CS 5604 Information Storage and Retrieval
Spring 2016
Interim Report 4

Text Classification

Team Members:

Hossameldin Shahin <hshahin@vt.edu>
Matthew Bock <mattb93@vt.edu>
Michael Cantrell <mcantrell@vt.edu>

Project Advisor:

Prof. Edward A. Fox

4/20/2016
Virginia Tech, Blacksburg

Abstract

In the grand scheme of a large Information Retrieval project, the work of our team will be that of performing text classification on both the tweet collections and their associated webpages. In order to accomplish this task, we will seek to complete three primary goals. We will begin by performing research to determine the best way to extract information that will be used to represent a given document. Following that, we will work to determine the best method to select features and then construct feature vectors. Our final goal is to use the information gathered previously to build an effective way to classify each document in the tweet and webpage collections. We intend for these classifiers to be built with consideration of the ontology developed for the IDEAL project.

The team assigned to perform this classification work last year researched various methods and tools that could be useful in accomplishing the goals we have set forth. Last year's team developed a system that was able to accomplish similar goals to those we have set forth with a promising degree of success. Our goal for this year is to improve upon their successes using new technologies such as Apache Spark. It is our hope that this will allow us to build a more optimized system capable of working with larger collections of tweets and webpages.

We also have several other goals for this semester's work in addition to improving on prior classification work. Two of those main goals include handling incremental data updates and performing evaluations as to the effectiveness of our classification system. We have designed our system with both of these in mind. Our approach to both of these goals are described in detail in this paper.

Should add about: incremental, running experiments, evaluations.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The classification team's goal is to take collections of tweets and webpages and classify them based on their relevance to given classes or topics. Our team fits into the grand scheme of the project by working between the Collection Management team and the Solr team. The Collection Management team will be responsible for taking the raw tweet and webpage data and filtering out any obvious spam, vulgarity, or otherwise unreadable and unwanted content. The Collection Management team will write the cleaned data into a table in our HBase instance. Once this is done, we will attempt to classify each document's relevance to a specific category. We will explore several methods for accomplishing this, starting with the methodology laid out by last year's Classification team [**cui2015classification**]. Once we have classified the data, we will be able to pass it along to the Solr team by writing the newly classified data back into the primary HBase table as a column family. The Solr team will then be able to use the column family in the indexing of all the tweet and webpage data. Solr will then provide the indexes for the data to the Front End team, allowing anyone to make use to the system we are creating.

This paper represents the fourth interim report from the Classification team working as a part of the larger scale project for CS5604 Information Storage and Retrieval. The current state of the work is a partial draft of the final report; therefore expect further additions and details in some chapters as the course progresses.

We begin by documenting our understanding of some of the pertinent literature, namely the course textbook and the Classification team report from last year; this can be found in Chapter 2, our literature review. Chapter 3 is the primary section of the document and includes our discussion of the project requirements, an outline for our design for the classification portion of the larger project, and finally a breakdown of our current progress and future plans for the text classification project.

These chapters are then followed by a User Manual in Chapter 4, where we will discuss details in which a user of our methods and programs would have interest. We then include a separate Developer Manual in Chapter 5, which will document and include details of the code base so that it might be extended and leveraged by other developers.

Following this we include our conclusions about the state of the project thus far and present our thoughts for future work in Chapter 8.

Chapter 2

Literature Review

2.1 Textbook

The textbook [**manning2008introduction**] introduces the classification problem. That is, given a set of classes, the goal is to determine what subset of those classes a document may belong to. To that end, the textbook describes a number of methodologies for selecting features. These features are then used by one of the classification methods discussed. The primary methods discussed are Naïve Bayes, Vector Space Classification, and the Support Vector Machine.

2.2 Papers

The primary paper that has been used as a reference is the final report of the Classification team from last year [**cui2015classification**]. We have read through this report to understand the progress that the Classification team made last year as well as using the paper to gain an understanding of the task and interactions of the systems that we are using to perform our work. At a high level, the paper described a methodology employed by the team in which they were able to apply the Naïve Bayes method to classify sets of data. The team used Apache Mahout machine learning technology to generate Naïve Bayes classifiers to make predictions for new data. The biggest difference from last year's work to this year's work is that we will be primarily using Apache Spark, and so while we will be able to reference their work, we are using entirely different technologies and will need to modify our approach accordingly.

Initially, we attempted an approach where we would issue queries to Solr to build a set of training data for a classifier. However, we were informed by the GRAs that this approach would not work because the Solr API was not exposed on the cluster, and so we would not be able to access it. When we started looking into new approaches, we were advised to look at Frequent Pattern Mining (FPM). We have looked at a paper by Han et al. [**han2000mining**], which presents a novel frequent pattern tree (FP-tree) structure and FP-tree based mining method called FP-growth. This method allows for mining the complete set of frequent patterns by pattern fragment growth. In this paper they also demonstrate that their new method is an order of magnitude faster than the typical apriori algorithm. The paper goes on to compare the performance against other popular data mining algorithms and discusses the use of the algorithm on some large industrial databases.

We are currently making use of the FP-growth algorithm and data structure in our working classification prototype.

Chapter 3

Requirements, Design, And Implementation

3.1 Requirements

Based upon group discussions in a classroom setting, it was decided what information that the Collection Management team would provide to the other teams that need to work with cleaned tweet data. A full specification of these decisions can be found by viewing the Collection Management team's report; however we will briefly discuss the points that were relevant to us.

Given the cleaned tweet data from Collection Management, our team will be able to perform the methodologies we describe later to classify whether a document is relevant or non-relevant to a given class. A detailed layout of the project with our interactions highlighted is provided by Figure 3.1.

We will then place our classification information in the HBase datastore to mark the relevance of each document. The Solr team will index this class information that we provide, allowing for more robust query results and more useful facets to be created.

3.2 Design

Our current design is based primarily off of recommendations from the GRAs assisting the class. We have also taken substantial input from last year's Classification team [**cui2015classification**] and the Professor.

We have designed our current solution around pulling training data from and testing on a small collection of around 100,000 tweets. This was originally going to be performed on the small collection that was assigned to our team, `#germanwings`. However, due to some changes and discussion among the other teams, we have decided to continue with designing and testing our solution using a cleaned dataset provided by the Collection Management team. Unfortunately, this dataset was not made available until after our current solution was mostly implemented using our original small dataset. Therefore for the rest of this document we will be using the small dataset `z_602`, `#germanwings`. All future work will be done using cleaned data from the Collection Management team.

