SPOONS: NETFLIX OUTAGE DETECTION USING MICROTEXT CLASSIFICATION

A Thesis

Presented to

the Faculty of California Polytechnic State University

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Eriq Augustine

March 2012

COMMITTEE MEMBERSHIP

TITLE:	SPOONS: Netflix Outage Detection Using Microtext Classification
AUTHOR:	Eriq Augustine
DATE SUBMITTED:	March 2012
COMMITTEE CHAIR:	Alex Dekhtyar, Ph.D.
COMMITTEE MEMBER:	Clint Staley, Ph.D.
COMMITTEE MEMBER:	Franz Kurfess, Ph.D.
COMMITTEE MEMBER:	Foaad Khosmood, Ph.D.

Abstract

SPOONS: Netflix Outage Detection Using Microtext Classification

Eriq Augustine

Every week there are over a billion new posts to Twitter services and many of those messages contain feedback to companies about their services. One company that has recognizes this unused source of information is Netflix. That is why Netflix initiated the development of a system that lets them respond to the millions of Twitter and Netflix users that are acting as sensors and reporting all types of user visible outages. This system enhances the feedback loop between Netflix and its customers by increasing the amount of customer feedback that Netflix receives and reducing the time it takes for Netflix to receive the reports and respond to them.

The goal of the SPOONS (Swift Perceptions of Online Negative Situations) system is to use Twitter posts to determine when Netflix users are reporting a problem with any of the Netflix services. This work covers the architecture SPOONS system and framework as well as outage detection using tweet classification.

Acknowledgements

Thanks Alex, ABRA, Netflix especially Kevin McEntee, and all the funions. Thanks to Farscape for encouraging team bonding and providing a common enemy.

Contents

List of Tables

List of Figures

Part 1

Introduction

Chapter 1

General Problem: Swift Perception Of Online Negative Situations

Twitter is an immensely popular micro-blogging service. According to Twitter, as of March 14th 2011, approximately one billion micro-posts, *tweets*, were being posted per week[28]. Because of the low time and effort cost of tweeting, only a few seconds from a smart phone, Twitter users post tweets about almost every aspect of their daily lives. Because of this large stream of information, Twitter makes an excellent source of information for data miners interested in real-time events. Already, researchers have been using Twitter to attempt to track and model disease outbreaks[5], earthquakes[15], and the stock market[11].

Netflix is the one of the largest online Internet subscription service for streaming movies and television shows. Netflix has over 25 million subscribers watching media streamed to over 450 different platforms. Even a short disruption of their streaming service can affect millions of users. Therefore, quickly detecting service outages is essential to keep customers happy. However, service outage detection is no trivial matter in Netflix's environment. In addition to constantly streaming thousands of different videos to hundreds of different platforms, Netflix also has to deal with problems caused by most of their infrastructure being hosted in the cloud with Amazon Web Services (AWS).

Netflix saw the power in Twitter as a potential data source for detecting service outages that is orthogonal to their current, more traditional outage detection methods. Currently, Netflix utilizes four different methods for detecting outages:

Internal Monitoring Systems. Like any sizable service providing company, Netflix utilizes many different internal monitoring systems to detect service outages. However, there are some classes of problems that are difficult to solve with internal monitoring. These problems include corrupt video files or a problem on a third-party delivery platform such as Roku or AppleTV. These problems are obvious to the end user, but very difficult to detect internally. In addition, the internal monitoring systems share the same infrastructure as the service providing system. Therefore, a problem in the infrastructure can cause both systems to go down at the same time.

External Monitoring Systems. Netflix contracts with external services that can periodically probe its systems to try and detect problems. However, this model too has problems. There are many problems that cannot be seen from an external probe. Also, if this system probes too often then it is taking compute time away from the servers that are trying to deliver content to end users.

Customer Service. Calls to customer service are a very straight-forward way to detect outages. Unfortunately, this method is very slow and inconsistent. It takes a lot of frustration to get a user to lookup a phone number and complain.

Manual Twitter Observation. Manual observation shows that there is usually a response on Twitter when Netflix suffers a service outage. Figure 1.1 shows some tweets that occurred during a disruption of Netflix's service to the Nintendo Wii. However without any infrastructure, Twitter observation is slow and inconsistent. It is also very time consuming to have someone constantly watching Twitter for signs of an outage.

Given all these deficiencies Netflix wanted a monitoring system that is separate from their infrastructure, fast, and does not require any human intervention[16].

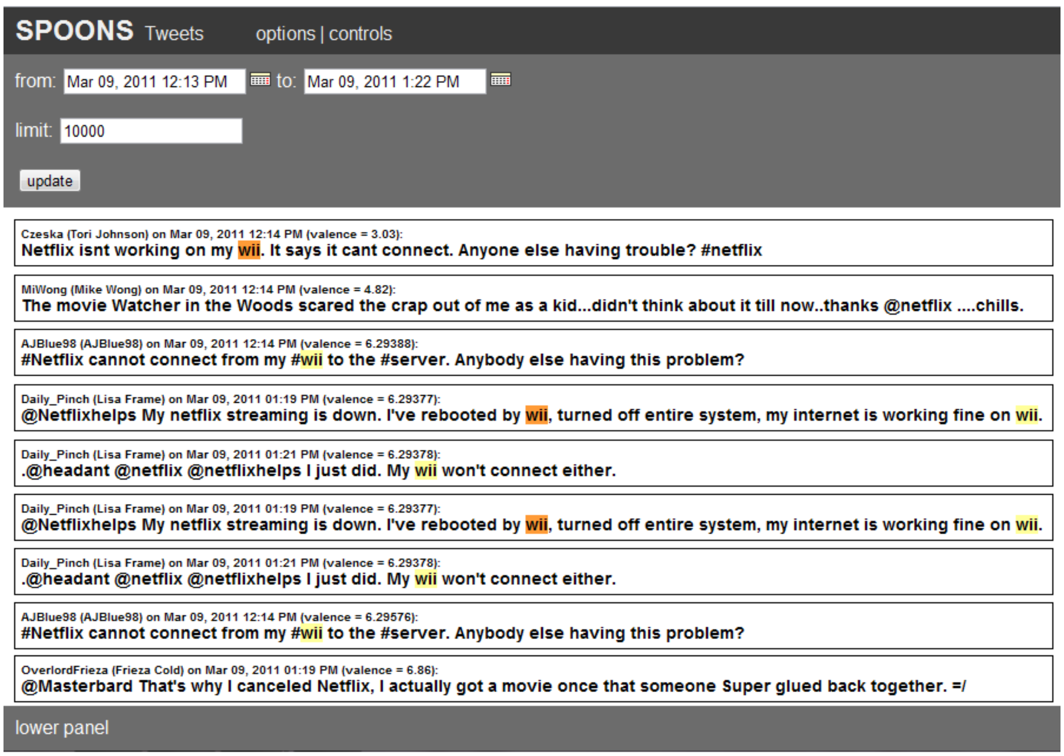

Figure 1.1: Tweets posted on March 9, 2011 during a disruption of Netflix streaming to the Nintendo Wii console.

Chapter 2

Solution Overview

SPOONS (Swift Perception Of Online Negative Situations) is a system that is designed to use tweets to detect outages in Netflix content delivery systems. At present, the system supports a wide variety of detection methods that use some combination of time series analysis, classification, natural language processing, sentiment analysis, and filtering.

Figure 2.1 shows how the SPOONS system can be divided into three main parts: input; analysis methods; and output. The inputs are tweets gathered from Twitter. Then the analysis methods use a combination of sentiment estimation, classification, and traffic volume analysis to detect when an outage is occurring. The outputs of the system are: email alerts to Netflix engineers, and a web UI that displays information about the outage.

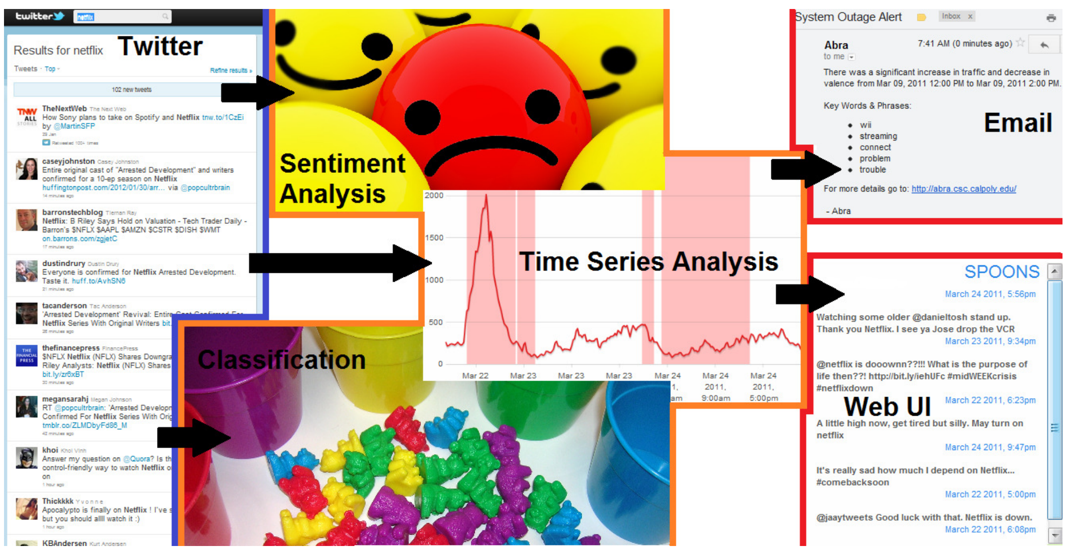

Figure 2.1: This system concept diagram shows the general flow of processing done in the SPOONS system.

Chapter 3

Ethics of Twitter Observation

The work in this project uses content that users post on Twitter without their knowledge. This monitoring system isn't being announced to the public because widespread knowledge of it would increase the likelihood of a malicious attack. This practice may lead to concerns about the level of privacy or ownership being provided to Twitter users regarding the content they post through the Twitter services. The goal of this section is to address these concerns by providing more information about the Twitter services and how the SPOONS system and this work uses the tweets.

3.1 Twitter Terms of Service

According to Twitter Terms of Service[29] agreement that everyone accepts automatically by accessing or using Twitter services:

"You retain your rights to any Content you submit, post or display on or through the Services. By submitting, posting or displaying Content on or through

the Services, you grant us a worldwide, non-exclusive, royalty-free license (with the right to sublicense) to use, copy, reproduce, process, adapt, modify, publish, transmit, display and distribute such Content in any and all media or distribution methods (now known or later developed)."

"This license is you authorizing us to make your Tweets available to the rest of the world and to let others do the same."

"You agree that this license includes the right for Twitter to make such Content available to other companies, organizations or individuals who partner with Twitter for the syndication, broadcast, distribution or publication of such Content on other media and services, subject to our terms and conditions for such Content use."

"We encourage and permit broad reuse of Content. The Twitter API exists to enable this."

"Such additional uses by Twitter, or other companies, organizations or individuals who partner with Twitter, may be made with no compensation paid to you with respect to the Content that you submit, post, transmit or otherwise make available through the Services."

In short, Twitter takes ownership of user tweets as soon as they are posted on Twitter. Using the Twitter API allows SPOONS to obtain the tweets with the consent of Twitter. Therefore, the collection and analysis of Twitter data by SPOONS is well withing the Twitter Terms of Service.

Chapter 4

SPOONS Requirements

Netflix has provided the following set of key requirements to be met by the SPOONS system:

Structural Independence. The outage detection system shall be structurally independent of both the software and the hardware infrastructure used by Netflix. It shall rely only on information that is publicly available and free for use. This ensures that the outage detection system stays up even when any or all Netflix servers are experiencing downtime.

Use of Amazon Web Services. Netflix is one of the largest customers of Amazon.com's cloud computing service, Amazon Web Services (AWS). AWS allows users to create new cloud machines (instances) in many regions throughout the world. The outage detection system shall be deployed on one or more AWS servers that are operationally independent of other AWS servers used by Netflix. Using a cloud solution allows the outage detection and alert system to be deployable on a global scale.

Real-Time. Netflix's streaming services run in real-time and any downtime has an immediate impact on customers. To minimize that impact, the outage detection system shall notify Netflix of detected outages as soon as possible.

Precise Outage Detection. The number of non-outage situations that raise an alert shall be minimized. While a small number of false positives detected in real-time may be acceptable, the outage detection system shall detect outages and generate alerts with as high precision as possible.

Comprehensive Outage Detection. Not all Netflix outages will generate a signal on Twitter. Those that don't may be allowed to go unnoticed by the outage detection system (as the system will have no basis for detecting them), but any outage that causes a signal on Twitter shall be detected.

User-Friendly Online UI. The outage detection and alert system shall have an easy-to-use, informative, online UI which shall provide Netflix employees with real-time information and historic data about the state of Netflix according to Twitter. The information provided shall include:

- times of outages;
- times of other anomalous events;
- current and recent Netflix-related Twitter traffic trends;
- and samples of Netflix-related tweets.

Chapter 5

Contributions and Organization

SPOONS is a continual team effort and has been touched and improved by many different people. The idea originated at Netflix and was passed to the ABRA team at Cal Poly. The ABRA team has published a paper on SPOONS [2]. In addition, Cailin Cushing defended a thesis using SPOONS[6].

