"Statement Classification Tool Based on Frequent Pattern Mining of Twitter Stream"

Abstract

For this project, I built a web application using Ruby on Rails and Java. I used this project as an opportunity to learn Ruby on Rails, as well as increase my skill set in Java, HTML, and SQL. The project was designed to classify a statement. The classification is based on frequent patterns from the Twitter stream along with the keywords used to classify the statements in the stream.

Faculty Advisor
Dr. Raphael Finkel

Masters Candidate
Mark Dazey

Index

Summary

For this project, I built a web application using Ruby on Rails and Java. I used this project as an opportunity to learn Ruby on Rails, as well as increase my skill set in Java, HTML, and SQL. The project was designed to classify a statement. The classification is based on frequent patterns from the Twitter stream along with the keywords used to classify the statements in the stream.

Introduction

There is an enormous ammount of data now passing through Twitter. In 2011, Twitter reported it took them over three years to reach one billion tweets. In that same year the time it took to reach one billion tweets had decreased to one week, they were now averaging 140 million tweets a day(2). The next year, Twitter reported that they were now receiving approximately 340 million tweets a day and one billion every three days(1). In March of this year Twitter reported over 400 million tweets are being created a day(3). At 400 million tweets being created a day there are almost three billion tweets created every week.

There are things to be learned from the Twitter ecosystem. Public and private organizations have realized the potential in the data being collected by Twitter. There are companies anazlying the data for business to help them better understand the communications on twitter along with consumer preferences. Three companies are certified by twitter to re-sell the twitter feed to compaines and or researchers(10).

The Library of Congress along with Twitter announced in April of 2010 that the entire twitter archive had been donated to the Library of Congress. The Library of Congress announced in January of 2013 they had received 170 billion twitter posts and that the number was growing daily(4). The Library of Congress has said they have had approximately 400 request from researchers wanting access to the data(4).

Twitter provides developers with access to the public stream of tweets. Twitter has two levels of access to the stream of tweets. The first is the sample stream which returns a small random sample of the stream of public tweets. The second is the firehose stream which returns all public tweets(5).

Frequent Pattern Mining

In data mining, frequent pattern mining has been a theme of research for over a decade(7). The concept was first proposed by Argwal et al in 1993 for analysis of "shopping baskets". The purpose was to analyze customer buying habits by finding associations between the different items a customer would place in their basket(7). Since this first research paper was published there have been hundreds of follow-up research publications, in which the topic was expanded to a wide variety of data-types, scalable minging methodologies were introduced and a variety of new applications(7).

There are three basic frequent pattern mining algorithms apriori, FP-growth, and eclat(7). Methods are also available for multi-level, multi-dimensional, and quantitative rule mining(7). The first algorithm apriori works off of the principle that if an itemset of 5 is frequent then the subsets of 1,2,3, and 4 are also frequent. To generate the candidate itemsets a pass of the dataset is done generating the 1-itemsets which are frequent. Then, a second pass is done starting from the itemsets generated in the first pass and generating all 2-itemsets which are frequent. This continues until no additional items are added to any itemset(7).

The apriori algorithm has a large cost associated with repeated scans of the dataset, and with the generation of candidate itemsets(7). In 2000, Han et al introduced an algorithm for generating frequent patterns without candidate generation this algorithm is FP-growth.

The FPGrowth algorithm works by using a divide-and-conquer approach(7). The first scan of the dataset by the FPGrowth algorithm generates a list of freqent items in frequency-decending order. Using the frequency-decending list the dataset is compressed into a frequent pattern tree (fp-tree)(7). By using the fp-tree the problem of finding long patterns is transformed into searching for shorted patterns recursively and then concatenating them(7). In this project I chose to use the FPGrowth algorithm for frequent pattern identification. This method was chosen because of the lower cost of running than the apriori algorithm.

Overview

For this project I wanted to use new technologies and learn additional skills and techniques. In order to accomplish the task of learning something new I looked for a language I had not previously used for web-development. I decided by using Ruby on Rails I would have the most opportunity to learn new topics. Through Ruby on Rails I was also able to learn about the Model View Controller architecture.