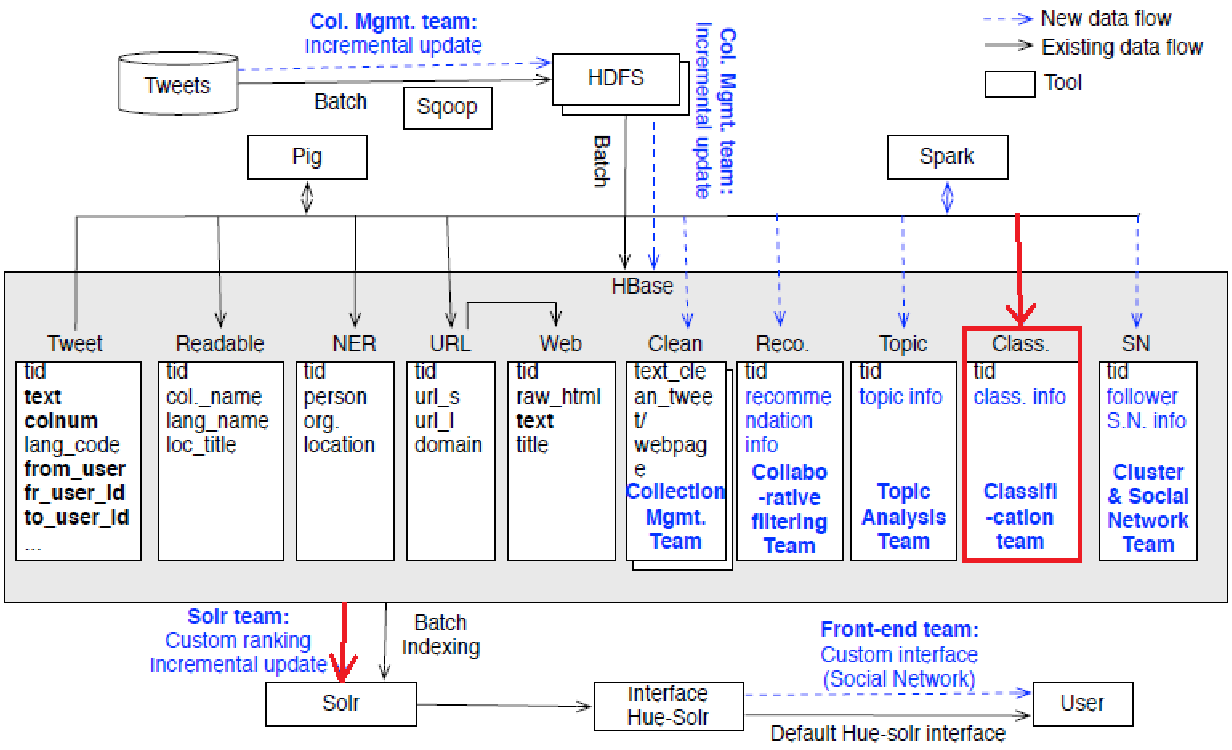


Figure 3.1: Layout of the project teams, as provided by Professor Fox and his GRAs.

3.3 Implementation

Our implementation can be easily laid out in a step by step workflow, with only one step requiring true manual input and evaluation from a person. Figure 3.2 illustrates the general flow of data that we have implemented thus far. Below we will discuss each step in detail.

Our methodology primarily revolves around building a training set. This training set will be used for training a machine learning model, a.k.a. a classifier.

3.3.1 Environment Set–Up

Our group decided to avoid working directly on the cluster, granting us full administrative rights to a machine, which allowed us to set up and test different configurations and algorithms beyond what might currently be supported by the Hadoop cluster being used for class.

A prime example, and the primary motivating factor for our choice, was our decision to use Frequent Pattern Mining (FPM) in the construction of our training sets. The cluster provided for the class was running Spark version 1.2.0, but FPM is not supported by Spark until version 1.5.0. Since FPM was a methodology suggested by the GRAs, and they assured us that they could upgrade the cluster to a more current version of Spark, we decided to proceed with this method.

Due to time constraints, we created a Debian Virtual Machine with sufficient specs to begin writing and testing our methodology. This machine is hosted on a VMware vSphere cluster that belongs to the Virginia