The main contributions of this work are the design and implementation of the SPOONS system, framework, server architecture, distribution model, and database schema. As well as the design and implementation of classification based outage detection methods.

The rest of the paper is organized as follows. Chapter 2 covers background and related work. Chapter 3 discusses the architecture of SPOONS. Chapter 4 discuss the details of the classifiers used in SPOONS. Chapters 5 extends the problem of classification to full outage detection. Chapter 6 wraps up the paper.

Part 2

Background & Related Work

Chapter 6

Twitter Classification

There has been much work in using classifiers on both tweets and Twitter users. Most of the classification effort has gone into trying to determine the sentiment, the general feeling, of a tweet [12][17][27][31]. Raz et al. tackle the task of classifying humorous tweets as a specific type of humor such as irony, observational, or wordplay[26]. The traditional text classification task of topic modeling has also been attempted various times[10][33]. Instead of trying to classify tweets, Pennacchiotti et al. try to classify a user associations from their tweets[20].

The common theme in all of these classification attempts is that tweets are much more difficult to classify than more traditional media sources, eg. news articles, because of their length and language. The 140 character limit on tweets severely limits the information content of a tweet. Partially because of the character limit, slang and informal language are commonplace in tweets.

Chapter 7

Classifiers

SPOONS uses a variety of different classifiers for text classification. This section gives an overview of each different type of classifier used.

7.1 Naive Bayes

Naive Bayes classifier work by applying the Bayes' theorem with the assumption that the probability of each feature in a document is independent from the probability of any other feature appearing in the same document.[13][8]

Let c be a specific class in the set of possible classes C. Let d be a document composed of a vector of n features, $d = (\mathrm{f}_1, \mathrm{f}_2, ..., \mathrm{f}_n)$. The Bayes' theorem states that the probability of observing class c given document d can be represented as:

$$Pr(c|d) = \frac{Pr(c) \cdot Pr(d|c)}{Pr(d)} \tag{7.1}$$

$Pr(c)$ is the prior probability of class c. That is, the probability of observing c regardless of the document attached to it. When training the classifier, this is just the percentage of times that the class appeared in the training set.

$Pr(d)$ is the prior probability of document d. Like $Pr(c)$, it is just the probability of observing the collection of features d regardless of the class associated with it. Note that for classification, it may not be necessary to compute $Pr(d)$ because it is constant among all documents and classes. A classifier can just choose the class with the largest $Pr(c) \cdot Pr(d|c$ term.

$Pr(d|c)$ is the probability of observing document d given that d is already recognized as belonging to class c. Remember that document d is really just a vector of n features, $(\mathrm{f}_1, \mathrm{f}_2, ..., \mathrm{f}_n)$. Assuming **conditional independence** (the naive part in Naive Bayes), $Pr(d|c)$ can be constructed as a product of the probability of observing each feature in d:

$$Pr(d|c) = Pr(\mathrm{f}_1|c) \cdot Pr(\mathrm{f}_2|c) \cdot ... \cdot Pr(\mathrm{f}_n|c) = \prod_{i=1}^{n} Pr(\mathrm{f}_i|c) \tag{7.2}$$

Now the last step is to estimate the conditional probabilities of the n features. When dealing with discrete features, then estimating $Pr(\mathrm{f}_m|c)$ $(1 \leq m \leq n)$ can be done by finding the percentage of training documents that contain feature f_m and have class c.

7.2 Bayes Net

A Bayesian Networks are probabilistic, directed acyclic graphs that represents a set of random variables and their conditional probabilities. In a Bayesian Network, each edge represents the conditional probability between two nodes. Each node represents a variable and a probability function that takes as input the state of the node's parents.[19][18]

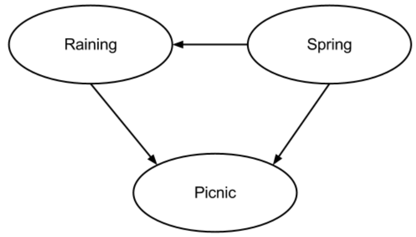

Figure 7.1: A simple Bayesian Network modeling the chance of going on a picnic given the season and weather. The season affects the weather and both the season and weather affect the chance of going on a picnic.

Figure 7.1 shows a simple Bayesian Network that models the chance of going on a picnic. Note that the whether or not it is Spring affects the chance of it raining; and both the season and weather affect the chance of going on a picnic.

Figure 7.2 shows the probability functions for the network. The chance of the season being Spring if fully independent, and therefore takes no parameters into its probability function. However, the weather and picnic decision takes one and two input parameters respectively.

7.3 J48

J48 is a specific implementation of the C4.5 algorithm. C4.5 is an algorithm that is used to generate a decision tree given a training set.

7.3.1 Decision Trees

A decision tree is a simple data structure used to come to come conclusion based off of a number of observations. At each non-terminal node, a question is asked. The answers to the question are represented by the node's outgoing

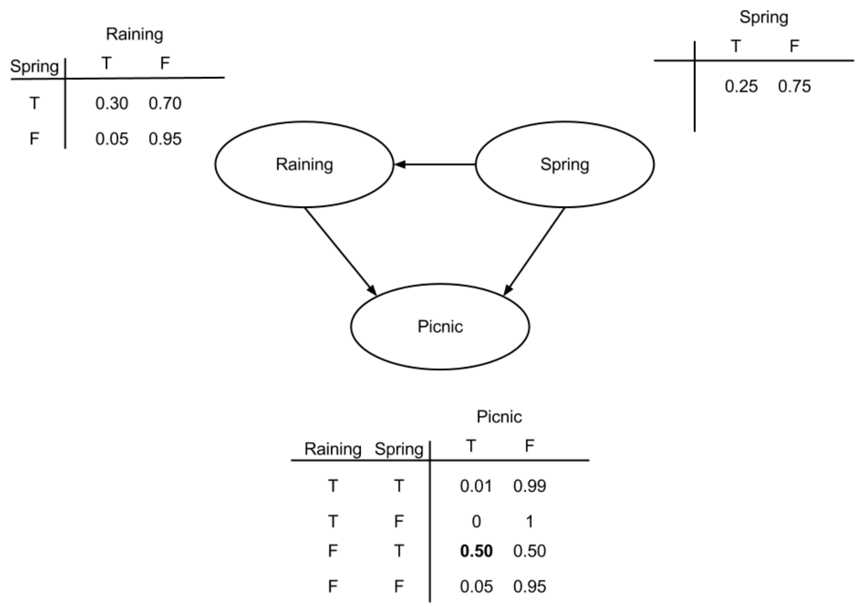

Figure 7.2: The simple Bayesian Network augmented with the probability functions for each node.

edges. The tree is traversed in this fashion until a terminal node is reached. The terminal node contains the final conclusion. In a classification context, each non-terminal node is labeled with an attribute, each edge is the value (or range of values) for that attribute, and each terminal node is a class. Each attribute can only appear once in the tree.

Figure 7.3 shows a decision tree that may be generated for the picnic example discussed in Section 7.2. Note that once the decision tree is built, reaching a terminal node is fairly trivial.

7.3.2 C4.5

C4.5 will recursively build a decision tree by continually splitting the dataset on a single attribute[24]. The splitting attribute is determined by the normalized

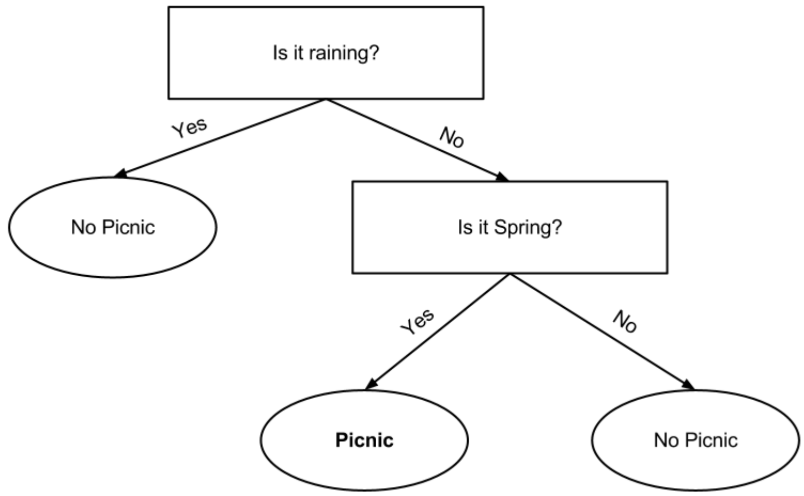

Figure 7.3: A simple decision tree trying to answer the question of whether or not to go on a picnic.

information gain (Kullback-Leibler divergence) and becomes a node in the tree and the possible values for the attribute become edges. Each subtree is then recursively built using only the data where the splitting attribute takes the value given by the incoming edge. The algorithm has two stopping conditions. First, when all the data has the same class. In which case a single node tree will be constructed that contains the class. Secondly, when there are no more attributes or when the information gain from splitting on each attribute is below a threshold. In this case, a single node tree will be constructed which contains the plurality class.

7.4 K-Nearest Neighbors

k-Nearest Neighbors (KNN) is simple and effective classification technique[7]. While training, the classifier remembers the entire training set. During the clas-

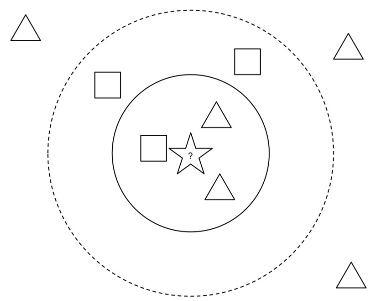

Figure 7.4: A simple example of KNN. If $k = 3$, then the query point (the star) will be classified as a triangle. However, if $k = 5$ then the query point will be classified as a square.

sification phase, the classifier will find the k nearest neighbors to the query point. The predicted class is simply the plurality of the k nearest neighbors. Figure 7.4 shows an example of k-Nearest Neighbors with a simple search space.

7.5 Support Vector Machines

Support Vector Machines (SVMs) are considered one of the best off-the-shelf classification techniques[4]. When training, SVMs use hyperplanes to partition the data into surfaces based off of the different classes of the training examples. When classifying, the SVM will find which surface the query point falls on and give that class to the point. SVMs will try and choose the partitioning hyperplane to maximize the margin between the two groups of data. Depending on the implementation, the SVM may choose the optimal partition or just an approximation.

Figure 7.5 shows a simple example of a linear binary SVM. Note that the partition line is chosen to maximize the distance between the triangles and squares. The query point (the star) falls into the squares' partition and is therefore clas-

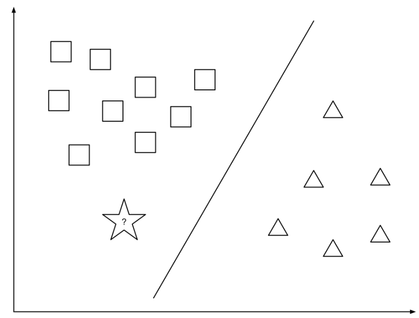

Figure 7.5: A simple example of a support vector machine. The SVM chose a partition that maximizes the margin between the squares and triangles.

sified as a square.

7.5.1 Sequential Minimal Optimization

Sequential Minimal Optimization (SMO) is an efficient algorithm for solving SVMs invented by John Platt in 1998[21].

7.6 BPNB

BPNB is a method developed by Chu[3]. It is based off of Naive Bayes, except the relative probability of each feature is accounted for.

Let c be a specific class in the set of possible classes C. Let d be a document composed of a vector of n features, $d = (\mathrm{f}_1, \mathrm{f}_2, ..., \mathrm{f}_n)$. BPNB states that the probability of observing class c given document d can be represented as:

$$Pr(c|d) = Pr(c) \cdot \prod_{i=1}^{n} g(\mathrm{f}_i, c) \tag{7.3}$$

Where $g(\mathrm{f}_m, c)$ is the weight of feature f_m in class c.

$$g(\mathrm{f}_m, c) = \beta^{1-\frac{Pr(\mathrm{f}_m|c)}{Ave(\mathrm{f}_m)}}, 0 < \beta < 1 \tag{7.4}$$

$$Ave(\mathrm{f}_m) = \frac{\sum_{i=1}^{|C|} Pr(\mathrm{f}_m|c_i)}{|C|}, c_i \in C \tag{7.5}$$

7.7 WEKA

SPOONS utilizes several classifiers provided in the *WEKA Machine Learning Package*. WEKA is an open source package written under the GNU General Public License[9].

Chapter 8

Twitter API

All of the data data that SPOONS uses is obtained in real time using the Twitter Search REST API[30].

8.1 Rate Limiting

Twitter imposes a limit on the number of queries to the Search API. Twitter does not publish the official limit. However, our experiments suggest that SPOONS can query the API for all new Tweets once every two minutes without suffering from rate limiting.

8.2 Pagination

Twitter paginates the results from its search API. The maximum results you can get per page is 100, and each query can return at most 15 pages. Therefore when there are more than 1500 tweets generated per minute, SPOONS must do multiple search queries.

