required for execution. The flags in `[]` are optional. More than 2 classes can optionally be named for multi-class classification.

Bash Script File Name	Purpose
compiledVMtoLocal.sh	Move compiled code from a Cloudera VM to local machine
compiledLocaltoCC.sh	Move compiled code from a local machine to the class cluster
classifyEclipseWebpage.sh	Build and classify 2017EclipseSolar2017 webpages on class cluster
classifyEclipseTweet.sh	Build and classify 2017EclipseSolar2017 tweets on class cluster
labelSolarEclipseCluster.sh	Build and hand label 2017EclipseSolar2017 tweets on class cluster
trainEclipseTweet.sh	Build and train 2017EclipseSolar2017 tweet Logistic Regression model
trainEclipseWebpage.sh	Build and train 2017EclipseSolar2017 webpage Logistic Regression model
trainW2VModelGoogle.sh	Deprecated: Build and convert Google News Word2Vec to Spark Word2Vec
trainW2VModelTable.sh	Build and train Word2Vec model on class cluster

Table 6.4: Included bash scripts

Bash scripts can contain multiple `spark-submit` calls. For example, the classifyEclipseTweet.sh has three `spark-submit` calls to classify the 3 2017EclipseSolar2017 tweet collections, and more can be added to classify other events or collections. It is recommended to log the outputs of the execution for review later. This is done by commands such as: "`./myExecutingScript.sh > log_example.txt`".

Chapter 7

Developer Manual

7.1 File Naming Convention

Naming conventions in the data folder must be adhered to for correct execution of the classification framework. A common naming convention also allows easier maintenance of data files. This is especially important when future events and tables need to be classified. All training data must be named as:

`<eventName>_<tweet/webpage>_training.csv`

Models are automatically generated with the correct naming conventions. Logistic Regression models that are imported must be compatible with Spark and be named with the convention:

`<eventName>_<tweet/webpage>_lr.model`

Word2Vec models that are imported must be compatible with Spark and be named with the convention:

`<tableName>_table_w2v.model`

Note that all models should be directories with file parts and other directories within. The '2017EclipseSolar2017' training data and Logistic Regression models should be included in the repository. The '2017ShootingLasVegas' webpage training data and Logistic Regression model are also included. Finally, the 'getar-cs5604f17' table Word2Vec model is included.

7.2 Source Code Files

The source code is available at the GitHub repository and file location mentioned in Section 6.2. Table 7.1 shows the current Scala source code files in the repository.

Scala File Name	Code Functionality
ClassificationUtility.scala	Overridden Spark library functions. Overrides .predictPoint()
CleanTweet.scala	Some document cleaning functions
DataRetriever.scala	Read a batch of documents from a table, classify, and write classification.
DataWriter.scala	Writes document data to a table. Converts class label to class name
FpGenerate.scala	Deprecated: frequent feature generator
HBaseInteraction.scala	Basic functions for HBase table reading and writing
IdfFeatureGenerator.scala	Deprecated: IDF model feature generator
MetricsCalculator.scala	Test trained models and output performance metrics
SparkGrep.scala	Main function here with training and testing functions
Word2VecClassifier.scala	Word2Vec + Logistic Regression model training and class predicting

Table 7.1: Scala source code files.

Extending this code can be fairly confusing at first. The original code generated many objects with essentially static functions within them rather than classes. Therefore, care must be taken for handling object scope variables and values such as HBase column family names and column names.

The main function parses the input arguments, and calls the desired function specified in the input arguments. Any modifications to training and the overall input argument handling should be done in the 'SparkGrep.scala' file. Any modifications to the classify documents on an HBase table should be done in the 'DataRetriever.scala' file. To modify the Word2Vec and Logistic Regression class prediction and training, modification should be done in the 'ClassificationUtility.scala' and 'Word2VecClassifer.scala' files. Modifiation to the writing to HBase table behavior should be done in the 'DataWriter.scala' file.

In the future, it is recommended to better organize code in 'DataRetriever.scala'. The DataRetriever code should only handle HBase document retrieval, but it has been made to include calling the clean document, classify document, and write document class functions provided by the other .scala files.

7.3 HBase Table Scanning

Documentation on the Hadoop filter library usage can be confusing when using filters for HBase table scanning. This library is explained here for ease of usage to future developers.

A Scan object is configured to add a column value to read from each row. The Scan object can also be associated with Filter objects. It is recommended to use a FilterList object from the org.apache.hadoop.filter library to use multiple filters. Use the .addFilter() method to add filters to the filter list. The filter is set to FilterList.Operator.MUST_PASS_ALL to ensure that rows read back pass all of the filters rather than a single filter. Filter lists can be used hierarchically to AND or OR together filters. Use FilterList.Operator.MUST_PASS_ALL to AND together filters and FilterList.Operator.MUST_PASS_ONE to OR together filters. Let it be known that the Apache documentation on filters uses the term 'emit'. Rows emitted by a filter means that the filter will keep that row as a result in the scan. Also note that batching of the scan cannot be used when filter lists are used, but the .next(int numRows) call can still limit the number of rows read back in a batch. Ensure that this numRows does not exceed the scan caching, to maximize performance.