The backend part of this project was done using Java, in my current position I am working as a Java developer and took this opportunity to further my skills in the Java programming language. I was able to learn additional skills in the language including multi threaded programming with the use of thread pools. I also was able to develop my skills in processing a stream from a web site.

Due to Twitter's access restrictions on the Firehose stream I was only able to access the sample stream. For the pruposes of this project this stream was sufficent, however better results could have been obtained with access to a larger dataset from a more shorter period of time. This project was aimed at the english language only, since the Twitter stream includes multiple languages the stream had to be filtered this reduced the sample size even further.

Design

The design of this project is broken down into two major parts. First the backend section which aquires and process the tweets from Twitter. And the Second part is the frontend which allows users to access and find suggested classifications for statements. The backend was designed to be multi-threaded so the stream from twitter could be consumed without interuption.

Backend

The system on the backend was developed using java, this allowed for me to increase my skill set with java. I used a multi-threaded approach to this project to allow for consumption of the Twitter feed without interruption. Twitter will restrict access if you have to many attempts to access the stream. Having one contiuous access of the stream prevented this from being an issue.

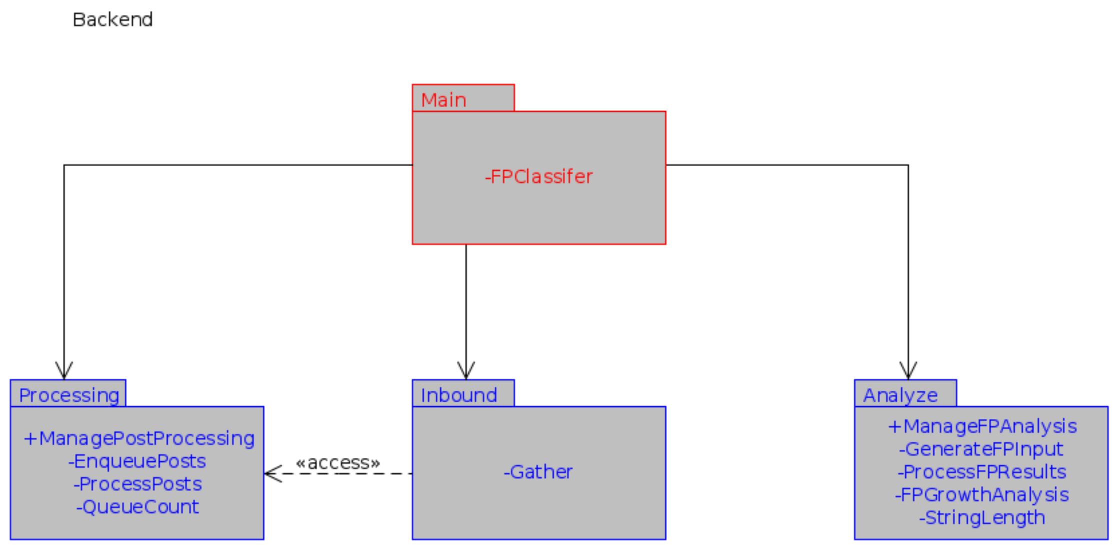


Figure 1: Diagram of Packages in the Backend

The backend was divided up into four packages; the main package, the inbound package, the processing package, and the analysis package. The main package detailed in figure 2, was used to manage the running of the application and to generate some statistics which were put into the log. The inbound package, was used to retrieve posts from Twitter, the detail of this package can be see in figure 3. The inbound package was designed to retrieve posts from twitter and to only stop when collection is stopped through an administrative function on the web interface. If the database reached a high level of unprocessed posts, collection was designed to be skipped while processing was able to catch up.