8.3 Query Anatomy

The typical structure of a Twitter API query is shown in Figure 8.1.

http://search.twitter.com/search.**json**?**q**=⟨query⟩&**rpp**=100& **result_type**=recent&**since_id**=⟨tweet id⟩&**max_id**=⟨tweet id⟩

Figure 8.1: The structure of a typical query to the Twitter API.

The parameters are:

json: Twitter can supply the result data in either ATOM or JSON format. Testing with both have shown that the ATOM results are less consistent and provide less data. Because of the more accurate information returned from the JSON API, we are able to write more efficient queries. Using the ATOM API, we could query Twitter only once every five minutes; as opposed to every two minutes with the JSON API.

q: The search query. Twitter supports some advanced search features such as conjunction and negation.

rpp: "Results Per Page". Twitter paginates the responses from the Search API. SPOONS always uses the maximum pagination value to decrease the number of requests per hour and lessen the chance of being rate limited.

result_type: Twitter allows users to get results ordered by either relevance or time. Since we want to gather all tweets about our query, we choose to get the results ordered by time. In addition, the "since_id" and "max_id" parameters do not work when results are sorted by relevance.

since_id: The id of the oldest tweet that should be returned. This is not a hard limit, but provides a nice starting point.

max_id: The id of the most recent tweet that should be returned. It may seem counter-intuitive to provide a cap on the most recent tweet, when one wants to query for all of the most recent tweets. However when a query spans across more than 15 pages, it will need to be broken into a new query restarting at the first page. In this situation, not providing an upper limit will include new tweets outside of the original search scope. This can result in tweets are forever lost to us.

8.4 Result Anatomy

Figure 8.2 shows the result from the query "eriq netflix". Notice that some fields, like the `geo` field, can be null. Also note that the API incorrectly guessed the language of the tweet as Danish.

```
{
    completed_in: 0.012,
    max_id: 298199940868489200,
    max_id_str: "298199940868489216",
    page: 1,
    query: "netflix+eriq",
    refresh_url: "?since_id=298199940868489216&q=netflix%20eriq&result_type=recent",
  - results: [
      - {
            created_at: "Sun, 03 Feb 2013 22:43:00 +0000",
            from_user: "eriq_augustine",
            from_user_id: 238374031,
            from_user_id_str: "238374031",
            from_user_name: "eriq",
            geo: null,
            id: 298199940868489200,
            id_str: "298199940868489216",
            iso_language_code: "da",
          - metadata: {
                result_type: "recent"
            },
            profile_image_url: "http://a0.twimg.com/sticky/default_profile_images/
                                default_profile_0_normal.png",
            profile_image_url_https: "https://si0.twimg.com/sticky/default_profile_images/
                                      default_profile_0_normal.png",
            source: "&lt;a href=&quot;http://twitter.com/&quot;&gt;web&lt;/a&gt;",
            text: "eriq love netflix",
            to_user: null,
            to_user_id: 0,
            to_user_id_str: "0",
            to_user_name: null
        },
    ],
    results_per_page: 100,
    since_id: 0,
    since_id_str: "0"
}
```


Figure 8.2: A JSON result from the Twitter Search API

Part 3

SPOONS Architecture

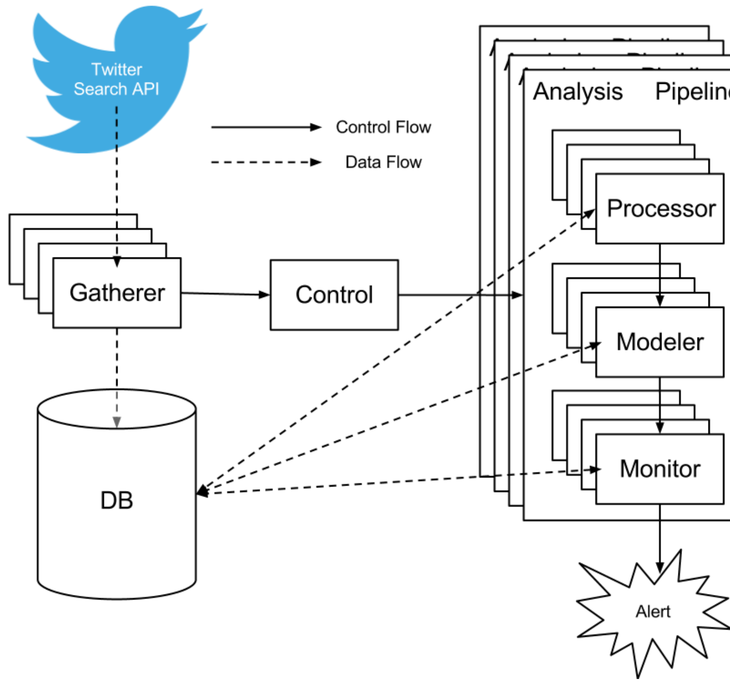

Figure 8.3: The flow of control and data through the SPOONS framework system.

There are multiple levels of architecture within SPOONS that need to be discussed. There is the Framework Architecture (Figure 8.3) that describes the relations between the different pieces of the framework; the Server Architecture (Figure 9.2) that describes the layout of the different servers involved in the SPOONS system; and the Distribution Model which describes how tasks are distributed between the different servers.

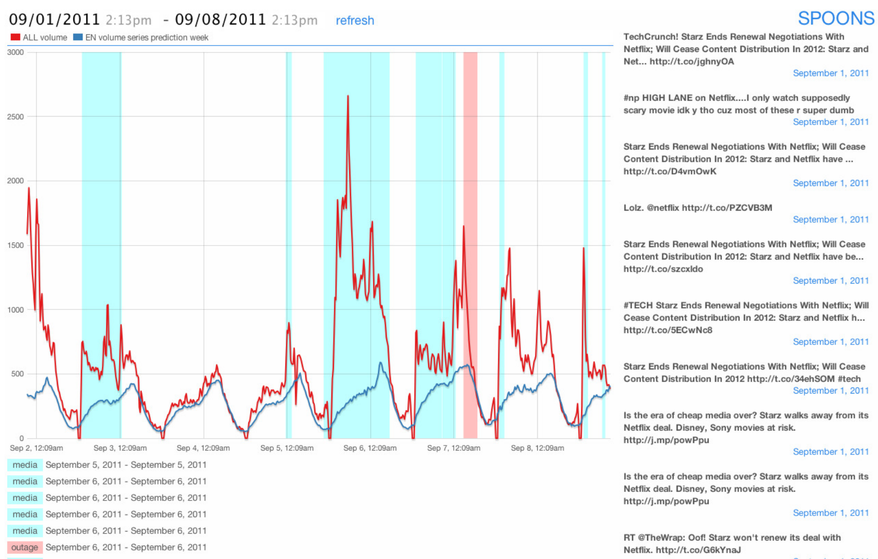

Figure 8.4: The web UI for SPOONS.

Chapter 9

Framework Architecture

This section describes the architecture of the SPOONS framework. The SPOONS framework includes all pieces of SPOONS that take the data from gathering all the way through to final analysis.

9.1 High Level Solution

The general solution taken by SPOONS consists of four main steps:

Collect: Tweets are collected from Twitter.

Process: The tweets are converted from plain text to some form of information that can be analyzed.

Model: Use the information generated from the previous step to build a mathematical model of the information. Use past information to predict what the current model of the data should look like.

Compare: Compare the two models generated in the previous step. A significant divergence means that there is anomalous traffic.

9.1.1 Framework Overview

Figure 8.3 shows the flow of control and data through the SPOONS framework. Data comes into SPOONS in the form of Tweets collected by the Gatherers, and leave SPOONS in the form of alerts generated by the Monitors.

Gatherer

Gatherers are responsible for collecting documents from a specified data source such as the Twitter Search API.

Database

After the tweets are gathered, they are placed in the database. In addition to storing just tweets, the database also stores configuration data, intermediary calculations, and the results of the Analysis Pipelines.

Control

The Control is responsible for controlling the SPOONS server. It maintains data structures with all of the Gatherers and Analysis Pipelines. It is also responsible for communication with other servers in the SPOONS cluster.

Processor

Processors are data transformation utilities that takes raw data and puts it in a form that other components can use.

Modeler

Modelers are responsible for building a mathematical model of the data and can be split into two groups: **Predictors** and **Counters**. Predictors build a predictive model of the data. Counters to build a model of the data that was actually gathered by the system.

Monitor

Monitors take the models produced by the Predictors and Counters and compares them. The Monitors are responsible for making the final decision on about a period of time being anomalous.

9.2 Gatherers

The data enters SPOONS at the Gatherers. The Gatherers run periodically (for Twitter, every two minutes). Gatherers are asynchronous and not dependent on any other part of the framework. There may be multiple different Gatherers running on the same machine. Gatherers are abstracted to be able to gather data from any source. Once the Gatherers get their data, they place the data in the database and notify the Control that there is new data available to the system.

9.2.1 Twitter Holes

It is worth noting that sometimes the Twitter Search API fails to return any data. We have not discovered the cause of this, but Twitter does not report any errors. For unspecified amounts of time the Twitter API will just report zero

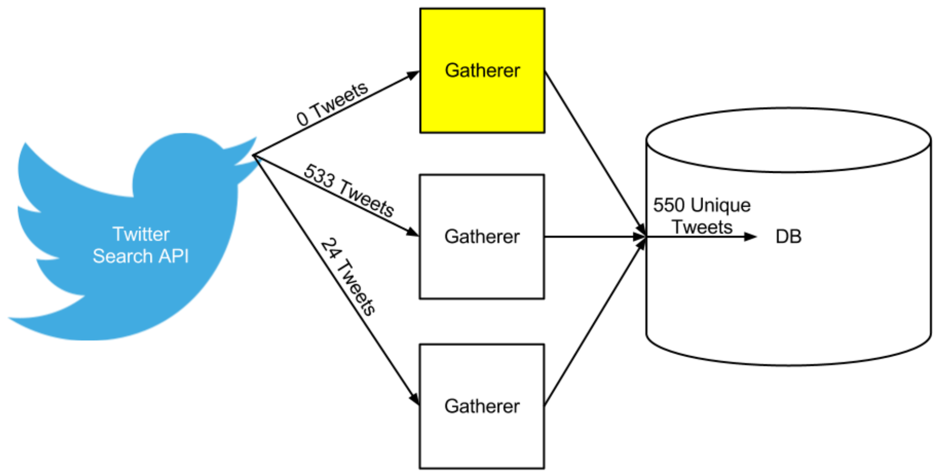

Figure 9.1: One server in a hole is covered by two other gathering servers.

new tweets. We call these dead zones "holes". We have found that a query from a different IP usually does not experience the same hole. To counteract holes, we run Gatherers on multiple servers and resolve uniques upon insertion into the database.

9.3 Processors

Processors are responsible for processing or transforming data before it goes into the analysis pipelines.

- Classifier Processors: There exists a Processor for every tweet classifier used in SPOONS (see Chapter 4. Because of the high number of classifiers used, these constitute the majority of Processors and form the largest unit of work in SPOONS. These Processors classify every tweet into one of the nine tweet categories discussed in Section 14.

- Author Processors: The Author Processors extract the author of tweets and try to establish which authors are credible. These Processors are outside the scope of this work and are discussed in other work[6].

- Valence Processors: The Valence Processors assign a numeric "happiness" score to every tweet. How that score is produced is outside the scope of this work (see Section 25.2).

- Document Frequency Processors: The Document Frequency Processors maintain term frequencies and inverse document frequencies for the collection of tweets in SPOONS.

Unlike most parts of the analysis pipeline, Processors are a shared resource. That is, multiple analysis pipelines invoke the same Processors. However, it does not make sense to restart the processing once it is started, or to start another instance of the same Processor for the same data. Processors have a finite amount of data to process and may be cumulative. To make sure that no redundant work is done, Processors are singleton. When multiple threads call into a Processor to do work, the Processor will block all incoming threads until the work is complete. Then, the Processor will release all of the threads that requested the work. This model allows all the analysis pipelines to share the same Processor without any redundancies.

9.4 Analysis Pipelines

An Analysis Pipeline (also called Analysis Method) is the analytical center of the SPOONS framework. The pipeline aggregates multiple tasks that it needs to run on the data.

An Analysis Pipeline typically starts with running any number of Processors on the data. Then, the pipeline invokes modelers on the data from the Processors. These modelers typically build models for the actual data coming into SPOONS as well as predictive models. Finally, the pipeline invokes tasks that assess the models produced in the previous step and decides whether or not there is an anomaly.

Every Analysis Pipeline gets its own thread, and there is no interdependence between the different pipelines. Currently, SPOONS usually runs more than 20 Analysis Pipelines at a time.

9.5 Tasks

Tasks are the core unit of computation in SPOONS. Almost everything that can be "run" is a child of the Task base class. Every Task gets its own thread, and callers into the Task may request that the task block the calling thread until the Task is complete.

Tasks are singleton with respects to the leaf child class. Therefore there are many tasks, but every task is unique. We do this by enforcing that the class name is unique upon construction. The uniqueness of tasks is very important to SPOONS distribution model that will be discussed in Section 9.9.

9.6 Modelers

Modelers are Tasks that are responsible for building a mathematical representation for the data.