SingleColumnValueFilter objects are used for retrieving documents to classify in the DataRetriever.scala file. Note that the scan must scan for columns using the .addColumn() method first before the single column value filter can filter for that column value. The filter condition resulting in true will result in the row being kept as a result, whereas the filter condition returning false will omit (not emit) that row. Also, each single column value filter can be set to run the filter or not depending on if the column exists or not. This is done by the .setFilterIfMissing() method. Setting .setFilterIfMissing(true) will omit that row if the desired column for that row is missing. Setting .setFilterIfMissing(false) will keep that row if the desired column for that row is missing (by not running that filter for that row).

7.4 Hand Labeling Tweet Training data

To hand label the tweet data for our training process, please follow the steps provided below:

- Make sure that the class cluster has our team's most updated code.
- Log in to the class cluster and node00.

- Go to the ISRProject folder. Then, replicate and open the copy of the script 'labelSolarEclipseCluster.sh'.
- In the script, you can change the following to label a different collection or table:
  - 3rd argument is the main input HBase table.
  - 4th argument is the number of tweets you want to label.
  - 5th argument is the name of the output CSV file.
  - 6th argument is the name of the collection that you want to sample the tweets from.
  - 7th and 8th argument are the binary classes that you want to label.
  - Listing 7.1 shows the parameters we used to label 2017 eclipse tweets.

```
spark-submit --master local ... "label" "tweet" "getar-cs5604f17"
"500" "eclipse_labeleddata_107th" "#Eclipse" "SolarEclipse"
"NotSolarEclipse"

#This script gets 500 tweets from "#Eclipse" collection in the
#"getar-cs5604f17" table & write to "eclipse_labeleddata_107th" csv
#
#The script labels the tweets as "SolarEclipse" / "NotSolarEclipse"
```

Listing 7.1: YAML Usage Example For Hierarchical Framework

- Run the script and label tweets that the script asks you to. Figure 4.1 shows an example of what the user will see after running the hand labeling script.
- To label a tweet, type "1" for the tweet that is related to the collection, or type "2" otherwise.

Chapter 8

Timeline

Week	Starting date	Task	Performed by
1	Aug 29	Forming the CLA team	All Members
2	Sep 5	Reading the previous year's report	All Members
3	Sep 12	Going through the previous year's source code for tweets	All members
		Setting future plans and goals	Ngo
4	Sep 19	Setting up the local machine	All members
		Testing the source code	
5	Sep 26	Preparing the IR1 presentation and report	All Members
6	Oct 3	Preparing a prototype for webpage classification	Mulchandani, Naik, Patil
7	Oct 10	Further work on the webpage prototype	Azizi, Vezvaee
		Adapt and test tweet framework on webpage data	Ngo, Yang
		Working on TF-IDF idea	Mulchandani
8	Oct 17	Preparing the IR2 presentation and report	All Members
9	Oct 24	Further work on the webpage prototype	Azizi, Vezvaee
		Fix original tweet framework code	Ngo, Yang
		Doc2Vec implementation research	Mulchandani

10	Oct 31	Further work on the webpage prototype	Azizi, Vezvaee, Mulchandani, Naik, Patil
		Generate training data from HBase	Ngo
		Hand label solar eclipse tweets	Ngo, Yang
		Update tweet framework for HBase interaction	Yang
		Hand labeled the following six school shooting collections: '2010SchoolShootingUniversityAlabama', '2008SchoolShootingNIU', '2014SchoolShootingReynoldsHighSchool', '2012SchoolShootingSandyHook', '2015SchoolShootingUmpqua', '2009SchoolShootingDunbar'	Mulchandani, Naik, Patil
12	Nov 7	Further work on the webpage prototype	Azizi, Vezvaee, Mulchandani, Naik, Patil
		Increase tweet framework accuracy and handle edge cases	Yang
		Generate results for comparison of webpage classification models	Mulchandani, Naik, Patil
		Create a script to hand label tweets on the class cluster	Ngo
12	Nov 21	Thanksgiving Break	All Members
13	Nov 28	Reclassify documents classified as not belonging to collection	All Members
		Code and test webpage classification and integrate into framework	Yang
		Benchmark our classifier's predictions	Ngo
14	Dec 5	Final report and presentation	All Members
		Finalize source code git repository	Yang

Table 8.2: Timetable of tasks

Chapter 9

Conclusion

The classification framework has been updated to be capable of classifying both webpage documents and tweets from an HBase table on the class cluster following a system-level schema. Extra features such as hand labeling up-to-date tweet documents off the class cluster table also have been added to the classification framework. Training data for events that need to be classified have been hand labeled, and an improved file schema has been developed to better organize event classification data and to apply it to the desired collections. Finally, the Word2Vec model training process has been improved to generate a more complete vocabulary. This entire framework has been tested and used to classify some tweet and webpage collections for a few events on the class cluster HBase table.