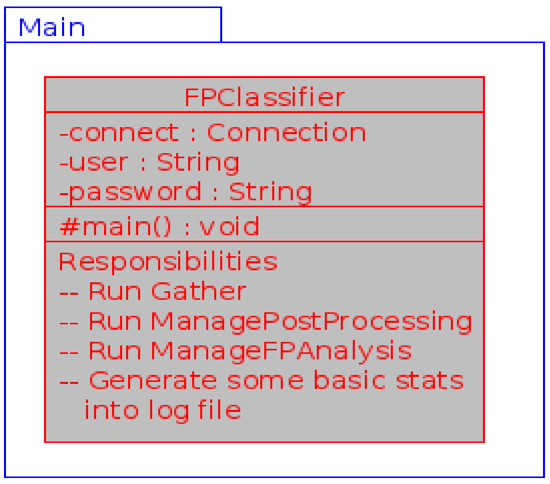


Figure 2: Class diagram for main package

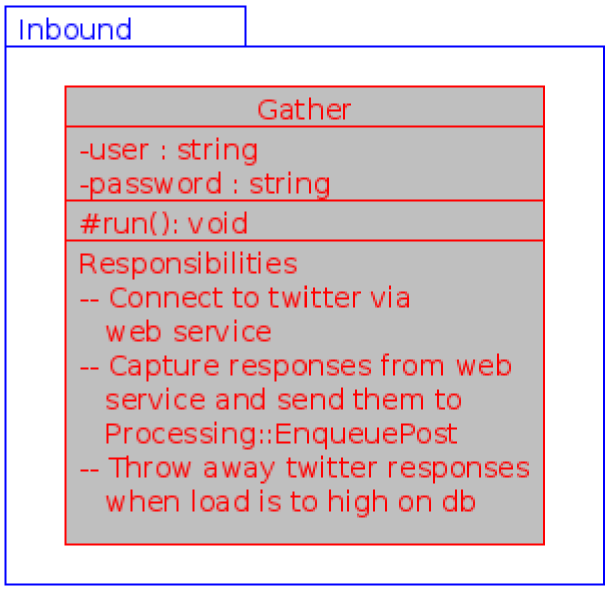

Inbound
Gather
-user : string
-password : string
#run(): void
Responsibilities
-- Connect to twitter via web service
-- Capture responses from web service and send them to Processing::EnqueuePost
-- Throw away twitter responses when load is to high on db

Figure 3: Class diagram for the inbound package

The third package was the processing package, this is where a majority of the work is done. The details of this package can be seen in figure 4. This package was responsible for enquing the incoming twitter posts. It is also responsible for processing the posts and removing un-usable posts. The un-usable posts contain any post which is not in english or does not contain a keyword.

The last and most important part of the backend is the analysis package. In this package the usable posts obtained from Twitter are prepared for analysis by the FPGrowth algorithm. The class diagram for the analysis package can be seen in Figure 5. This package is responsible for preprocessing the data to be analyzed by the FPGrowth algorithm, running the data through the FPGrowth algorithm, and for storing the results from the FPGrowth algorithm.