9.6.1 Predictors

Predictors build a predictive model of the data. For example, we have noticed that tweet volume tends to be periodic day-to-day and week-to-week. Therefore, a Predictor may model that prediction by guessing that the volume in the future will be the same as it was the previous week or day.

9.6.2 Counters

Counters attempt to build a model of data that was actually gathered by the system. Going with the previous example, the Counter for modeling tweet volume would simply count the number of tweets gathered for a period.

9.7 Monitors

Monitors take the models produced by the Predictors and Counters and compares them. The Monitors are responsible for making the final decision about a period of time being anomalous.

9.7.1 Auto-Tuning

Monitors take anywhere from two to six tuning parameters. To find the best set of parameters, the Monitors can automatically run themselves on a training set and search the space of all possible parameters. They then keep the parameters that result in the best score.

9.7.2 Resistance

A monitor's "resistance" is its tendency not to move into or out of an alert state. The resistance is the number of normal or abnormal observations it needs to be trigger a state change. Monitors are given resistance because otherwise outliers could cause monitor to rapidly switch between alert and normal states. There currently are three different methods of observing resistance. The method of resistance as well as the resistance thresholds can also be auto-tuned.

Fighting Resistance

Parameter	Description	Restrictions
A	The number the counter must reach to enter an alert state.	$A > 0$
R	The number the counter must reach to enter a normal state.	$R > 0$

Fighting resistance counts every time that there is a normal period as a +1, and every time there is an anomalous period as a -1. If the counter reached $-A$, then the monitor is put into an alert state. If the counter reaches R, then the monitor is put into a normal state.

Continuous Resistance

Parameter	Description	Restrictions
A	The number the counter must reach to enter an alert state.	$A > 0$
R	The number the counter must reach to enter a normal state.	$R > 0$

Continuous resistance must get A continuous anomalous observations to enter an alert state, and R continuous normal observations to enter a normal state.

Window Resistance

Parameter	Description	Restrictions
W	The window size.	$W > 0$
C	The number of anomalous observations necessary for an alert state.	$0 < C <= W$

Window resistance remembers W previous observations as being normal or anomalous. If the number of anomalous observations is or exceeds C, then an alert state is declared. Otherwise, the monitor stays in a normal state.

9.7.3 Smoothers

The monitors have a chance to smooth the data before it gets analyzed. As with resistance methods, different smoothers and smoothing parameters can be auto-tuned.

No Smoother

Do not smooth. If this smoother is put into the parameter search space, then the effects of no smoothing can be seen.

Moving Mean Smoother

Parameter	Description	Restrictions
W	The window size.	$W > 0$

Any value becomes the mean of that value and the $W - 1$ values that precede it.

9.8 Control

The Control is the center of a SPOONS instance. It handles the flow of all control and has the ability to start and stop any task or pipeline on demand. It holds references to all the threads for the Gatherers and Analysis Pipelines. The Control handles all the setup and tear down in the system.

There are different types of Controls that decide the behavior SPOONS on each respective server. The Control is singleton with respects to the base class. Therefore, only one instance of any type of Control can be active at any given time.

The Control is very careful to never allow anyone to own a reference to the currently running Control. All requests to the Control are made statically to the “Control” base class. The base class will then forward the request onto the specific instance of Control. We do this so that the rest of the SPOONS system will never know what kind of Control is currently active. So we can switch a server between different roles without restarting the system or notifying any other components of the SPOONS system.

All Controls will always run the entire slew of Gatherers.

9.8.1 Master Control

The Master Control is the Control that is responsible for the controlling SPOONS when it is in distributed mode. The Master Control maintains information on all the active worker servers. It will send the worker servers messages to tell them what work to do.

The Master Control maintains “shallow execution” of every pipeline in the system. This means that this control will run each pipeline, but then distribute work for each pipeline as it is created.

9.8.2 Worker Control

Worker Controls do not take any initiative to run any tasks. Instead, they just wait for a Master Control to tell them what to do.

9.8.3 Single Control

The Single Control is for a SPOONS instance that wants to run on a single server.

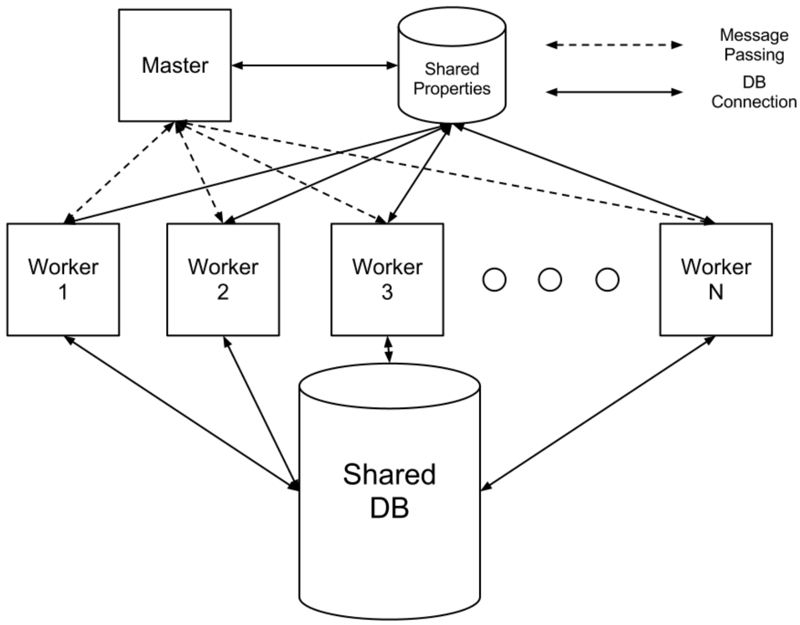

Figure 9.2: The server architecture of the SPOONS system.

9.9 Distributed Model

As discussed before, SPOONS is a multi-server system (Fig 9.2). The SPOONS system uses the master/worker paradigm with a single master and N workers.

All of the servers share two primary resources: the primary database and a NoSQL property store. When a master or worker comes online, it inserts and entry for itself into the shared property store. If the new server is a worker, it will alert the current master about its existence; and visa-versa if the new server is a master. In addition, all workers are required to heartbeat to the master every 15 seconds and the master heartbeats to the workers every 15 seconds. Using this system, the master always knowns about all of the workers and the worker always knows about the current master. When a server misses three heartbeats, the server expecting that heartbeat assumes that the server has gone down.

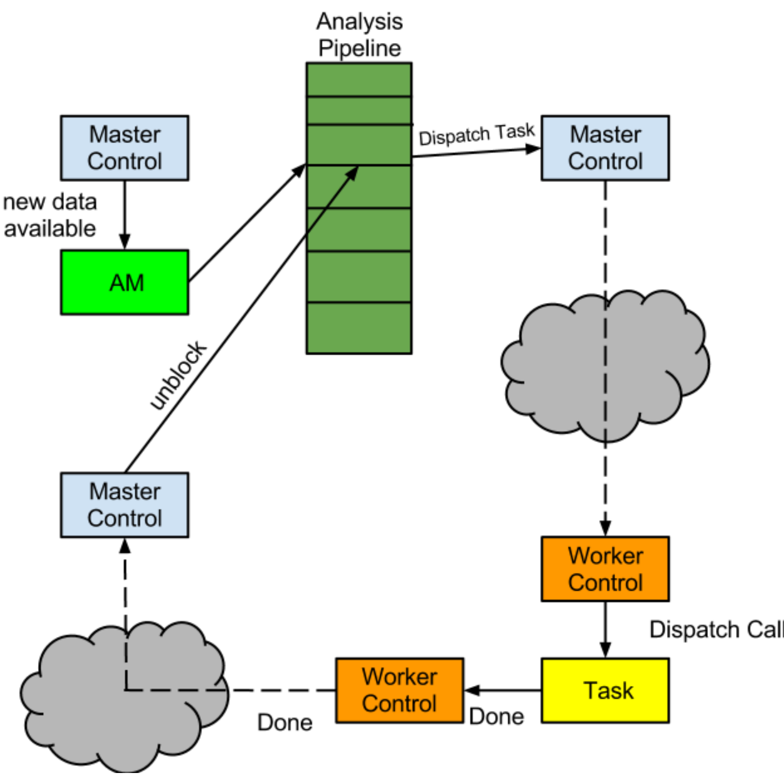

Figure 9.3: The control flow for distributable tasks.

9.9.1 Distribution Assumptions

The SPOONS distribution model relies on two assumptions about the system: every server contains exactly the same data in memory and every Task can be uniquely referenced.

Same Data

SPOONS assumes that every server will have the same data in memory on every server. This means that not only does every server need to have the same data structures in memory, but also that every server needs to have the same classes instantiated. The only exception to this assumption is the Control. Depending on the role of the server, a different Control will be instantiated. Because of this assumption, we do not have to worry about active replication between servers or a worker being asked to do work that requires a class that is not instantiated.

Uniquely Referenced Tasks

As stated in Sec 9.5, Tasks are the basic unit of work inside SPOONS. When a worker is told to execute some work, it is being asked to execute a specific task with specified parameters. Therefore, Tasks need to be able to be referenced by a key that can be serialized and sent over the wire from the master to the worker.

9.9.2 Distributable Tasks

Distributable Task is a subclass of Task that provides some of the distribution mechanism for Tasks. When a task is to be distributed, the Distributable Task calls into the Control and requests that the Control distributes it. The next step varies depending on the type of Control that is active:

Master Control

The task distributing control flow is described in Image **??**. A Master Control will take the pause the calling thread and send a message to a selected worker[1] telling to run the task with given parameters. The message that goes to the worker just contains the task's unique identifier and the parameters to the task's run. When the task is complete, the Worker Control will send the Task's return status back to the Master Control. When the master receives a message from the worker that the requested task has completed its run, it will resume the original calling thread and have it return with the return status given by the worker.

Worker Control

If a Worker Control receives a task, then it is being asked to distribute a task that is already being distributed. We consider this a violation of the framework and will throw an error.

Single Control

A worker will just call back into the task and tell it to run itself.

9.9.3 Shared Properties

As previously stated, all servers must maintain a consistent in-memory view of the system. This can be troublesome if a Task needs to maintain cumulative

[1]The current scheduling algorithm chooses the worker that has the fewest tasks currently assigned to it.

settings or member datum. Not only will this data need to be consistent on all the servers, but it also needs to maintain this data between starts and stop of the system. An Analysis Pipeline should be able to the stopped for an arbitrary amount of time and then restarted without losing data or its place.

To enforce these restrictions, we use a shared property store. The shared property store is a MongoDB server. Whenever a Task needs to store member datum, it places it in the shared store. Therefore, any server may access this data. A Task can first be run on Server A and then on Server B. Because it stores the necessary information in the shared property store, Server B can have all the information gained from the run on Server A and not lose any positional information.

In addition to storing shared properties, the shared property store houses information on every active server. When a server comes online, it queries the property store to find all the other active servers and inserts itself into the store. If a server fails to heartbeat, then the servers that still live will remove the entry that server from the property store.

Chapter 10

Database

SPOONS is backed by a MySQL database. SPOONS currently uses 225 tables and 35 stored procedures. The 225 tables are further divided into six different categories that are used in different stages of the analysis pipeline. In addition to tweets being stored in the database configuration data, intermediary calculations, analysis results, and final alerting decisions are stored in the database. Keeping all of this data allows the users to look back at any point in time for reference or debugging. The majority of the stored procedures are a bridge between the UI and database. The stored procedures provide a consistent interface for the UI without the need to underlying details.

The database uses naming and schema conventions to maintain organization on its tables. The naming and schema conventions allow different components of the Analysis Pipeline to be interchanged without any need to change/reprocess the data. In addition the conventions allows the UI to represent new tables without the need for specifying them.

10.1 Tables and Schemas

Each stage in an analysis pipeline generally stores some information in the database. Because each stage generally deals with similar types of data, these tables are considered to be in the same group. We enforce group membership using hints in the table names. For example, the table name "RESULT_EN_class_heuristic_bayes_net" gives five hints as to the type of the table.

1. RESULT - Marks this table as a result table. This means that it is guaranteed to be shown in the UI.
2. EN - The language of the tweets that were input into this method.
3. class - Indicates that this these results are output from a tweet classifier.
4. heuristic - States that the type of classifier used was a heuristic classifier.
5. bayes_net - The name of the classifier used.

Using all of these hints, the UI can then ask for data for specific types of tables (eg. all result tables that are for English tweets).

The six different top level categories that SPOONS recognizes are:

1. CALC - These are intermediary tables in analysis pipelines. CALC tables are typically only used when large sets of past data are needed for cumulative models. They are never shown to the UI.
2. CONFIG - Contains information that analysis methods used to configure themselves before runs. These tables have been mostly replaced with the shared property store (see Section 9.9.3).

3. DATA - Raw input data. These tables are generally the output from the Gatherers.

4. META - Contains information that is not analyzed, but required by the system. For example, the different classes that the classifiers use along with descriptions of each class.

5. RESULT - These tables are output from some analysis pipeline. They are guaranteed to be shown in the UI.

6. TEST - These tables are used for debugging and development. They are never shown in a user-facing UI, however may be shown in development UIs.

The full schema for select tables are described in Appendix A.