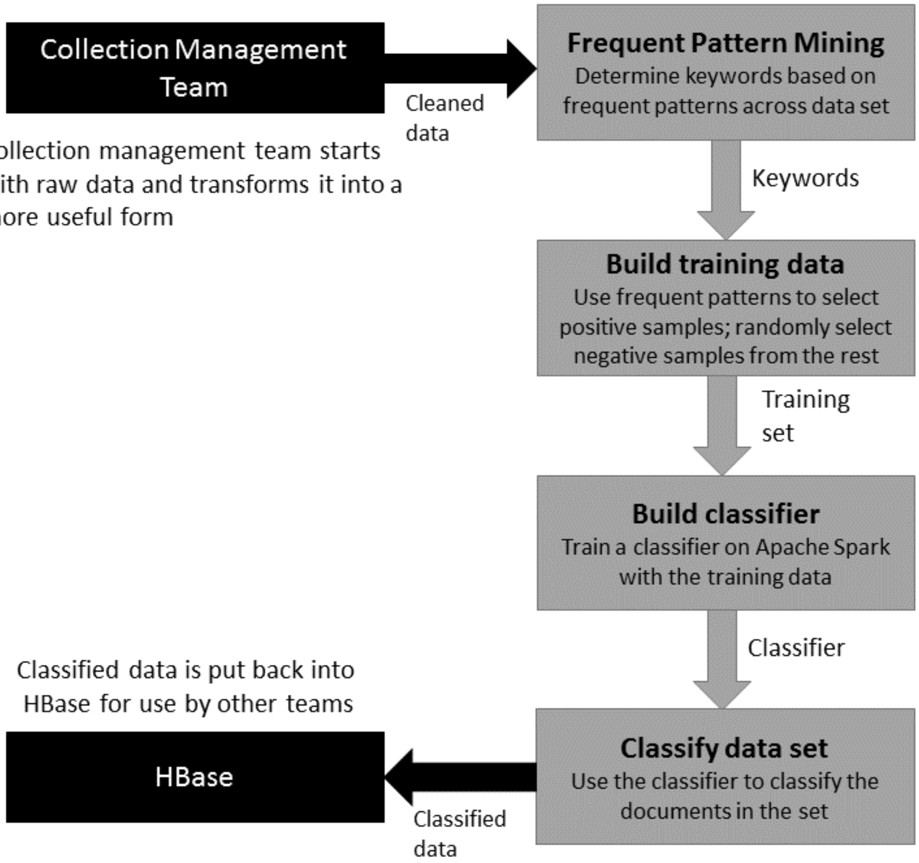

Collection Management Team
Cleaned data
Collection management team starts with raw data and transforms it into a more useful form
Frequent Pattern Mining
Determine keywords based on frequent patterns across data set
Keywords
Build training data
Use frequent patterns to select positive samples; randomly select negative samples from the rest
Training set
Build classifier
Train a classifier on Apache Spark with the training data
Classifier
Classified data is put back into HBase for use by other teams
HBase
Classified data
Classify data set
Use the classifier to classify the documents in the set

Figure 3.2: High level overview of the flow of data through the classification system.

Tech IT Security Office/Lab. Since one of the group members works in this lab, we were able to provision this VM. It has been assigned 32 GB of RAM, 16 processor cores, and 40 GB of storage. This has proven to be more than adequate for the testing that we have performed on our small data set.

The Hadoop cluster has since been upgraded to Spark 1.5.0 allowing us to transition our code and methods over and work directly with the HBase instance. As this was a recent change, we are still in the process of this transition and cannot report on what effect it will have, i.e. whether there are any issues encountered that break our implementation.

3.3.2 Building the Training Data

In order to begin the classification process we have to prepare a set of training data to use for the machine learning classification algorithm. In order to do this, we assume that we are working with data that has been cleaned of profanity and non-printable characters.

For our methodology we then need to take the content of each tweet and webpage as a string and tokenize it. This means that we will remove any duplicate words and have the result be a set of the vocabulary in the tweet (or webpage). At this stage we also remove any stop words that are in the vocabulary to make sure our algorithms in the next phase are not skewed by taking stop words into account. During this phase we also strip the # from any hashtags that are present in the tweet.

In our initial design and testing we did not yet have access to cleaned tweets and webpages from the Collection Management team, so we worked primarily to build the methodology that would be used once those resources became available. Therefore, some of the steps mentioned previously, such as the removal of the stop words and the removal of the # from hashtags are unnecessary when working with the data provided by Collection Management in HBase.

The next step in our algorithm involves the use of Frequent Pattern Mining (FPM) to determine the most frequently used patterns of words in the text of a series of tweets or webpages. FPM looks at the set of existing vocabulary for each text and determines which tokens appear together most often within a tweet's vocabulary.

This is the stage where manual intervention in necessary. We look at a file containing a sorted list of all the frequent patterns; from this we choose a set of features to train on. In our attempts at training a classifier for our small data collection, we chose to select a frequent pattern of four words. This was essentially an arbitrary choice, but we believe that it strikes a good balance between being too specific and missing some relevant tweets and being too broad and pulling in a number of non-relevant tweets.

At this stage we need to create positive and negative training samples. To do this, we pull a random subset of tweets that contain our frequent pattern and classify those as positive samples. For our negative sample we pull a random subset of tweets that do not contain our frequent pattern. To ensure that this labeling is at least mostly correct, we will inspect both the positive and negative training sets (or a portion of them) to confirm that each set is composed of either relevant tweets and webpages or non-relevant ones as appropriate. If in this inspection we find that the choice of frequent pattern has generated a poor training set, we will choose a different frequent pattern and go through the process again.

How long does one iteration take? How big are the training sets? How many iterations are commonly needed?
Add details somewhere.

3.3.3 Training the Classifier

We then use the sets of positive and negative samples to train a classifier. We are currently using a logistic regression classifier in our implementation, primarily due to its ease of implementation in Spark.

FPM allows us to develop a training set by filtering the tweets and webpages down to some subset of those that do and do not contain the selected frequent patterns. We take these subsets as the positive training data and the negative training data.

At this point we then feed the selected documents into the classifier; this step uses all the words in each document of the training data, not just the words used for FPM. As we progress we intend to try using several different classifiers to determine which one provides us with the best results.

Remember to give details, along with tables and examples, so your work can be easily reproduced

3.3.4 Predicting the Class

After training the classifier, we apply the classifier to all the tweets across our small data collection. This results in a binary scoring for each tweet as relevant or non-relevant to the collection based upon the training data we selected.

3.3.5 Evaluating the Classifier

We can evaluate the accuracy of our model by judging how well it classifies some of the data that could have been in our positive or negative samples. This is the most intuitive evaluation however it requires a great deal of manual effort to perform. We need to pull out a sampling of classified data and look through it manually, marking whether or not the classification was correct.

Another evaluation that we can perform looks at how well each small collection does at being relevant to the topic at hand. If a large number of the documents in the collection are classified as non-relevant, then the collection as a whole has not done a good job capturing the event in question.