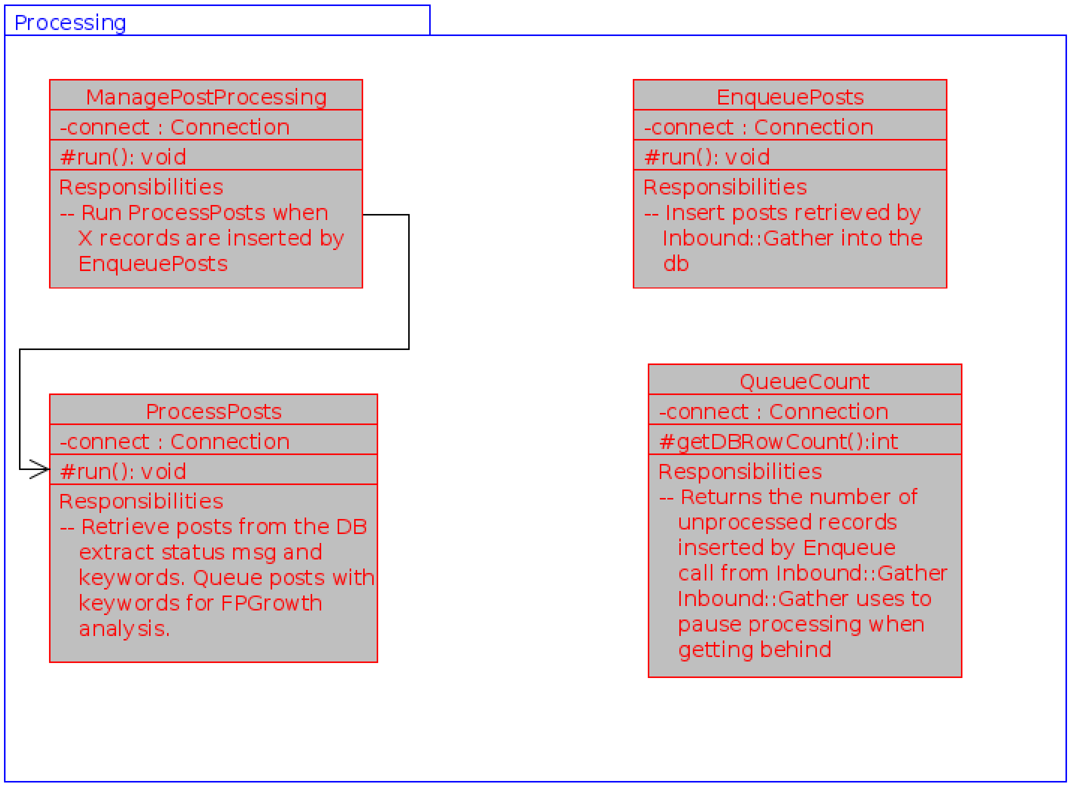

Processing
ManagePostProcessing
-connect : Connection
#run(): void
Responsibilities
-- Run ProcessPosts when X records are inserted by EnqueuePosts
EnqueuePosts
-connect : Connection
#run(): void
Responsibilities
-- Insert posts retrieved by Inbound::Gather into the db
ProcessPosts
-connect : Connection
#run(): void
Responsibilities
-- Retrieve posts from the DB extract status msg and keywords. Queue posts with keywords for FPGrowth analysis.
QueueCount
-connect : Connection
#getDBRowCount():int
Responsibilities
-- Returns the number of unprocessed records inserted by Enqueue call from Inbound::Gather Inbound::Gather uses to pause processing when getting behind

Figure 4: Class diagram for the processing package

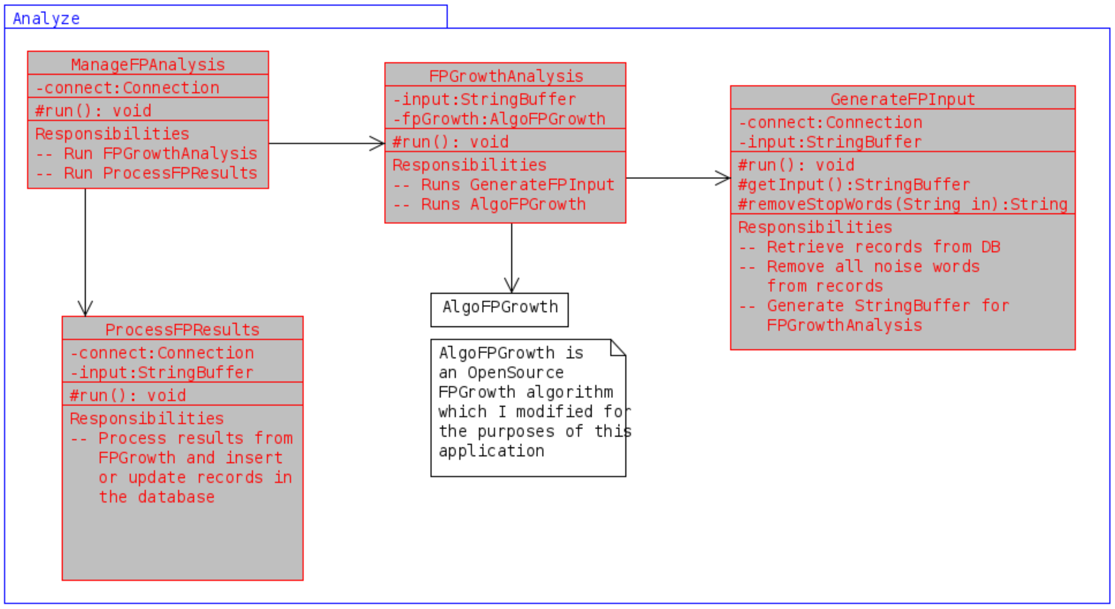


Figure 5: Class diagram for analysis package.