10.1.1 Data Flow

The flow of data through the different types of tables is described in Figure 10.1. The data originates from the Gatherers and is moved into DATA tables. Information from DATA, CONFIG, and META tables are analyzed and placed in either CALC, RESULT, or TEST tables. At a later time the data from CALC tables is further analyzed and the results are placed in a RESULT table.

10.1.2 Tweets Table

As the most used and important table in the database, the table that houses all of our tweets, “DATA_tweets”, gets special attention.

The tweets table contains ten attributes which are described in Table **??**.

Name	Description
id	An auto-incremented primary key.
twitter_id	The unique id Twitter gives a tweet.
published	The epoch time that the tweet was posted according to Twitter.
content	The raw content of the tweet.
source	Information on where the tweet was posted from (eg. from a third party app).
lang	The suggested language of the tweet.
author	The author of the tweet.
frame_id	The frame that this tweet falls into, has an index on it.
place	Information on where the tweet was posted from. This is a JSON structure and may contain fields such as "city" and "state".
geo	Geographical coordinates of place.

Table 10.1: The database attributes used to describe tweets.

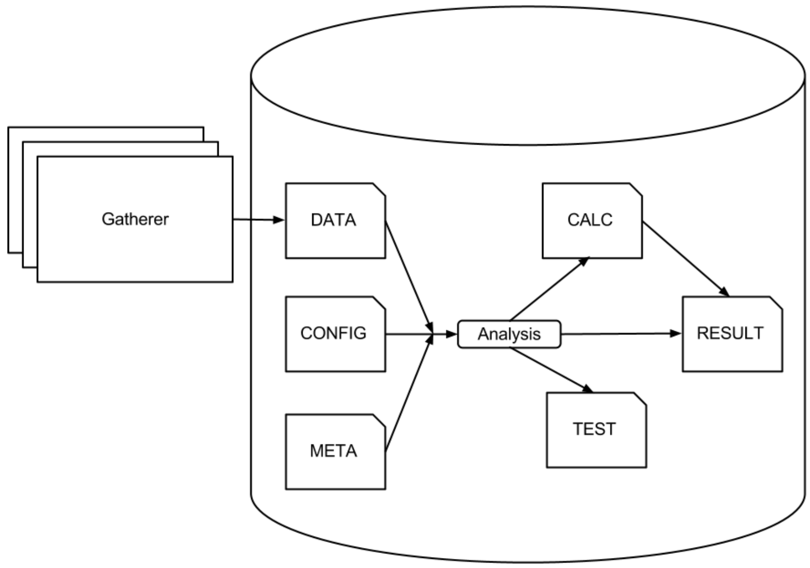

Figure 10.1: The flow of data through the different types of tables in SPOONS.

Frames

Inside SPOONS, we use a "frame" as the atomic unit of time. Currently, a frame corresponds to a minute. Bucketing the tweets into frames allows us to gain a natural aggregation and smoothing. It also provides a natural index. Maintaining an index on *frame_id* allows quick retrieval of time series data which is the primary task of SPOONS. Because insertions are generally chronological, insertions are also quick and do not require a rebuild of the B-Tree index[1].

10.2 UI Stored Procedures

In addition to utility procedures, the database holds many stored procedures used by the UI. This keeps the UI fairly stable in the face of database changes.

Schema Name	Required Columns
Volume	start_frame, value
Volume Prediction	start_frame, prediction
Valence	start_frame, value
Valence	start_frame, prediction
Class	start_frame, undecided, media, neutral, snafu, watching, response, complaint, refuse_to_rate, happy
Group	start_frame, media, bad, other

Table 10.2: The different types of schemas that the UI looks for in RESULT tables.

10.2.1 Expected Schemas

The UI Stored Procedures look for 6 distinct name/schema combinations all of which are required to be RESULT tables. The different schema requirements are shown in Figure 10.2, and described below:

Volume This schema is for tables that contain time series information about tweet volumes. This includes tables that holds the time series for the total Netflix-related Twitter traffic.

Volume Prediction These tables contain time series that are predictive models of Netflix-related Twitter traffic.

Valence These tables contain time series for estimates of the current sentiment about Netflix.

Valence Prediction These tables contain time series that are predictive models of the sentiment about Netflix.

Class These tables contain time series for the volume of tweets that were classified into each of the nine categories described in Section 14.

Group These tables contain time series for the volume of tweets that were classified into each of the thee different groupings described in Section 14.1.

The stored procedures will further divide the tables by language. The currently recognized languages are English, Spanish, and Portuguese.

Part 4

Classifiers

Chapter 11

Why Classification?

Classification helps discover Netflix service outages by differentiating between different types of Twitter traffic.

Figure 11.1 shows the normal pattern of Netflix-related Twitter traffic. The peaks appear at around 7pm PST and the valleys are around 2am PST. This kind of pattern is very regular and repeats weekly during normal times. However where there is some sort of event, the traffic develops spikes. Figure 11.2 shows a period with two anomalous spikes. However, sampling tweets from the different spikes hints that the causes for the two different spikes are very different. Figure 11.3 shows tweets sampled from each spike. The left spike is composed mostly of tweets indicating that Netflix is experiencing a service outage. The right spike however, is composed mainly of tweets linking to a news article about Netflix. Therefore, we see that not only service outages generates spikes in Netflix-related Twitter traffic.

This is where classifiers become useful. If tweets can be placed into different classes according to the type of traffic that they generate, then the different types

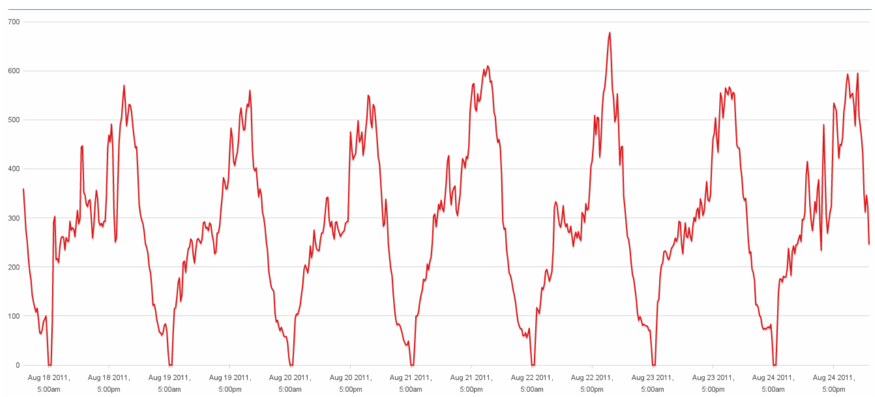

Figure 11.1: A weeks worth of Netflix-related Twitter traffic. Notice the daily periodicity.

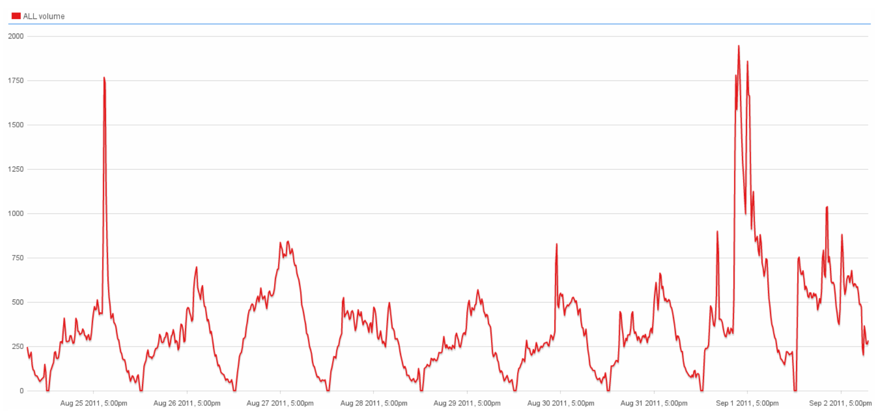

Figure 11.2: Netflix-related Twitter traffic with two different anomalies.

of traffic can be differentiated. Figure 11.4 shows the result of classifying the tweets and then building time series of the classes respective traffic. It becomes obvious that the spike on the left is caused by outage related traffic and that the spike on the right is caused by media related traffic.

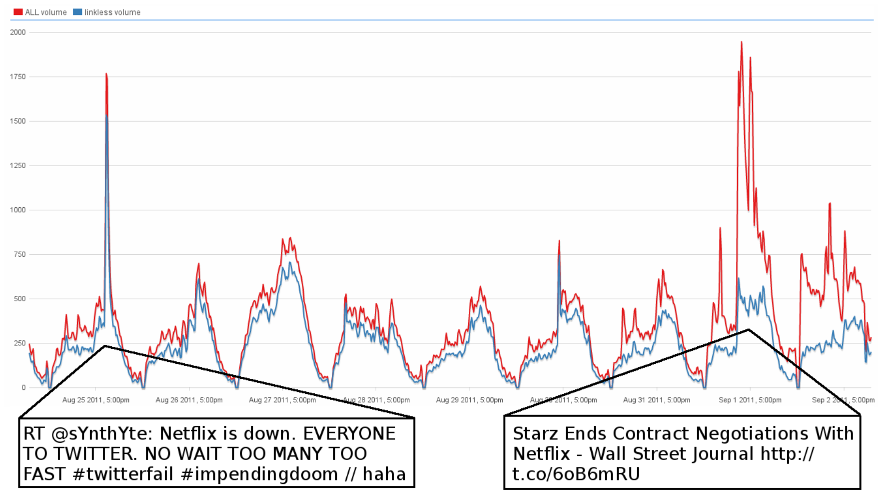

Figure 11.3: The same traffic shown in Figure 11.2, with an additional line showing Netflix-related Twitter traffic that does not contain a URL.

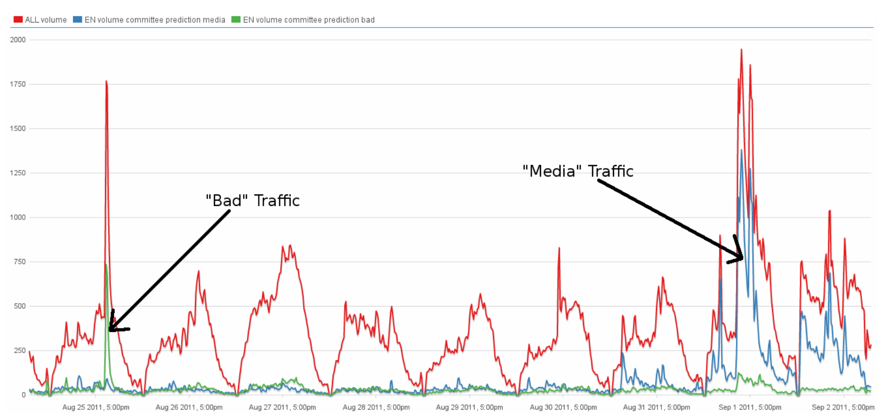

Figure 11.4: The same traffic shown in Figure 11.2, with two addition lines: the volume of tweets classified as "Bad" and the volume of tweets classified as "Media".

Chapter 12

Classification Roadmap

The steps that SPOONS takes to use classification to detect service outages is as follows:

1. Observe Netflix-related Twitter traffic and observe the different classes that the tweets fall into.
2. Build a training set biasing anomalous traffic.
3. Classify incoming tweets.
4. Group classified tweets according to the type of traffic that class produces.
5. Establish the best classifiers.
6. Use the best classifiers in an Analysis Pipeline.
7. Observe the differences between the total traffic and the classified traffic.
8. Declare an outage when the two traffics diverge significantly.

Chapter 13

Fitting Into The SPOONS Framework

Although the classifiers can be used at any stage in an analysis pipeline, the classifiers are usually implemented as a Processor. It takes in a range of tweets and produces a classification for each tweet.

Chapter 14

Tweet Classes

After observing Netflix-related Twitter traffic, it was discovered that tweets fall into at least one of nine different categories.

- `Media` – Relate to a media story about Netflix.
- `Snafu` – Talk about a Netflix outage.
- `Complaint` – Complain about Netflix.
- `Happy` – Express joy about Netflix.
- `Neutral` – Neutral observation or comment about Netflix.
- `Watching` – Updates on what the user is currently watching.
- `Response` – Neutral response to another user in a Netflix-related conversation.
- `Refuse To Rate` – Used for tweets that we refuse to rate entirely (usually tweets that are in a different language than the training set).

Class	Tweet Example
`Media`	Netflix Now Available Through Facebook - http://bit.ly/ffpBHH - [Geeky Gadgets]
`Snafu`	And netflix is broken. Why is this happening to me.
`Complaint`	netflix keeps taking little things i like about the site away...Why?
`Happy`	Netflix :)
`Neutral`	about to download this netflix free trial
`Watching`	Watching Family Guy on Netflix
`Response`	@BeehiveBlog Both good movies. I think I'll put on the netflix list.
`Refuse To Rate`	en serio, QUIERO pagar por algo como Netflix, DEJARME pagar

Table 14.1: The database attributes used to describe tweets.

- `Undetermined` – This class does not exist in the wild. It is used during classification as default for all tweets that don't match any other class.

Examples of tweets with their corresponding classes are shown in Table 14.1.