The classification framework could use more improvements such as flagging documents that the Logistic Regression model cannot properly classify. This would allow these documents to be added to the training set to improve model accuracy. Also, more training data could be hand labeled to further improve the accuracy of the classification models. The current classification models are accurate enough for complete end to end processing for the GETAR project, but may not be accurate enough to satisfy end users.

We classified the '#Eclipse2017', '#solareclipse' and, '#Eclipse' tweet collections. These comprised of about 1,562,215 tweets. We also classified the 'Eclipse2017', '#August21','#eclipseglasees','oreclipse' and, 'VegasShooting' webpage collections. These had 3,454 solar eclipse event webpages and 912 Vegas shooting webpages.

While obtaining results during this classification process, we found a few things that could be implemented in a better way. These ideas could help serve GETAR's purpose in a better way. These are explained in Chapter 10 below.

Chapter 10

Future Work

10.1 Hierarchical Framework

In the future, we plan to compare a hierarchical classification model with the existing linear regression classifier model. After Word2Vec generates features from raw text, the hierarchical framework will classify it into general categories. Once a set of features is classified into a general category, the framework will spin off a thread that will classify the feature set into the more specific subcategories of the general identified category. The whole process will continue until it reaches a specific event.

Some advantages of this include:

- The framework is scalable, meaning the framework will still be able to run with a larger set of classes without compromising on the accuracy or performance.
- The training process can be accelerated by the use of multi-threads.

A disadvantage of this framework would be its runtime during training and predicting. In the worst case, when every row in a table is related to the same class, this framework will lead to a sequential runtime to categorize the event, meaning all threads that lead to a subclass will wait for each other to finish classifying.

To address the problem, our team proposes to benchmark all the classifier models on the existing framework to choose which classifier will match with the specifications listed below:

- The classifier has to train very quickly for small categories as it does not have to predict for a big set of classes; the hierarchical framework will break down the classes into more categories.

- Because of this, it will not need to have a perfect or very high accuracy as long as it gives us a low prediction error.

10.2 Cron task and YAML Configuration File

A cron task will be needed to detect the need of retraining models for new classes. A YAML file is very helpful to map 'Classification ID values to Classification-labels'. Also, it is a good idea to provide an API to add a new class to the YAML configuration file. Classification ID values are used internally by our code to quickly represent an event class string. Whenever new event classes are added to the YAML file, once the training task is executed again, it will detect the new event classes and retrain the models. Potentially, we would like to retrain the models periodically, keeping the model trained with both the old and newly classified data.

A YAML configuration file will need to include configurations for the hierarchical framework. The YAML file can be used to create a tree representation of general topics and their sub-topics and it does not limit users to have a fixed depth for all the categories. Listing 10.1 proposes the potential use of the YAML configuration file for the new hierarchical framework.

```
#Mapping of classification events to classification IDs
overallCategories:
        - naturalDisasters
        - shootingEvents
naturalDisasters:
        - hurricane
        - earthquake
hurricane:
        - Sandy
        - Harvey
earthquake:
        - ...
shootingEvents:
        - Las_Vegas

#This config file will generate the following:
config = yaml.load()
print config["overallCategories"]
#This returns: ["naturalDisasters", "shootingEvents"]

print config["hurricane"]
#This returns: ["Sandy", "Harvey"]
```

Listing 10.1: YAML Usage Example For Hierarchical Framework

By using this format, we can traverse through the topics, from the top categories to a specific event, to find every category that leads to the event.

10.3 Hand Labeling Process

There is some more work that could be done to the hand labeling process to improve the accuracy of prediction. One of the improvements that could be made is sampling randomly from the main HBase table, rather than from the top of the table. Our team would also suggest to sample across multiple collection names of the same real world event. Lastly, the similar hand labeling process can be replicated for webpages to provide convenience.