Data Preprocessing

For the FPGrowth algorithm to be able to handle the data obtained from the Twitter stream some preprocessing needs to be completed. For the algorithm to process the posts they need to be formatted with each statement being on a single line with the words seperated by a single space. There are also many words which are necessary to form a coherent sentence that do not convery any actual meaning. These nuisance words also needed to be remove prior to processing by the algorithm to prevent false positives on words such as 'are'.

The posts from Twitter required for the nuisance words to be removed. Two approaches to removing the nuisance words were considered. The first approach considered was removing any word with less than X characters. The second approach considered was to develop a list of words to be removed that contain most of the nuisance words.

The first approach could have lead to too many or too few words being removed. Take for example the word mom, if we choose X to be 3 then mom would be removed from the post for processing. This word could convey meaning for a post, if it were around mothers day, it could potentially lead to a classification relating to mothers day. This method would have also left words like what, where, and when, these words do not convey a specific meaning for a sentence.

For this project I chose to use the second approach and developed a list of words which should removed. A partial list of removed words is in table 1, these words do not convey meaning and are frequently used in many sentences. By using the list of words I was able to accomplish the task of removing words which are not necessary and leaving those which are. This also allows for the list to be modified and changed to provide better results.

Post Processing

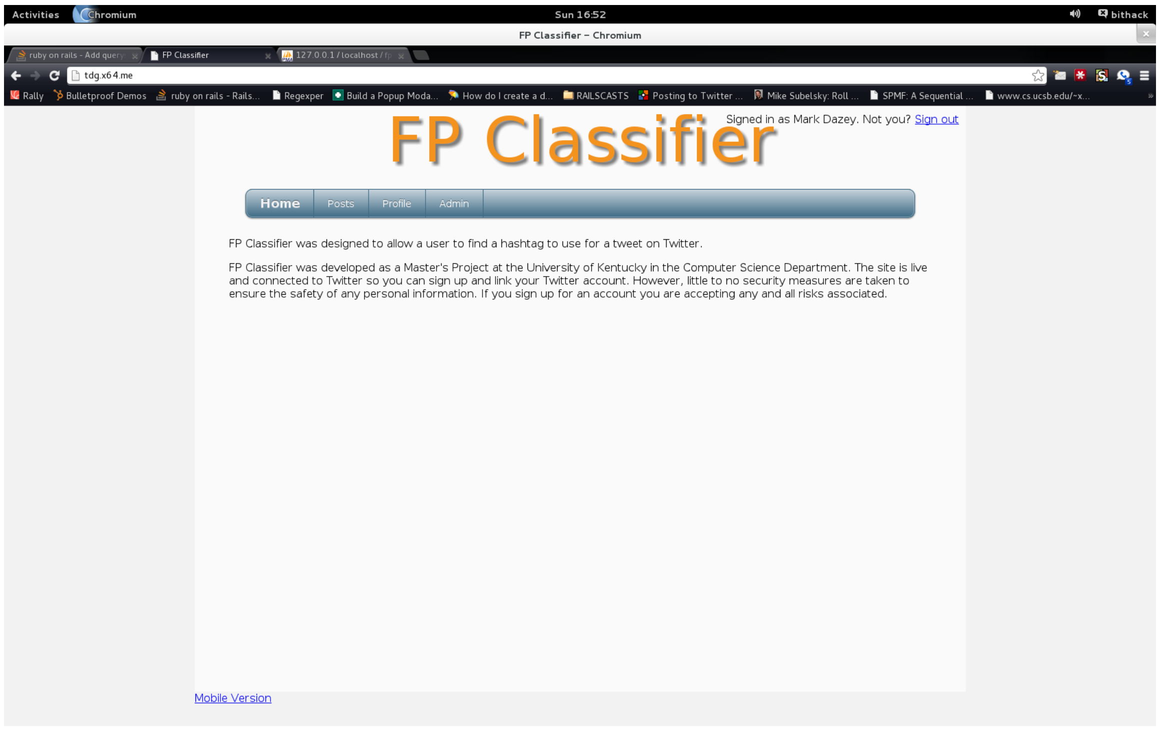
After the data was processed by the FPGrowth algorithm it needed to be processed into a usable form for user interactions. The ouput of the algorithm consisted of the frequent pattern and the number of times the pattern was found. The results need to be stored in the database so the ruby on rails web application can access them and provide classifications to a user. The pattern was stored along with pattern frequency.