14.1 Tweet Groups

Because the goal of SPOONS is to detect anomalous times, it is useful to collapse the nine classes into three different groups that account for the different types of Netflix-related traffic.

- `Media`: Contains only the `media` class.
- `Bad`: Contains both the `snafu` and `complaint` classes.
- `Other/Normal`: Contains all other classes.

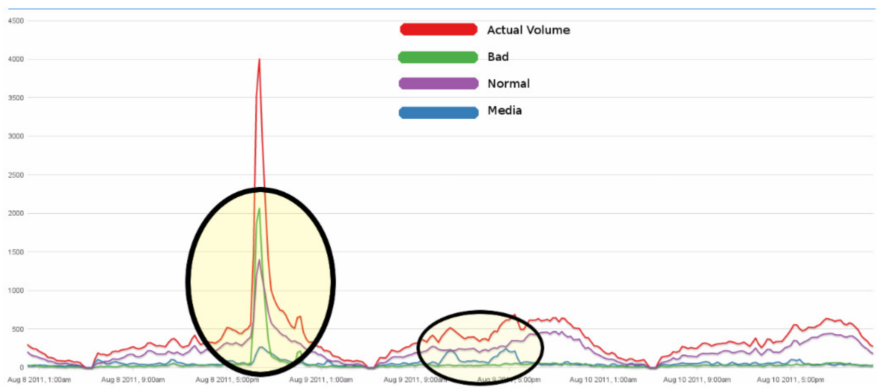

Figure 14.1: The different volumes for different tweet classes during an outage (left) and media event (right).

Figure 14.1 shows the amount of Netflix-related tweets during a Netflix outage and media event. During normal times, the `normal` traffic is responsible for the majority of the overall traffic. However during outage and media events, we see that the `bad` and `media` dominate the respective periods.

Chapter 15

WEKA Classifiers

SPOONS uses several classifier from the WEKA machine learning package[9]. All of these classifiers have been discussed in Section 7.

- Naive Bayes
- Bayes Net
- J48 (a method of generating a C4.5 decision tree[25])
- K-Nearest Neighbors
- SMO (Support Vector Machine trained with Sequential Minimal Optimization[22])

Chapter 16

Non-WEKA Classifiers

In addition to the WEKA classifiers, SPOONS uses two classifiers implemented from scratch. Because of low performance and inflexible API, the WEKA classifiers are being reimplemented. As of now, only Naive Bayes has been reimplemented. The other classifier implemented from scratch is a BPNB classifier which is discussed in Section 7.6.

Chapter 17

Text Processing

Before the tweets are classified, they are processed. During processing, the text is transformed to make classification of the text easier. Standard text operations like stemming, stopword removal, and case normalization; as well as Twitter and Netflix specific operations like hashtag and movie title recognition are preformed. After the text is processed, it is split into unigrams to be used as features in the classifiers.

17.1 Text Filtering

Before the input text is split into features, it goes through heavy pre-processing. The text filtering involves normalizing the case, remove extra characters, and replacing special features.

17.1.1 Link Replacement

The first step in processing the text is to replace links. Following a link may provide information about a tweet, however the link text of the link itself provides no information. The presence of a link, however, can provide information about a tweet.

17.1.2 Twitter Specific Symbols

Tweets often contain several special symbols specific to tweets.

RT

"RT" stands for "re-tweet". It means that the posted tweet is a repost of a tweet made by another user. This symbol contains no reference to the original post. "RT" usually appears at the beginning of the tweet. For example, after the comedian Conan O'Brien posted the following tweet:

> If I'm ever a ghost, I hope the person I haunt has Netflix.

There were hundreds of identical tweets that said:

> RT: If I'm ever a ghost, I hope the person I haunt has Netflix.

#

In Twitter, a "#" (pronounced "hashtag") is a reference to some topic in Twitter. Users can search for tweets by hashtag and see the collection of tweets supposedly about the same topic. A hashtag does not have to reference a pre-existing topic.

An “@” in Twitter, simply pronounced “at”, is a reference to another Twitter user. A reference to a user will alert that user about the posted tweet. For example, the following tweet will reference my Twitter account.

Hi there, @eriqaugustine

17.1.3 Emoticon Parsing

Emoticons are parsed out and replaced with meta words. SPOONS emoticon parser was written by Ryan Hanarkis and Allen Dunlea as part of a project for Graduate Artificial Intelligence. Emoticons provide a plethora of information about a tweet. Sarcasm aside, an emoticon can surmise the sentiment of an entire tweet.

17.1.4 Title Replacement

Because our tweets are always about Netflix, a television show and movie streaming service, titles are a common occurrence. However, movie and show titles often contain words that can be detrimental to our analysis. For example, “Death At A Funeral” is the title of a movie, but contains two words that have very negative connotations: “death” and “funeral”.

Without title replacement, the following tweet would be very difficult to classify:

Death at a Funeral is hilarious! #netflix

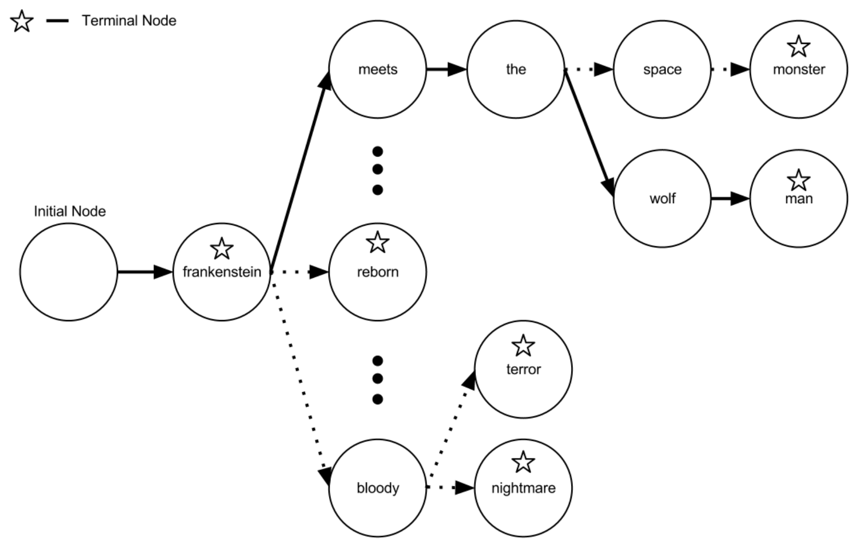

Figure 17.1: A sample trie of Frankenstein movie titles. The solid lines show what nodes the search for "Frankenstein Meets The Wolf Man" would traverse.

However after title replacement, the tweet becomes very easy to classify:

$\langle\$title\$\rangle$ is hilarious! #netflix

SPOONS contains a table that has over 50,000 movie and show titles on Netflix. The titles were gathered using the Netflix API. On startup, SPOONS will build a trie (prefix tree)[32] of all of the titles. In this trie titles can only be split on a word level, not on a character level. Therefore, moving to the next node consumes a single word. Searching for a title becomes a simple walk down the trie. If the walk of the trie ends on a terminal node, then a title is found. If not, then the trie is walked again from the beginning starting at the next word in the tweet. Figure 17.1 shows a sample walk in the title trie.

17.1.5 Stemming

Stemming finds the root of a word. This allows words to be categorized by their roots which decreases the number of unique words being evaluated and emphasizes linguistic patterns. This preprocessor uses Porter's stemmer for the English language [23].

17.1.6 Stop Word Removal

Stopwords, or words that carry little or no semantic information, are identified based on a static table of words mapped to levels. Stopwords are assigned levels which allow processes to use different sets of stop words. All words less than 3 character are also automatically considered stop words.

17.1.7 Punctuation/Non-English Character Removal

Removes all punctuation and characters not in the English alphabet. This simplifies word extraction and comparison.

17.1.8 Meta Words

Below is an overview of meta words that SPOONS recognizes:

⟨**$link$**⟩ Indicates the presence of a URL.

⟨**$emote:*$**⟩ Replaces an emoticon.

⟨**RT**⟩ Indicates that a tweet is a "retweet" (a repeat of another tweet).

⟨\$#\$⟩ Inserted when a "hashtag" is found in a tweet. The original subject of the hashtag is separated off into another word. E.g. "#Netflix" becomes "⟨\$#\$⟩ Netflix".

⟨\$@\$⟩ Inserted when a reference to another Twitter user is made. The user that is the subject of the reference is separated off into another word.

Chapter 18

Training Set

The classifiers were trained on a small set of **759** tweets which were pulled from from periods of both normal and anomalous traffic. Each tweet in the training set was manually classified by multiple researchers until consensus about the classification was reached. Because the goal is anomalous traffic detection, the training set over-samples the tweets from `media`, `snafu`, and `complaint`: categories. Table 18.1 documents the structure of the training set and shows the number of tweets classified into each of the eight categories. Tweets were allowed to belong to multiple classes because of posts like, "`I love netflix! Watching Law and Order online!`", which could be classified as both `happy` and `watching`.

Class	# Tweets	Class	# Tweets
`Media`	103	`Neutral`	66
`Outage`	158	`Watching`	135
`Complaint`	146	`Response`	30
`Happy`	147	`Undetermined`	48

Table 18.1: Overview of the Netflix-related Twitter post training set used to train classifiers in SPOONS.

Chapter 19

Evaluation

Each classifier is individually evaluated just on its ability to classify tweets against the training set. Each classifier varied two parameters: the type of filtering and the feature selection. The measure of accuracy is the percentage of correctly classified tweets.

Table 19.1 shows a summary of the results of the evaluation. The SMO classifier took the top three spots with a top accuracy of **.5750**. This is a decent accuracy, but much of the misclassification occurs between classes that don't matter as much when trying to identify the different types of classes. For example, it does not matter if a `watching` tweet is misclassified as a `happy` tweet. Both of those classes contribute to normal background traffic. Therefore the classifiers will be evaluated again, but the different classes will be compressed into their respective groups before the results are evaluated.

Classifier	Feature Selection	Text Filter	Accuracy
SMOClassifier	**TweetFSG**	**NoFilter**	**0.5750**
SMOClassifier	NGram	NoFilter	0.5726

SMOClassifier	NGram	EriqFilter	0.5618
BinaryNaiveBayesClassifier	NGram	NoFilter	0.5606
SMOClassifier	TweetFSG	EriqFilter	0.5546
NaiveBayesClassifier	NGram	NoFilter	0.5462
J48Classifier	NGram	EriqFilter	0.5414
BinaryNaiveBayesClassifier	NGram	EriqFilter	0.5414
J48Classifier	TweetFSG	NoFilter	0.5366
J48Classifier	NGram	NoFilter	0.5294
J48Classifier	TweetFSG	EriqFilter	0.5282
Non-Weka BPNBClassifier	NGram	EriqFilter	0.5210
NaiveBayesClassifier	NGram	EriqFilter	0.5198
BinaryNaiveBayesClassifier	TweetFSG	NoFilter	0.5186
BinarySMOClassifier	NGram	EriqFilter	0.5174
BinaryNaiveBayesClassifier	TweetFSG	EriqFilter	0.5150
BinaryJ48Classifier	TweetFSG	NoFilter	0.5126
BinarySMOClassifier	NGram	NoFilter	0.5126
BinarySMOClassifier	TweetFSG	NoFilter	0.5102
BinarySMOClassifier	TweetFSG	EriqFilter	0.5030
BinaryBayesNetClassifier	NGram	EriqFilter	0.4982
BayesNetClassifier	NGram	NoFilter	0.4970
NaiveBayesClassifier	TweetFSG	NoFilter	0.4970
BinaryBayesNetClassifier	NGram	NoFilter	0.4958
Non-Weka BPNBClassifier	NGram	NoFilter	0.4958
BinaryJ48Classifier	TweetFSG	EriqFilter	0.4922
NaiveBayesClassifier	TweetFSG	EriqFilter	0.4778

BinaryBayesNetClassifier	TweetFSG	NoFilter	0.4694
BayesNetClassifier	NGram	EriqFilter	0.4646
BinaryKNNClassifier	TweetFSG	NoFilter	0.4622
BinaryJ48Classifier	NGram	EriqFilter	0.4622
KNNClassifier	TweetFSG	NoFilter	0.4586
Non-Weka BPNBClassifier	TweetFSG	EriqFilter	0.4538
Non-Weka NaiveBayesClassifier	TweetFSG	EriqFilter	0.4526
BinaryJ48Classifier	NGram	NoFilter	0.4514
Non-Weka BPNBClassifier	TweetFSG	NoFilter	0.4454
BinaryKNNClassifier	NGram	NoFilter	0.4406
KNNClassifier	NGram	NoFilter	0.4382
BinaryBayesNetClassifier	TweetFSG	EriqFilter	0.4226
Non-Weka NaiveBayesClassifier	TweetFSG	NoFilter	0.4226
BinaryKNNClassifier	TweetFSG	EriqFilter	0.4130
Non-Weka NaiveBayesClassifier	NGram	EriqFilter	0.4082
KNNClassifier	TweetFSG	EriqFilter	0.4046
BayesNetClassifier	TweetFSG	NoFilter	0.3998
KNNClassifier	NGram	EriqFilter	0.3854
BinaryKNNClassifier	NGram	EriqFilter	0.3830
Non-Weka NaiveBayesClassifier	NGram	NoFilter	0.3553
BayesNetClassifier	TweetFSG	EriqFilter	0.3289