10.4 Overriding Spark Library Word2Vec

The Google News pre-trained Word2Vec model has a 300 billion word vocabulary. Some peers have reported that there are repeats of words in this model, but it is a high word count regardless. The Spark Word2Vec models are not compatible with this Google Word2Vec binary model, so a conversion function in SparkGrep.scala called loadGoogleW2vBin() was written. However, Scala is based on Java, and the Spark Scala implementation of the Word2Vec model uses a Java array to index the model's vocabulary. This Java array is indexed with 32-bit integers which cannot fit a 300 billion word vocabulary. One method to overcome this issue is to override the Spark library's Word2Vec.scala code. The Spark library Word2Vec source code can be found at:

https://github.com/apache/spark/blob/master/mllib/src/main/scala/org/apache/spark/mllib/feature/Word2Vec.scala

Note that this type of override operation has been done before as seen in the ClassificationUtility.scala file for the Logistic Regression predictPoint() method. Another possible way is to redefine the token from words into other formats. Finally, the least frequent words could be trimmed off the 300 billion vocabulary to save memory space as it isn't uncommon to have the Word2Vec model ignore very uncommon words.

Another override feature that would be desirable is to modify the training process of a new Spark Word2Vec model. As previously mentioned, Word2Vec model training cannot be done iteratively. This is disadvantageous as training is very time consuming, and prevents simply re-using the older model and further training on new

incoming documents. Instead, the model must be re-training on the entire corpus, old and new documents, again. An override to the training functions should remove the internal variable re-initializations upon calling the training function. This was what prevented iterative training in the first place.

10.5 More Cleaned Text for Tweet Documents

A requirement determined this semester was that the collection management teams must complete the entire text cleaning process on their end. This semester, the text of tweet and webpage raw data was cleaned and put into its dedicated clean text columns. The text from webpages are generally substantial enough to not warrant reading text from other data columns, but for tweets, data from other columns had to be scanned and combined with the provided clean text.

The reason this was done was due to the tweet cleaning process removing hashtags, URLs, and mentions from the cleaned tweet text. These were put into other clean-tweet data columns such as the "clean-tweet:hashtags" or "clean-tweet:long-url" column. These columns were not cleaned of symbols such as the "#" symbol or URL syntaxing symbols. However, these columns contain desirable data. The reason that these columns were not fully cleaned is due to their usage by other processes. Therefore, in the future work, it would be ideal for a clean-tweet column that cleans and combines all of these columns for the CLA team to use. Otherwise, a cleaned text column for each of these data columns would also be useful, and the classification process will then combine them with the existing clean text column.

10.6 Webpage Classifier Performance

As previously mentioned in Section 5.5, the webpage classifier models were trained using entire document collections of the event being classified. Therefore, it was not feasible to test the performance of the models outside of what was tested via the testing set. In the future, when more webpage data is collected that is outside the training set, it is needed to evaluate the webpage classifier performance on the new data.

Chapter 11

Acknowledgements

We would like to acknowledge and thank the following for assisting and supporting us in this project.

- Dr. Edward Fox
- NSF grant IIS - 1619028, III: Small: Collaborative Research: Global Event and Trend Archive Research (GETAR)
- Digital Library Research Laboratory
- Graduate Teaching Assistant - Liuqing Li
- Hithesh Peddamekala for providing the ontology of events
- All teams in the Fall 2017 class for CS 5604

Bibliography

[1] Saurabh Chakravarty and Eric Williamson, "Final Project Report - CS 5604 Information Storage and Retrieval CLA Team, Fall 2016, Virginia Tech, Blacksburg, VA 24061".
`https://vtechworks.lib.vt.edu/handle/10919/73713`, [Accessed 1 Dec. 2017]

[2] Christopher D. Manning, Prabhakar Raghavan, Hinrich Schutze, "Introduction to information retrieval", Cambridge University Press, 2008.

[3] Jey Han Lau and Timothy Baldwin, "An empirical evaluation of doc2Vec with practical insights into document embedding generation", arXiv:1607.05368v1, 2016.

[4] Kaggle: "Large Scale Hierarchical Text Classification", 2015.
`https://www.kaggle.com/c/lshtc` [Accessed 12 Sept. 2017]

[5] Ioannis Partalas, Aris Kosmopoulos, Nicolas Baskiotis, Thierry Artieres, George Paliouras, Eric Gaussier, Ion Androutsopoulos, Massih-Reza Amini, and Patrick Galinari, "LSHTC: A Benchmark for Large-Scale Text Classification", arXiv:1503.08581v1, 2015.

[6] Pedregosa et al., Scikit-learn: Machine Learning in Python, JMLR 12, pp. 2825-2830, 2011.
`http://scikit-learn.org/stable/modules/model_evaluation.html`, [Accessed 18 Sept. 2017]

[7] Yu-Yu Chou and Linda G. Shapiro, "A hierarchical multiple classifier learning algorithm", Pattern Analysis and Applications, vol. 6, **150-168**, 2003.

[8] Saurabh Chakravarty, "Master Of Science in Computer Science and Applications Thesis: A Large Collection Learning Optimizer Framework ", Virginia Tech, Blacksburg, VA 24061, 2017.
`http://hdl.handle.net/10919/78302` [Accessed 1 Sept. 2017]