Removed Words
And
You
I
An
Are
Of
Who
What
Where

Table 1: Partial list of nuisance words

User Interface

The user interface was designed to be a web user interface, allowing a user to interact with the system through any web browser. With the number of browsers available to access websites today and the types of devices available a flexible user interface was designed. The user interface uses javascript, html, and css to provide an interactive system to the user. This is also designed to work with mobile browsers, tablet browsers and traditional web browsers.



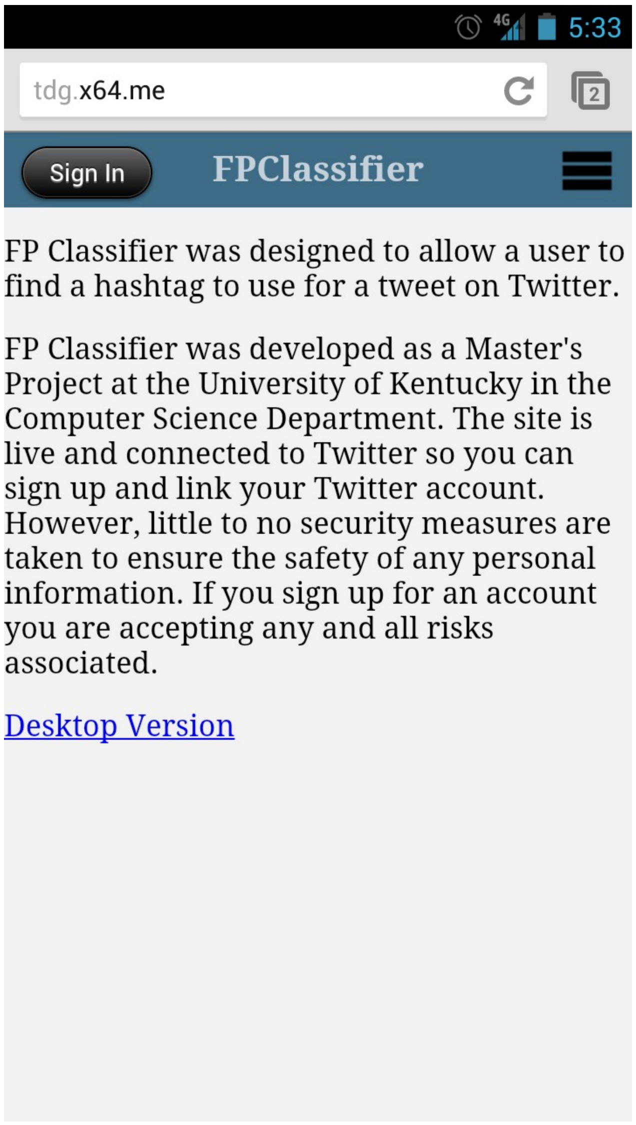
FP Classifier was designed to allow a user to find a hashtag to use for a tweet on Twitter.

FP Classifier was developed as a Master's Project at the University of Kentucky in the Computer Science Department. The site is live and connected to Twitter so you can sign up and link your Twitter account. However, little to no security measures are taken to ensure the safety of any personal information. If you sign up for an account you are accepting any and all risks associated.

Figure 6: Web browser user interface

To generate the dynamic content on the web page I decided to use Ruby on Rails. Ruby on Rails uses a Model-View-Controller set up, this allowed for the greatest flexibility in the application development. One controller is used for multiple views of the same information. The multiple views allow for a user interface designed for mobile browsers and one for traditional browsers.