Table 19.1: Uncompressed Classification Results Summary

Classifier	Feature Selection	Text Filter	Accuracy
SMOClassifier	**TweetFSG**	**NoFilter**	**0.8583**
SMOClassifier	NGram	EriqFilter	0.8583
SMOClassifier	NGram	NoFilter	0.8499
SMOClassifier	TweetFSG	EriqFilter	0.8319
Non-Weka BPNBClassifier	NGram	EriqFilter	0.8271
BinarySMOClassifier	NGram	NoFilter	0.8259
BinaryNaiveBayesClassifier	NGram	NoFilter	0.8235
BinarySMOClassifier	NGram	EriqFilter	0.8235
BinaryNaiveBayesClassifier	NGram	EriqFilter	0.8199
BinarySMOClassifier	TweetFSG	NoFilter	0.8175
NaiveBayesClassifier	NGram	NoFilter	0.8175
Non-Weka BPNBClassifier	NGram	NoFilter	0.8151
BinarySMOClassifier	TweetFSG	EriqFilter	0.8139
J48Classifier	NGram	EriqFilter	0.8091
NaiveBayesClassifier	NGram	EriqFilter	0.8079
BinaryNaiveBayesClassifier	TweetFSG	NoFilter	0.7923
BinaryKNNClassifier	NGram	NoFilter	0.7899
J48Classifier	NGram	NoFilter	0.7887
J48Classifier	TweetFSG	EriqFilter	0.7875
KNNClassifier	NGram	NoFilter	0.7863
J48Classifier	TweetFSG	NoFilter	0.7839
NaiveBayesClassifier	TweetFSG	NoFilter	0.7803
BinaryNaiveBayesClassifier	TweetFSG	EriqFilter	0.7707
NaiveBayesClassifier	TweetFSG	EriqFilter	0.7671

KNNClassifier	NGram	EriqFilter	0.7623
Non-Weka NaiveBayesClassifier	NGram	EriqFilter	0.7611
BinaryKNNClassifier	NGram	EriqFilter	0.7575
BinaryKNNClassifier	TweetFSG	NoFilter	0.7527
KNNClassifier	TweetFSG	NoFilter	0.7479
BinaryJ48Classifier	TweetFSG	NoFilter	0.7419
BinaryKNNClassifier	TweetFSG	EriqFilter	0.7371
Non-Weka NaiveBayesClassifier	TweetFSG	EriqFilter	0.7371
Non-Weka NaiveBayesClassifier	TweetFSG	NoFilter	0.7335
KNNClassifier	TweetFSG	EriqFilter	0.7311
BayesNetClassifier	NGram	NoFilter	0.7263
Non-Weka NaiveBayesClassifier	NGram	NoFilter	0.7263
BinaryJ48Classifier	TweetFSG	EriqFilter	0.7059
Non-Weka BPNBClassifier	TweetFSG	NoFilter	0.6939
BinaryBayesNetClassifier	TweetFSG	NoFilter	0.6927
BinaryJ48Classifier	NGram	NoFilter	0.6879
BinaryBayesNetClassifier	NGram	EriqFilter	0.6855
BinaryBayesNetClassifier	NGram	NoFilter	0.6819
BinaryJ48Classifier	NGram	EriqFilter	0.6759
Non-Weka BPNBClassifier	TweetFSG	EriqFilter	0.6651
BayesNetClassifier	TweetFSG	NoFilter	0.6615
BayesNetClassifier	NGram	EriqFilter	0.6603
BinaryBayesNetClassifier	TweetFSG	EriqFilter	0.6567
BayesNetClassifier	TweetFSG	EriqFilter	0.6279

Table 19.2: Compressed Classification Results Summary

Table 19.2 shows a summary of the compressed results. After the classes have been compressed, the SMO classifier is still on top but now with a best accuracy of **.8583**. Compressing the classes into groups greatly increases the accuracy of the classifiers.

Full results for both the uncompressed and compressed evaluation can be found in Section B.

Part 5

Outage Detection

Chapter 20

Ground Truth

Netflix has provided us with a list of outages that occurred between March 14, 2011 and January 30, 2012. This list is not comprehensive and some of the times are questionable. Some of the outages contained in the list are internal outages that did not affect their streaming service. These outages generated no signal on Twitter. Therefore, errors of omission could fall into one of two categories: true failures to recognize outages, and uncatchable outages. Regardless, we use this as our base truth about all of the Netflix outages in that time period.

Chapter 21

Success Metrics

The accuracy of outage detection is measured using three metrics: Recall, Precision, and $F_{0.5}$.

The following definitions are used to calculate the accuracy metrics:

- tp - True Positive. Any intersection between a reported outage range and a detected outage range.
- fp - False Positive. Any detected outage that has no intersection in the events reported by Netflix.
- fn - False Negative. An alert that has no intersection on an event reported by Netflix is a false negative.

Recall

The percent of the reported events that were caught.

$$Recall = \frac{tp}{tp+fn}$$

Precision

The percent of the alerts generated that occurred during an outage event. Netflix has specified that a precision of 0.5 is an acceptable amount of noise.

$$Precision = \frac{tp}{tp+fp}$$

$\mathbf{F_{0.5}}$ Score

A harmonic mean between recall and precision. The standard F_1 score evenly weighs precision and recall. $F_{0.5}$ weighs precision more than recall. Precision is being weighed more heavily than recall because every alert that SPOONS generates would require the intervention of a Netflix engineer. Generating too many false positives would just cause SPOONS to be ignored.

$$F_\beta = (1+\beta^2) \cdot \frac{precision \cdot recall}{\beta^2 \cdot precision + recall}$$

$$F_{0.5} = 1.25 \cdot \frac{precision \cdot recall}{.25 \cdot precision + recall}$$

Chapter 22

Outage Detection Pipeline

22.1 Processors

22.2 Modeler

22.3 Monitors

TODO(eriq): Describe how the different monitors will be described. TODO(eriq): Make sure equations get numbers.

22.3.1 Monitor Parameters

TODO(eriq) Section 9.7.3 Section 9.7.2

22.3.2 Baseline Monitor

Parameter	Description	Restrictions
B	Baseline	$B > 0$

Input	Description	Restrictions
X	Any time series	-

The Baseline Monitor looks for any value above B, and counts that as anomalous. Ironically, because of its naivety it also provides a decent baseline for the Monitors.

22.3.3 Windowed Standard Deviation Monitor

Parameter	Description	Restrictions
W	Window Size	$W > 0$
L	Lower Threshold	$L > 0$
U	Upper Threshold	$U >= L$

Input	Description	Restrictions
X	Any time series	-

The Windowed Standard Deviation Monitor is one of the simplest monitors. This monitor takes a W sized window worth of data and uses the standard deviation of the window to find outliers. Any outliers more than L are considered anomalies and counts towards an alert, but are still included in the windowed standard deviation calculation. Any value above U is considered an anomaly, but not included in the standard deviation calculation. The reason for this is that values above U are extreme outliers.

The calculation for the standard deviation, is based off of an iterative approach described in Knuth's "The Art of Computer Programming" is used [14]. Because Knuth's approach was iterative, it could be modified it to calculate for a range of values in an on-line fashion.

Adding the kth value, x, to the window:

$$Mean(k) = \frac{Mean(k-1) * (k-1) - Mean(k-W) + x}{k}$$

$$V(k) = (x - Mean(k)) * (x - Mean(k-1))$$

$$T(k) = T(k-1) - V(k-W) + V(k)$$

$$WindowStdDev(k) = \sqrt{\frac{T(K))}{k-1}}$$

22.3.4 Weekly Offset Windowed Standard Deviation Monitor

Parameter	Description	Restrictions
W	Window Size	$W > 0$
L	Lower Threshold	$L > 0$
U	Upper Threshold	$U >= L$

Input	Description	Restrictions
X	Any time series	-

The Weekly Offset Window Standard Deviation Monitor leverages the periodicity of tweet volume. Not only is there a daily pattern in traffic, but there is also an even stronger weekly pattern. The stronger weekly pattern makes sense if one views Netflix-related Twitter traffic as a representative for the number of people currently watching Netflix. People tend to have pattern that they follow, and people are more available on different days of the week (especially Friday).

This monitor holds a windowed standard deviation for every 30 minute time period with 15 minute offsets for every week. Therefore, values are not compared to the other values around it, but to expected values from previous weeks. This monitor uses the same tactics as the Windows Standard Deviation Monitor for counting anomalies.

22.3.5 Mean Squared Error Monitor

Parameter	Description	Restrictions
W	Window Size	$W > 0$
T	Threshold	$T > 0$

Input	Description	Restrictions
X	The expected time series	-
Y	The actual time series	$Y(k) < X(k)$

The Mean square Error Monitor keeps a windowed mean squared error (MSE). This monitor requires two sets of input, a set of expected values and a set of actual values. Any value that causes the MSE to go above a certain threshold, T, counts towards an anomaly.

Adding the kth value to the window MSE:

$$V(k) = (X(k) - Y(k))^2$$

$$MSE(k) = \frac{MSE(k-1) * (k-1) - V(k-W) + V(k)}{k}$$

22.3.6 Ratio Monitor

Parameter	Description	Restrictions
T	Threshold	$0 < T < 1$

Input	Description	Restrictions
X	The expected time series	-
Y	The actual time series	$Y(k) < X(k)$

The Ratio Monitor takes the ratio of the actual value over the expected value for every time period. Whenever the ratio dips under the threshold T, then that period counts towards an anomaly. This monitor may seem simple, but the real challenge lies in picking a proper X and Y. If a good approximating time series can be chosen, then the monitor can be very successful.

22.3.7 Class Correlation Monitor

Parameter	Description	Restrictions
W	Window Size	$W > 0$
T	Threshold	$-1 < T < 1$

Input	Description	Restrictions
X	The expected time series	-
Y	The actual time series	$Y(k) < X(k)$

The Correlation Monitor takes the Pearson Correlation between X and Y for a running window of size W. Pearson Correlation is used because of its ability to catch the linear correlation between two time series within a normalized range.

For performance reasons SPOONS uses an approximation of Pearson Correlation which uses the windowed standard deviation approach described in Section 22.3.3.

Adding the kth value, to the window:

$$Let \bar{X} bethewindowedmeanof X.$$

$$Let \bar{Y} bethewindowedmeanof Y.$$

$$T(k) =$$

$$T(k-1) + (X(k) * Y(k)) - (X(k-W) * Y(k-W))$$

$$Pearson(k) =$$

$$\frac{T(k) - (W * \bar{X} * \bar{Y})}{(W-1) * WindowStdDev(X) * WindowStdDev(Y)}$$

Chapter 23

Results

TODO(eriq): Numbers

Part 6

Conclusions

TODO(eriq)

Chapter 24

Current Limitations of SPOONS

TODO(eriq) Severity Nature of Outage Malicious Tweet Attack Know What To Search For (dynamic search generation)

Chapter 25

Current and Future Work

25.1 WEKA Classifier Reimplementation

The WEKA machine learning package offers a wide variety of classifiers. However, their implementation and API has some room for improvement. Because of this, SPOONS already uses two classifiers implemented from scratch. I plan on continuing this to make a classification package centered around performance and ease of use.

25.2 Advanced Sentiment Analysis

Kim Paterson, a member of the SPOONS team, is currently working on improving the sentiment analysis work from Cailin Cushing[6]. If completed, then SPOONS can use both text classification and sentiment analysis to determine when there is an outage. Because of their orthogonal natures, having both would allow SPOONS to recognize even more outages.

25.3 SPOONS Scaling

Another member of the SPOONS team, Brett Armstrong, is working to improve the scalability of SPOONS. Because of its distributed architecture (see Section 9.9), SPOONS already has the potential to scale horizontally. If there is too much traffic/work, then another server can just be added to the cluster. However, that currently requires manual intervention. Since there are spikes when outages occur, we may not know when there is going to be a lot of traffic. To solve this problem, Brett will use SPOONS to monitor itself. The end result of an Analysis Pipeline will not be an email alert, rather it will be the creation of a new AWS instance.