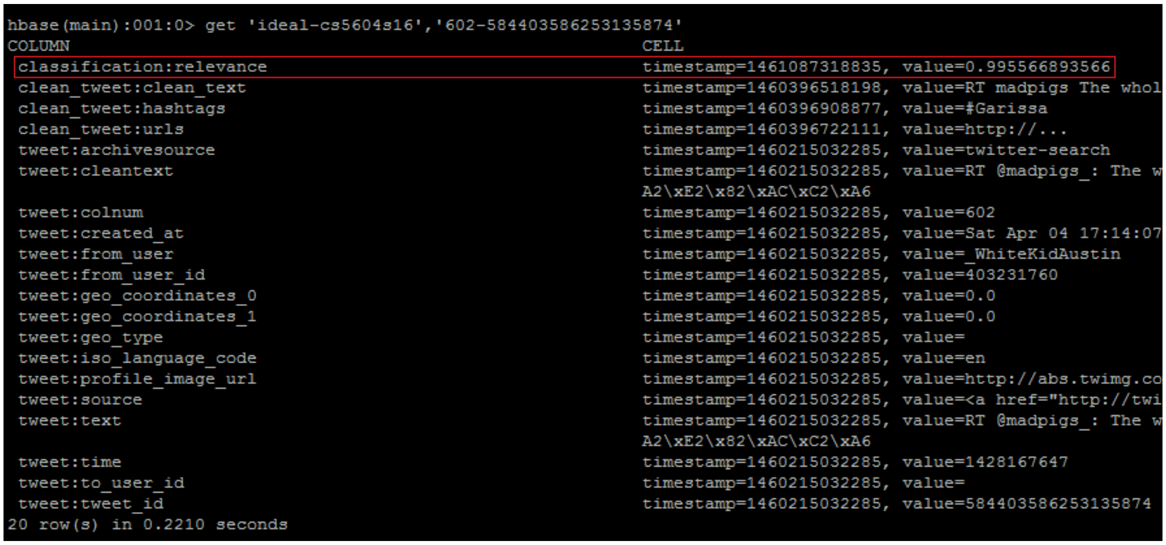
Finally, we need to perform an evaluation of how well our method of using FPM to determine the training documents works. This is much like the first evaluation in that it can be very manual. We will need to dump the generated training sets out to files and then manually decide whether a given set has all (or primarily) relevant or non-relevant documents, as appropriate.

3.3.6 Interfacing with HBase

Manual File Uploading

We are currently able to process the six small data files and determine a probability representing how likely it is that a specific tweet is relevant to the collection. For each of the data sets we have produced a .tsv file with two columns: one containing the tweet ID (which serves as the row key in HBase), and one containing a value between 0.0 and 1.0 which represents the probability that the tweet is relevant to the category.

Within HBase, we have created a column family named "classification." Within our column family, we only have need of one column, named "relevancy." This column is where the probability value for each document in the database will be stored for access by other teams. Figure 3.3 shows an example query for a row, which returns all of the data in that row. The data produced by our system is highlighted.



```
hbase(main):001:0> get 'ideal-cs5604s16','602-584403586253135874'
COLUMN                                                       CELL
 classification:relevance                                    timestamp=1461087318835, value=0.995566893566
 clean_tweet:clean_text                                      timestamp=1460396518198, value=RT madpigs The whol
 clean_tweet:hashtags                                        timestamp=1460396908877, value=#Garissa
 clean_tweet:urls                                            timestamp=1460396722111, value=http://...
 tweet:archivesource                                         timestamp=1460215032285, value=twitter-search
 tweet:cleantext                                             timestamp=1460215032285, value=RT @madpigs_: The w
                                                             A2\xE2\x82\xAC\xC2\xA6

 tweet:colnum                                                timestamp=1460215032285, value=602
 tweet:created_at                                            timestamp=1460215032285, value=Sat Apr 04 17:14:07
 tweet:from_user                                             timestamp=1460215032285, value=_WhiteKidAustin
 tweet:from_user_id                                          timestamp=1460215032285, value=403231760
 tweet:geo_coordinates_0                                     timestamp=1460215032285, value=0.0
 tweet:geo_coordinates_1                                     timestamp=1460215032285, value=0.0
 tweet:geo_type                                              timestamp=1460215032285, value=
 tweet:iso_language_code                                     timestamp=1460215032285, value=en
 tweet:profile_image_url                                     timestamp=1460215032285, value=http://abs.twimg.co
 tweet:source                                                timestamp=1460215032285, value=<a href="http://twi
 tweet:text                                                  timestamp=1460215032285, value=RT @madpigs_: The w
                                                             A2\xE2\x82\xAC\xC2\xA6

 tweet:time                                                  timestamp=1460215032285, value=1428167647
 tweet:to_user_id                                            timestamp=1460215032285, value=
 tweet:tweet_id                                              timestamp=1460215032285, value=584403586253135874
20 row(s) in 0.2210 seconds
```

Figure 3.3: An example row in HBase. Our data is highlighted.

For this version of the system, we are inserting the data into HBase manually. We accomplish this by using Hadoop's importTsv script as explained in the tutorial provided by the Solr team. The importTsv script is provided by HBase and allows you to take a .tsv file and specify what the columns in the file contain. We accomplish this with the following command:

```
$ hbase org.apache.hadoop.hbase.mapreduce.ImportTsv \
  -Dimporttsv.columns=HBASE_ROW_KEY,classification:relevance \
  ideal-cs5604s16 DATA_FILE
```

Here you can see we specify in the "-Dimporttsv.columns=" parameter that the first column in our .tsv file corresponds to the row key and the second corresponds to the classification:relevance column. We also specify that we are writing to the "ideal-cs5604s16" table and give it the file to upload.

Direct Reading and Writing

Using the importTsv script has allowed us to work on our system in our own virtual environment, since we are able to produce output files that can then be manually uploaded to the cluster. This was very useful while we were waiting for the software on the cluster to be updated so that we would have access to necessary libraries. Now that the software on the cluster has been updated, we are moving our implementation on to the cluster and making it so that our classification system will directly read data from and write data to the database.

Directly reading and writing from the database has several advantages. The main advantage and the primary motivation for doing it this way is that it will allow us to automate the process. Our system will be able to work directly with the HBase table, and will not need someone to manually feed it data files as input and handle the files it produces as output. This will greatly simplify the classification process for any future data that gets added to the database.

Directly reading and writing from HBase is done via the pyspark library provided by Spark. The library allows python to establish a connection to HBase which it can use to stream data back and forth. We currently have a rudimentary "hello world" version of this process working on our local virtual environment. We are working towards moving the implementation to the cluster and adapting it to read and write from the "ideal-cs5604s16" table. This functionality will be complete by the final report.

Chapter 4

User Manual

4.1 Installation Requirements

In this documentation, we make the assumption that you have already installed at least Spark version 1.5.0 and have it configured properly. If you have not performed this step, you can find installation and setup instructions at `http://spark.apache.org/downloads.html`.

The next step will be to clone the Git repository of our team. This can be found hosted on Github at: `https://github.com/hosamshahin/Spring2016_IR_Project.git`

Finally, you will need to extract the `small_data.zip` file that can be found in the `Spring2016_IR_Project/data` directory

4.2 Preparing the Server

In order for some of the Machine Learning libraries in Spark to work, you need to ensure that you have `libgfortran3` installed.