Through the user interface anyone can sign up for an account. After signing up they will have access to generate classifications for a post they have created. They can go further and setup Twitter integration and actually submit the post to their Twitter account. The administrative interface allows for an admin user to shutdown the processing of new Twitter posts or to just shut down the retrieval of new Twitter posts.



FP Classifier was designed to allow a user to find a hashtag to use for a tweet on Twitter.

FP Classifier was developed as a Master's Project at the University of Kentucky in the Computer Science Department. The site is live and connected to Twitter so you can sign up and link your Twitter account. However, little to no security measures are taken to ensure the safety of any personal information. If you sign up for an account you are accepting any and all risks associated.

Desktop Version

Figure 7: Mobile Interface

Classification Generation

The overall goal of this project is to provide a classification of a new post based on the data mined from existing Twitter posts. To accomplish this task thus far the data has been collected, processed, and Frequent Patterns have been identified. Now we need to take a new post and determine a classification. The method for determining the classification is based on looking for the best match.

To find a classification the post provided by a user is received. Then the nuisance words from the serve application are removed from post. Also the words from the post are sorted by length. The results of the first step of processing in the classification generation can be seen in table 2.

Users Post	**Processed Post**
"This is an example post looking for a classification"	"classification example looking post"

Table 2: Results from first step of classification generation for an example post

The second step for generating a classification is to find the best match from the database. To do this we look for the longest match with the highest frequency of usage. Psuedo code for this process is shown in table 3. From this point we need to find the best keyword to use as a classification.

```
select * from frequents order by frequency desc, length(pattern) desc
for each pattern in frequent
    for each word in post
       if frequent.include(word)
           patterns.add(pattern)
patterns.sort_by(frequency,pattern.length)
```

Table 3: Psuedo code for finding best matched pattern

Keywords are associated to the original posts not the patterns which were mined. So we need to find the best keyword to use as a classification. The keyword with the most usage for existing posts containing the best patten is suggested as the classification. In addition to the best classification the entire list of keywords for the pattern are returned so modifications can be made.

In an effort to increase the accuracy of the classifications new posts are added to the next round of calculations. If a user selects the third classification in the list as the one they wish to use that is recorded. Next time someone makes a post with the same pattern match the third choice may be bumped up to the second position. After many times of this happening the best classification can be altered to match what users are selecting.

What Was Learned

While working on this project, I learned to select the best technologies, design, and implement a complete system. I also learned the concepts of model-view-controller, multi-threaded design, and integration of a whole system. Through this project I was able to teach my self more aspects of web design including Ruby on Rails, HTML, javascript, and CSS. I was also able to teach myself about database design and using Ruby's activerecord to complete queries of a database. The most important part of this project was getting to learn and work through the entire software development lifecycle.

Future Work

This project can be improved upon in many ways in the future. The first item I would like to see improved upon is to add some sort of spelling correction. Currently misspellings of words result in words not matching in the FPGrowth algorithm and potential patterns being missed.

References

(1) http://blog.twitter.com/2012/03/twitter-turns-six.html

(2) http://blog.twitter.com/2011/03/numbers.html

(3) http://blog.twitter.com/2013/03/celebrating-twitter7.html

(4) http://blogs.loc.gov/loc/2013/01/update-on-the-twitter-archive-at-the-library-of-congress/

(5) https://dev.twitter.com/docs/api/1.1

(6) https://dev.twitter.com/docs/api/1.1/get/statuses/sample

(7) http://www.cs.ucsb.edu/~xyan/papers/dmkd07_frequentpattern.pdf

(8) Agrawal R, Imielinski T, Swami A (1993) Mining association rules between sets of items in large databases. In: Proceedings of the 1993 ACM-SIGMOD international conference on management of data (SIGMOD'93), Washington, DC, pp 207–216

(9) Han J, Pei J, Yin Y (2000) Mining frequent patterns without candidate generation. In: Proceeding of the 2000 ACM-SIGMOD international conference on management of data (SIGMOD'00), Dallas, TX, pp 1–12

(10) https://dev.twitter.com/programs/twitter-certified-products/products