On a Debian based system, you can run the following command:

```
$ sudo apt-get install libgfortran3
```

Next install Anaconda which is a completely free Python distribution. It includes more than 400 of the most popular Python packages for science, math, engineering, and data analysis. For complete instructions on how to download and install Anaconda go to:

`https://www.continuum.io/downloads`

Anaconda is shipped with IPython notebook, a powerful interactive shell for Python.

To set up Spark and IPython to run together and provide a nice web interface, there are a number of things that need to be done.

Begin by creating an IPython profile for PySpark. This can be done using the command:

```
$ ipython profile create pyspark
```

If you are running other IPython profiles, you may want to change the default port. This can be done by modifying `~/.ipython/profile_pyspark/ipython_notebook_config.py`.

You can then change the default port within that file.

```
c = get_config()
# Simply find this line and change the port value
c.NotebookApp.port = <your port number>
```

Next the following lines need to be placed in `.bashrc`.

```
export PYTHONPATH=$SPARK_HOME/python:$PYTHONPATH
export PYTHONPATH=$SPARK_HOME/python/lib/py4j-0.8.2.1-src.zip:$PYTHONPATH
```

After this the `.bashrc` file should be reloaded by relogging in, or sourcing the `.bashrc` file.

To run the IPython server you should run the following command from within the cloned git repository, replacing <IP> with the IP address of your server.

```
ipython notebook --profile=pyspark --no-browser --ip <IP>
```

You can then navigate to <IP>:<PORT> in your browser to get access to the IPython notebook. By default the port will be 8888.

4.3 Using the IPython Notebook

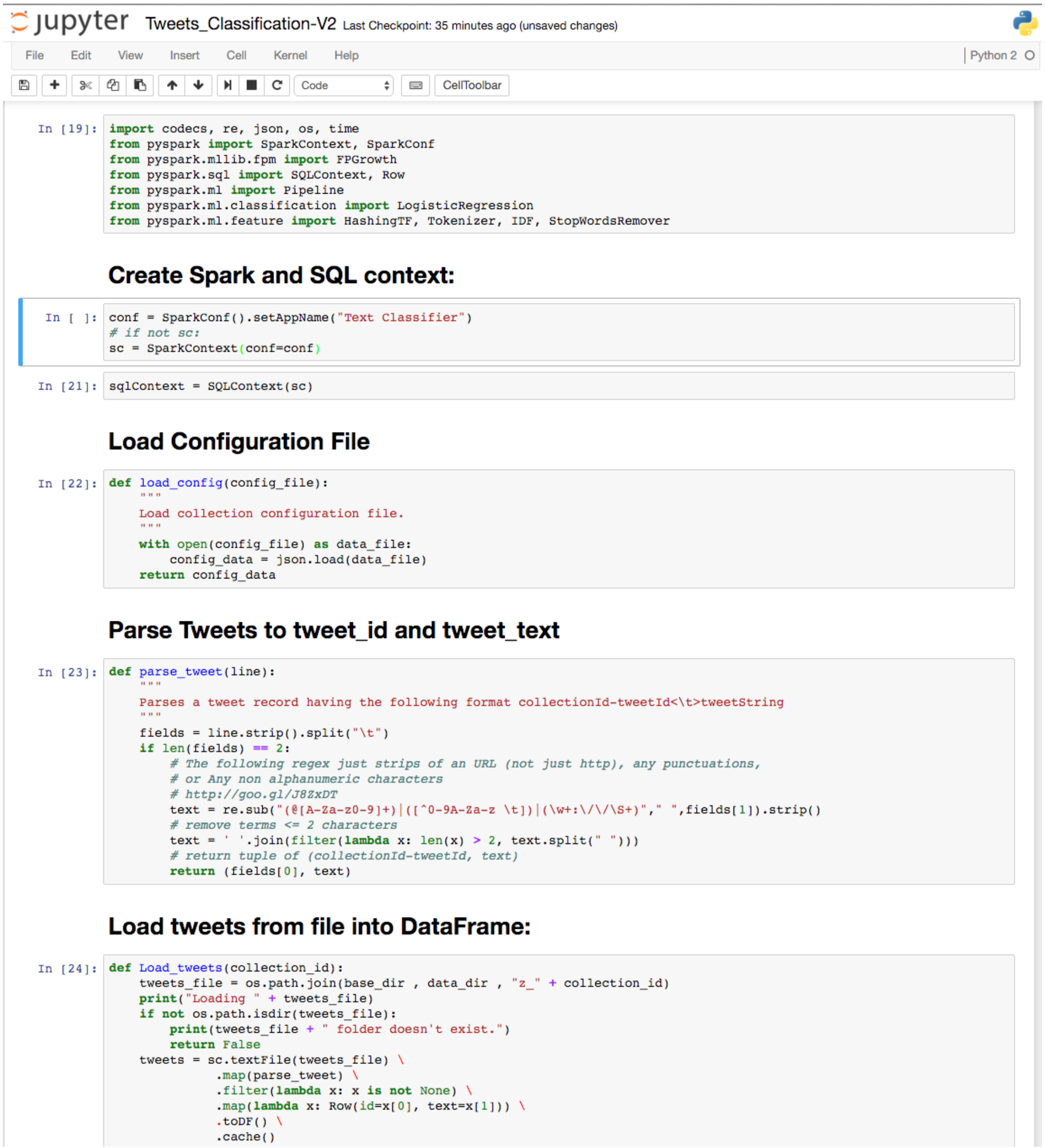
Once the IPython notebook has been opened in the browser, open the `Tweets_Classification-V2.ipynb` IPython notebook. You should see something similar to Figure 4.1.

In order to use the script, you will need to modify the `base_dir` variable in cell 10. Set this to point at the `data` directory.

4.3.1 Using the Configuration File

To go further in using the script, you may need to make modifications to the `collections_config.json` file under `data`; see Figure 4.3. The provided configuration file is already configured to work with the six small datasets that were assigned to the groups at the start of the semester. However, the frequent patterns (FP) field has not been properly configured. This will need to be done for each collection as you work through the tutorial.

To classify your data, you need to specify the table ID of your collection in the `collections_config.json` configuration file. You should model your entry off of one of the sample entries in the file, or modify a sample directly. It is also suggested that you update the name field to the name of your collection. At the moment, don't worry about the FP values.


jupyter
Tweets_Classification-V2 Last Checkpoint: 35 minutes ago (unsaved changes)
File
Edit
View
Insert
Cell
Kernel
Help
Python 2
Code
CellToolbar

In [19]:

```python
import codecs, re, json, os, time
from pyspark import SparkContext, SparkConf
from pyspark.mllib.fpm import FPGrowth
from pyspark.sql import SQLContext, Row
from pyspark.ml import Pipeline
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import HashingTF, Tokenizer, IDF, StopWordsRemover
```

Create Spark and SQL context:

In [ ]:

```python
conf = SparkConf().setAppName("Text Classifier")
# if not sc:
sc = SparkContext(conf=conf)
```

In [21]:

```python
sqlContext = SQLContext(sc)
```

Load Configuration File

In [22]:

```python
def load_config(config_file):
    """
    Load collection configuration file.
    """
    with open(config_file) as data_file:
        config_data = json.load(data_file)
    return config_data
```

Parse Tweets to tweet_id and tweet_text

In [23]:

```python
def parse_tweet(line):
    """
    Parses a tweet record having the following format collectionId-tweetId<\t>tweetString
    """
    fields = line.strip().split("\t")
    if len(fields) == 2:
        # The following regex just strips of an URL (not just http), any punctuations,
        # or Any non alphanumeric characters
        # http://goo.gl/J8ZxDT
        text = re.sub("(@[A-Za-z0-9]+)|([^0-9A-Za-z \t])|(\w+:\/\/\S+)"," ",fields[1]).strip()
        # remove terms <= 2 characters
        text = ' '.join(filter(lambda x: len(x) > 2, text.split(" ")))
        # return tuple of (collectionId-tweetId, text)
        return (fields[0], text)
```

Load tweets from file into DataFrame:

In [24]:

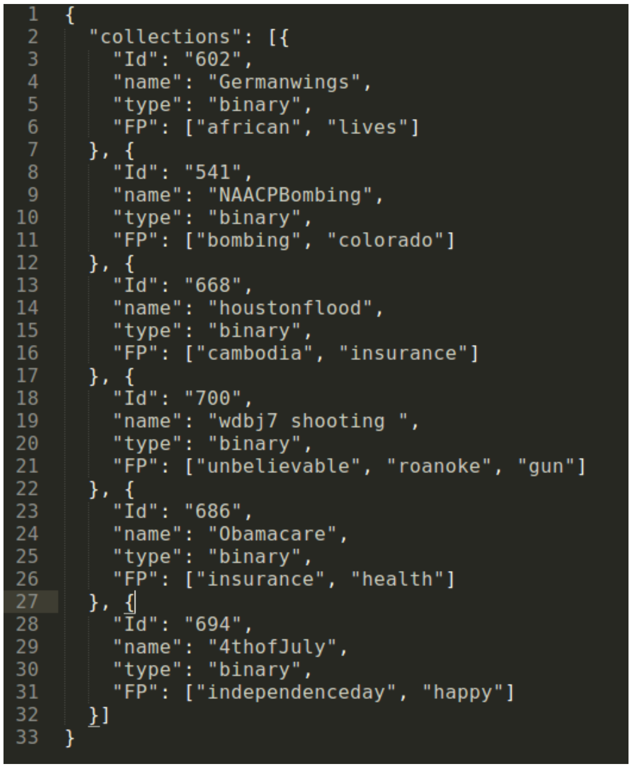
```python
def Load_tweets(collection_id):
    tweets_file = os.path.join(base_dir , data_dir , "z_" + collection_id)
    print("Loading " + tweets_file)
    if not os.path.isdir(tweets_file):
        print(tweets_file + " folder doesn't exist.")
        return False
    tweets = sc.textFile(tweets_file) \
            .map(parse_tweet) \
            .filter(lambda x: x is not None) \
            .map(lambda x: Row(id=x[0], text=x[1])) \
            .toDF() \
            .cache()
```

Figure 4.1: IPython notebook

4.3.2 Selecting a Frequent Pattern

After setting the configuration file, begin at the first cell and press Shift + Enter to execute each cell in order. Continue this until you reach "Manually choose frequent patterns and write them in the configuration file."

You will now need to open each of the frequent pattern output files located at:
`data/FPGrowth/«timestamp»_«collectionId»_«collection_name».txt`



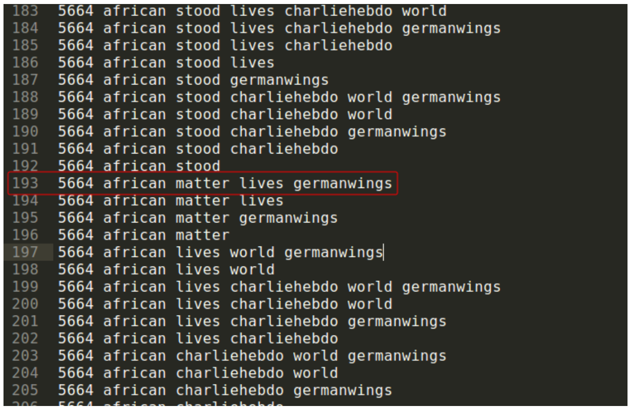
```
{
  "collections": [{
    "Id": "602",
    "name": "Germanwings",
    "type": "binary",
    "FP": ["african", "lives"]
  }, {
    "Id": "541",
    "name": "NAACPBombing",
    "type": "binary",
    "FP": ["bombing", "colorado"]
  }, {
    "Id": "668",
    "name": "houstonflood",
    "type": "binary",
    "FP": ["cambodia", "insurance"]
  }, {
    "Id": "700",
    "name": "wdbj7 shooting ",
    "type": "binary",
    "FP": ["unbelievable", "roanoke", "gun"]
  }, {
    "Id": "686",
    "name": "Obamacare",
    "type": "binary",
    "FP": ["insurance", "health"]
  }, {
    "Id": "694",
    "name": "4thofJuly",
    "type": "binary",
    "FP": ["independenceday", "happy"]
  }]
}
```

Figure 4.2: Collections configuration file

The FP entries seem odd. (HS: I've changed the configuration file image with a new one that has more meaningful terms) Was there overfitting to an odd positive set?

You should now inspect the file for the tokens frequently found together. Look for high frequency patterns. Choose a pattern that seems highly relevant to your collection. We suggest a pattern of two to four words to strike a balance between over and under specification.

Take the pattern and copy it as the value of "FP" in the configuration file.



```
5664 african stood lives charliehebdo world
5664 african stood lives charliehebdo germanwings
5664 african stood lives charliehebdo
5664 african stood lives
5664 african stood germanwings
5664 african stood charliehebdo world germanwings
5664 african stood charliehebdo world
5664 african stood charliehebdo germanwings
5664 african stood charliehebdo
5664 african stood
5664 african matter lives germanwings
5664 african matter lives
5664 african matter germanwings
5664 african matter
5664 african lives world germanwings
5664 african lives world
5664 african lives charliehebdo world germanwings
5664 african lives charliehebdo world
5664 african lives charliehebdo germanwings
5664 african lives charliehebdo
5664 african charliehebdo world germanwings
5664 african charliehebdo world
5664 african charliehebdo germanwings
```

Figure 4.3: Frequent Patterns Mining algorithm output file for Germanwings collection

Can you give examples? (HS: this is an example of FPM output of Germainwings tweets collection)

4.3.3 Building a Training Set

After the frequent pattern has been selected and input into the configuration file, continue to step through the script using Shift + Enter in each cell.

This will take you through the process of building a training set by identifying a positive sample set and a negative sample set and placing those into a DataFrame.

4.3.4 Training a Classifier

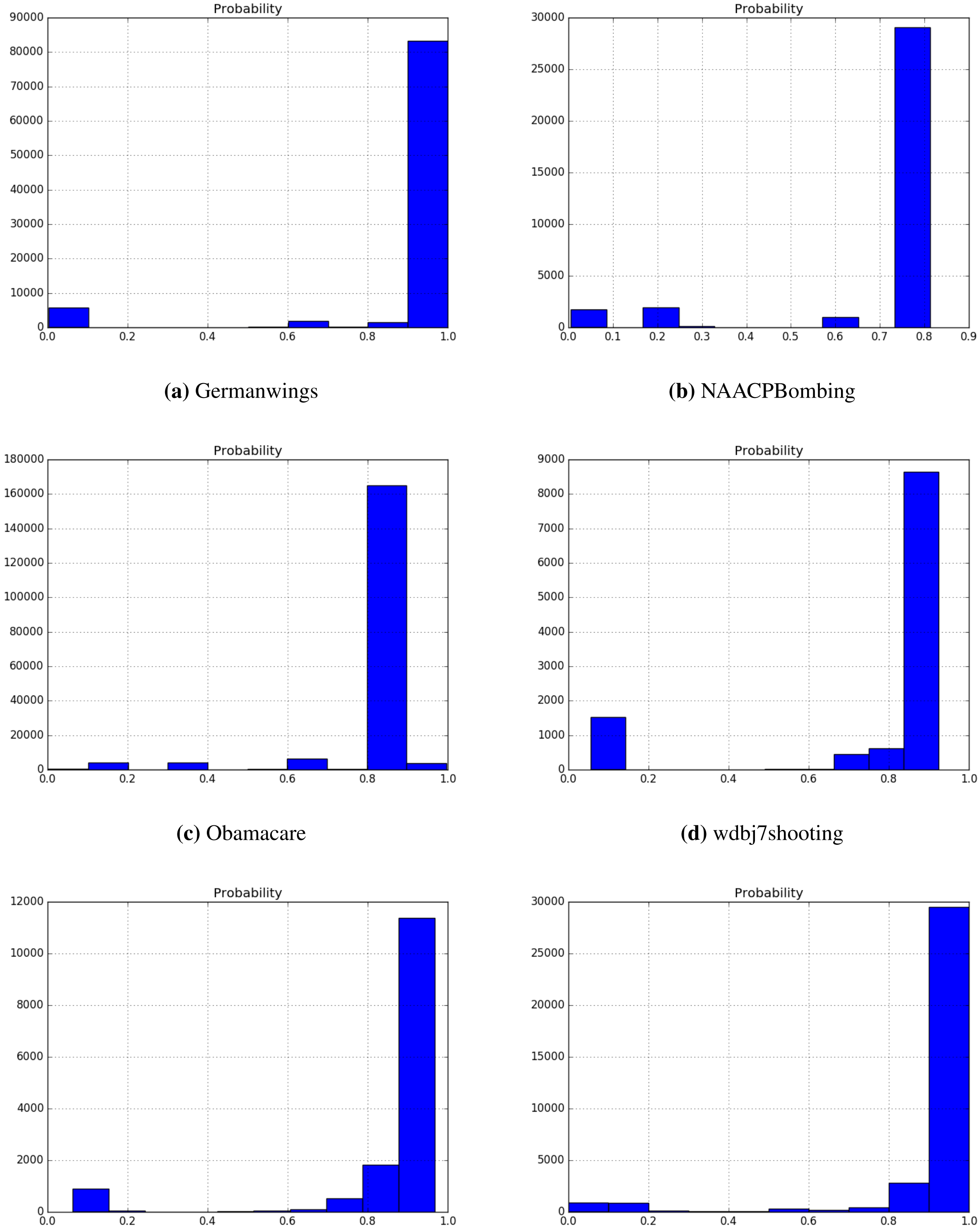
Further progression through the script will actually use the training data constructed in the last step to train a logistic regression classifier.

4.3.5 Running the Classifier

Finally, apply the classification model to our test data and provide a prediction of relevance (1.0) or non-relevance (0.0) for each tweet in the test data. This data can be found in the `predictions` directory and has the same name formatting as the frequent pattern output files.

HS: Figure 4.4 shows the different histograms for the small data sets where higher probability means more likly a tweet to be relavant to the collection

A cross-reference to Figure 4.4, and a cross-reference to Subfigure 4.4e.



(a) Germanwings

(b) NAACPBombing

(c) Obamacare

(d) wdbj7shooting

(e) houstonflood

(f) 4thofJuly

Figure 4.4: Histograms for the small collections of tweets

Chapter 5

Developer Manual

The developer manual will grow and evolve with the project. It documents the tools we are currently using, not tools we are considering for future steps.

5.1 Algorithms

5.1.1 Frequent Pattern Mining

Our methodology for constructing training data is based on a frequent pattern mining algorithm [**han2000mining**]. Spark.mllib version 1.5 provides a parallel implementation of FP-growth, a popular algorithm for mining frequent itemsets.

5.1.2 Logistic Regression

Logistic regression is a popular method to predict a binary response. It is a special case of Generalized Linear models that predicts the probability of the outcome. For more background and more details about the implementation, refer to the documentation of the logistic regression in spark.mllib.

What might a developer have to change in the future?
Might there be scaling concerns?
Might some methods need to be changed for different collections?
Are there different concerns for webpages than tweets?

5.2 Environment Setup

Should this section be in the user manual?

5.2.1 Dependencies

Java

You will want the latest version of Java, which at the time of this writing is Java Version 8 Update 73. Download and installation instructions for your environment can be found at:

https://java.com/en/download/

The Hadoop cluster we are working with is currently running Java Version 7 Update 99. This version of Java is capable of performing everything that we require for this work. However, due to security concerns among other things we recommend using the latest stable version of Java if setting up a cluster from scratch.

Python

You will need Python 2.6.6 (our work should be compatible with newer versions of the 2.x.x line, but we have not tested it). Download and installation instructions for your environment can be found at:

https://www.python.org/downloads/

While the cluster that we have implemented our solution on is using Python 2.6.6, we recommend using the latest stable version of the Python 2.x.x line if setting up a cluster from scratch.

5.2.2 Apache Spark

This system is built on Apache Spark 1.5.0, which can be downloaded at:

http://spark.apache.org/downloads.html

Any newer versions of Spark should also be compatible for the foreseeable future. So if setting up a cluster from scratch, we recommend installing the latest stable version of Spark.

Chapter 6

Plan

You should update this section as time advances

Our work on this project will happen in two phases. In the first phase, we will attempt to create an initial prototype working with a small set of sample data. This phase will be accomplished mainly in our own external environment so that we can experiment more freely with the technologies to gain a better understanding of how they work and interact together.

Phase two of the project will involve taking the work done in phase one and expanding it out to the other small sets of data provided to us, and eventually to larger data sets. We will explore other potential techniques as we deem them necessary. This phase will also include a definition and implementation of an evaluation approach. Phase one is small enough that we feel a subjective evaluation is sufficient, but as we expand it will become essential to have a more concrete, quantitative measure of effectiveness.

Please see Table 6.1 for a weekly breakdown of work.

Table 6.1: Weekly breakdown of work to be done.

Weeks	End Date	Tasks
Week 1	22 Jan.	Understanding the classification task
Week 2	29 Jan.	• Understanding the classification task • Read about Hadoop streaming using Python
Week 3	5 Feb.	Start online tutorials about Hadoop and Apache Spark. [**pythonSparkTutorial**][**solrTeamTutorial**]
Week 4	12 Feb.	Continue online tutorials about Hadoop and Apache Spark. [**hbaseShellTutorial**]
Week 5	19 Feb.	**Phase 1 will include only tweets small data set:** • Understanding the classification task • Read about Hadoop streaming using Python
Week 6	26 Feb.	• Prepare training data using Solr • Build classifier using Apache Spark
Week 7	4 March	• Build classifier using Apache Spark • Get output data format from Solr team
Week 8	11 March	Optimize classifier performance

Weeks	End Date	Tasks
Week 9	18 March	**Phase 2 will include tweets and webpages:** **Once Spark on the cluster is updated to version 1.5 we will do the following:** • Run our methodology to classify the tweets on the cluster. • Apply the Frequent Pattern methodology on the cleaned webpages provided by Collection Management team. • Develop HBase interface through which the classifier prediction output will be saved on HBase.
Week 10	25 March	Design an evaluation approach to test and evaluate our methodology.
Week 11	1 April	Discuss with GRAs and Solr team to finalize HBase schema.
Week 12	8 April	Move functioning prototype over to newly updated Hadoop cluster. Train multiple classifiers and pick the best model per collection.
Week 13	15 April	More research on hyper-parameter optimization. Check the feasibility of integrating hyper-parameters optimization library [**bergstra2013hyperopt**] output with Spark.
Week 14	22 April	Search for known approaches to select the most representative data samples for each collection. Then check whether training a classifier using these data samples will enhance the performance or not.
Week 15	29 April	Final evaluations and modifications to the system.

Chapter 7

Conclusion

Make clear if did both tweets and webpages

Since our last report, we have made a great deal of progress in terms of getting our prototype ready to work with other teams. We have managed to process all six of the provided small data files. From those files, we have created the first batch of usable, classified data and imported that data into the HBase system for use by other teams in their areas of the project. In the conclusion of our last paper, we mentioned that we were waiting for updates to be done to the cluster, namely updating to a newer version of Spark. This has since been completed as of this week, and we will now be able to move our system from our private virtual machine over to the Hadoop cluster. This is our next step, and one that we aim to have done as soon as possible. Once the system is up and running on the cluster, we will be able to modify it as mentioned previously so that it will directly read to and write from HBase, which will speed up the processing and eliminate the need for someone to manually index the output files.

Chapter 8

Future Work

There are a few points with our work that need to be accomplished soon to provide the other groups working with this project some useful information. First and foremost, we are in a position to provide the other groups information regarding the precision of the small collections. This will be useful for them to know if they should be working with the entirety of the collections or only a subset that we deem as related to the small collection and the event it is supposed to encapsulate.

Another point that we need to work towards is transitioning our work from the VM we created initially over to the Hadoop cluster now. The cluster has been upgraded with the necessary version of Spark that will allow us to perform Frequent Pattern Mining (FPM) analyses to build training sets for collections of documents.

Since we can now transition to the cluster, we also need to update our scripts to interact directly with HBase instead of writing to intermediary files that need to be loaded in manually.

Further reaching future work would be combining multiple classifiers, trained on different aspects of the same training set, into a single classifier that may provide a more robust prediction than any of the single classifiers on their own.

Also discuss about how to handle collection growth.

Always have a year.
For webpages you can give date accessed.