Appendix A

SPOONS Database Schema Highlights

A.1 DATA_tweets

```
CREATE TABLE DATA_tweets (
    twitter_id varchar(32) COLLATE utf8_unicode_ci NOT NULL,
    published int(11) NOT NULL,
    content text COLLATE utf8_unicode_ci NOT NULL,
    source text COLLATE utf8_unicode_ci ,
    lang varchar(3) COLLATE utf8_unicode_ci NOT NULL,
    author varchar(50) COLLATE utf8_unicode_ci NOT NULL DEFAULT 'Jon Doe'
    frame_id int(11) DEFAULT NULL,
    id int(11) NOT NULL AUTO_INCREMENT,
    place text COLLATE utf8_unicode_ci ,
    geo text COLLATE utf8_unicode_ci ,
```

```
    PRIMARY KEY (id),
    UNIQUE KEY tweet_id (twitter_id),
    UNIQUE KEY twitter_id (twitter_id),
    KEY frame_index (frame_id),
    KEY published_index (published),
    KEY lang (lang)
);
```

Appendix B

Full Classifier Evaluation Results

Classified \ Actual	undecided	media	neutral	snafu	watching	response	complaint	refuse to rate	happy
undecided	**0**	0	0	0	0	0	0	0	0
media	0	**48**	50	0	1	2	2	0	0
neutral	0	1	**40**	3	8	8	2	0	4
snafu	0	0	48	**69**	3	6	32	0	0
watching	0	2	65	3	**46**	6	3	1	9
response	0	2	11	4	6	**1**	0	1	5
complaint	0	0	41	53	4	1	**44**	0	3
refuse to rate	0	2	24	0	2	2	0	**18**	0
happy	0	2	40	36	22	8	7	2	**30**

(a) Non-Weka NaiveBayesClassifier

Classified \ Actual	undecided	media	neutral	snafu	watching	response	complaint	refuse to rate	happy
undecided	**0**	0	0	0	0	0	0	0	0
media	0	**93**	1	1	1	3	3	1	0
neutral	0	9	**12**	5	16	13	4	0	7
snafu	0	0	2	**104**	10	5	34	1	2
watching	0	4	23	5	**83**	7	2	1	10
response	0	3	7	4	9	**0**	0	1	6
complaint	0	6	3	66	6	3	**57**	0	5
refuse to rate	0	9	2	0	2	2	1	**29**	3
happy	0	4	17	41	32	6	10	2	**35**

(b) Non-Weka BPNBClassifier

Classified \ Actual	undecided	media	neutral	snafu	watching	response	complaint	refuse to rate	happy
undecided	**0**	0	0	0	0	0	0	0	0
media	0	**85**	4	1	2	4	0	6	1
neutral	0	0	**25**	3	11	13	3	0	11
snafu	0	0	6	**94**	7	7	25	1	18
watching	0	0	26	2	**82**	9	1	0	15
response	0	2	14	3	6	**3**	0	0	2
complaint	0	1	12	42	8	10	**54**	0	19
refuse to rate	0	6	4	0	1	2	2	**30**	3
happy	0	1	22	15	14	6	7	0	**82**

(c) NaiveBayesClassifier

Classified \ Actual	undecided	media	neutral	snafu	watching	response	complaint	refuse to rate	happy
undecided	**0**	0	0	0	0	0	0	0	0
media	0	**87**	4	0	2	0	1	8	1
neutral	0	1	**1**	9	20	0	1	2	32
snafu	0	0	0	**77**	9	0	21	0	51
watching	0	0	0	6	**105**	1	1	0	22
response	0	2	0	4	9	**0**	1	0	14
complaint	0	2	0	36	10	0	**24**	0	74
refuse to rate	0	9	1	2	0	0	1	**23**	12
happy	0	1	0	26	21	0	2	0	**97**

(d) BayesNetClassifier

Table B.1: Uncompressed, NGram, NoFilter Classification Confusion Matricies

Classified \ Actual	undecided	media	neutral	snafu	watching	response	complaint	refuse to rate	happy
undecided	**0**	0	0	0	0	0	0	0	0
media	0	**95**	1	0	2	0	1	3	1
neutral	0	4	**9**	5	13	1	7	3	24
snafu	0	0	5	**84**	3	0	36	2	28
watching	0	2	6	2	**84**	5	7	0	29
response	0	2	6	5	5	**0**	1	0	11
complaint	0	2	4	37	5	1	**54**	4	39
refuse to rate	0	11	2	4	0	1	6	**14**	10
happy	0	1	5	9	17	5	9	0	**101**

(e) J48Classifier

Classified \ Actual	undecided	media	neutral	snafu	watching	response	complaint	refuse to rate	happy
undecided	**0**	0	0	0	0	0	0	0	0
media	0	**56**	0	0	1	2	0	0	44
neutral	0	0	**3**	2	7	4	5	0	45
snafu	0	0	0	**45**	0	4	59	0	50
watching	0	1	10	0	**73**	4	5	0	42
response	0	2	4	4	5	**0**	1	1	13
complaint	0	0	4	36	2	0	**61**	0	43
refuse to rate	0	2	0	0	1	1	0	**4**	40
happy	0	0	3	1	10	5	5	0	**123**

(f) KNNClassifier

Classified \ Actual	undecided	media	neutral	snafu	watching	response	complaint	refuse to rate	happy
undecided	**0**	0	0	0	0	0	0	0	0
media	0	**98**	0	0	2	2	0	0	1
neutral	0	3	**11**	2	16	7	7	0	20
snafu	0	0	1	**87**	4	4	41	0	21
watching	0	2	12	3	**88**	5	4	0	21
response	0	2	5	4	7	**0**	3	1	8
complaint	0	2	5	40	4	1	**68**	1	25
refuse to rate	0	9	1	1	1	1	0	**18**	17
happy	0	1	9	3	13	6	8	0	**107**

(g) SMOClassifier

Classified \ Actual	undecided	media	neutral	snafu	watching	response	complaint	refuse to rate	happy
undecided	**0**	0	0	0	0	0	0	0	0
media	0	**98**	1	0	2	2	0	0	0
neutral	0	1	**23**	2	12	13	4	0	11
snafu	0	0	6	**97**	7	7	25	1	15
watching	0	0	21	2	**92**	5	2	0	13
response	0	2	14	3	6	**3**	0	0	2
complaint	0	1	11	45	10	8	**53**	0	18
refuse to rate	0	13	4	1	0	3	1	**23**	3
happy	0	1	17	17	19	6	9	0	**78**

(h) BinaryNaiveBayesClassifier

Table B.1: Uncompressed, NGram, NoFilter Classification Confusion Matricies Cont.

Classified \ Actual	undecided	media	neutral	snafu	watching	response	complaint	refuse to rate	happy
undecided	**0**	0	0	0	0	0	0	0	0
media	0	**97**	0	2	0	0	1	2	1
neutral	0	2	**4**	10	10	0	3	1	36
snafu	0	0	0	**61**	6	0	26	0	65
watching	0	1	4	6	**80**	0	13	0	31
response	0	2	2	4	4	**0**	1	0	17
complaint	0	2	1	30	5	0	**41**	0	67
refuse to rate	0	14	0	2	1	0	5	**5**	21
happy	0	1	3	20	11	0	24	0	**88**

(i) BinaryJ48Classifier

Classified \ Actual	undecided	media	neutral	snafu	watching	response	complaint	refuse to rate	happy
undecided	**0**	0	0	0	0	0	0	0	0
media	0	**90**	10	0	2	0	0	1	0
neutral	0	2	**2**	9	18	0	32	0	3
snafu	0	0	1	**85**	6	0	64	2	0
watching	0	0	1	4	**107**	0	20	0	3
response	0	2	1	4	9	**0**	14	0	0
complaint	0	1	2	33	10	0	**99**	0	1
refuse to rate	0	10	2	2	0	0	12	**22**	0
happy	0	1	2	26	19	0	91	0	**8**

(j) BinaryBayesNetClassifier

Classified \ Actual	undecided	media	neutral	snafu	watching	response	complaint	refuse to rate	happy
undecided	**0**	0	0	0	0	0	0	0	0
media	0	**57**	0	0	1	2	0	0	43
neutral	0	0	**3**	1	8	4	5	0	45
snafu	0	0	0	**45**	0	4	59	0	50
watching	0	1	10	0	**74**	4	5	0	41
response	0	2	5	4	5	**0**	0	1	13
complaint	0	0	4	36	2	0	**61**	0	43
refuse to rate	0	2	0	0	1	1	0	**4**	40
happy	0	0	3	1	10	5	5	0	**123**

(k) BinaryKNNClassifier

Classified \ Actual	undecided	media	neutral	snafu	watching	response	complaint	refuse to rate	happy
undecided	**0**	0	0	0	0	0	0	0	0
media	0	**92**	7	0	1	2	1	0	0
neutral	0	1	**33**	3	10	4	3	0	12
snafu	0	0	32	**69**	2	4	43	0	8
watching	0	2	28	2	**84**	4	2	0	13
response	0	2	11	4	5	**0**	0	1	7
complaint	0	1	38	37	3	0	**55**	0	12
refuse to rate	0	9	13	1	0	1	1	**19**	4
happy	0	1	50	1	13	5	2	0	**75**

(l) BinarySMOClassifier

Table B.1: Uncompressed, NGram, NoFilter Classification Confusion Matricies Cont.

Classified \ Actual	undecided	media	neutral	snafu	watching	response	complaint	refuse to rate	happy
undecided	**0**	0	0	0	0	0	0	0	0
media	0	**59**	36	0	2	3	3	0	0
neutral	0	1	**39**	4	9	6	3	0	4
snafu	0	0	42	**69**	5	5	36	0	1
watching	0	2	64	5	**45**	5	4	0	10
response	0	2	11	4	5	**1**	1	1	5
complaint	0	1	39	51	4	1	**47**	0	3
refuse to rate	0	2	17	4	2	2	0	**21**	0
happy	0	2	39	13	21	5	7	1	**59**

(a) Non-Weka NaiveBayesClassifier

Classified \ Actual	undecided	media	neutral	snafu	watching	response	complaint	refuse to rate	happy
undecided	**0**	0	0	0	0	0	0	0	0
media	0	**88**	2	4	2	3	3	1	0
neutral	0	9	**9**	5	19	7	8	0	9
snafu	0	1	2	**100**	9	5	39	0	2
watching	0	3	19	8	**82**	5	2	1	15
response	0	3	5	4	9	**1**	1	1	6
complaint	0	7	4	66	5	2	**57**	0	5
refuse to rate	0	7	1	4	5	2	1	**26**	2
happy	0	4	11	18	27	5	10	1	**71**

(b) Non-Weka BPNBClassifier

Classified \ Actual	undecided	media	neutral	snafu	watching	response	complaint	refuse to rate	happy
undecided	**0**	0	0	0	0	0	0	0	0
media	0	**81**	3	0	1	3	0	14	1
neutral	0	0	**23**	3	14	11	11	1	3
snafu	0	0	7	**81**	8	4	55	2	1
watching	0	0	26	0	**82**	5	12	0	10
response	0	2	12	3	6	**3**	1	0	3
complaint	0	1	9	36	9	8	**77**	3	3
refuse to rate	0	5	4	1	2	4	9	**21**	2
happy	0	0	18	5	22	5	31	1	**65**

(c) NaiveBayesClassifier

Classified \ Actual	undecided	media	neutral	snafu	watching	response	complaint	refuse to rate	happy
undecided	**0**	0	0	0	0	0	0	0	0
media	0	**94**	1	0	0	0	1	5	2
neutral	0	3	**0**	8	3	1	49	0	2
snafu	0	0	0	**48**	1	2	103	0	4
watching	0	0	3	5	**79**	2	39	0	7
response	0	2	0	8	4	**2**	9	1	4
complaint	0	2	0	20	2	4	**112**	0	6
refuse to rate	0	14	0	5	0	0	20	**8**	1
happy	0	1	0	11	10	2	79	0	**44**

(d) BayesNetClassifier

Table B.2: Uncompressed, NGram, EriqFilter Classification Confusion Matricies

Classified \ Actual	undecided	media	neutral	snafu	watching	response	complaint	refuse to rate	happy
undecided	**0**	0	0	0	0	0	0	0	0
media	0	**99**	0	0	0	0	2	1	1
neutral	0	4	**10**	2	8	3	13	1	25
snafu	0	0	5	**71**	4	3	50	0	25
watching	0	1	7	2	**90**	2	11	1	21
response	0	2	5	2	8	**0**	2	0	11
complaint	0	2	7	35	4	1	**72**	0	25
refuse to rate	0	13	2	1	3	1	6	**10**	12
happy	0	1	11	4	14	3	15	0	**99**

(e) J48Classifier

Classified \ Actual	undecided	media	neutral	snafu	watching	response	complaint	refuse to rate	happy
undecided	**0**	0	0	0	0	0	0	0	0
media	0	**58**	17	9	1	2	7	0	9
neutral	0	0	**21**	13	8	4	10	0	10
snafu	0	0	20	**57**	0	4	72	0	5
watching	0	1	39	12	**40**	4	11	0	28
response	0	2	5	5	4	**0**	2	1	11
complaint	0	0	18	53	2	0	**66**	0	7
refuse to rate	0	0	13	9	2	1	7	**4**	12
happy	0	0	36	16	6	5	9	0	**75**

(f) KNNClassifier

Classified \ Actual	undecided	media	neutral	snafu	watching	response	complaint	refuse to rate	happy
undecided	**0**	0	0	0	0	0	0	0	0
media	0	**98**	1	0	1	2	0	1	0
neutral	0	4	**7**	3	15	5	6	0	26
snafu	0	0	3	**86**	2	4	50	0	13
watching	0	2	13	2	**92**	5	8	0	13
response	0	2	6	4	5	**1**	1	1	10
complaint	0	2	4	48	5	3	**74**	0	10
refuse to rate	0	11	0	1	2	2	6	**8**	18
happy	0	1	7	6	16	5	10	0	**102**

(g) SMOClassifier

Classified \ Actual	undecided	media	neutral	snafu	watching	response	complaint	refuse to rate	happy
undecided	**0**	0	0	0	0	0	0	0	0
media	0	**95**	2	0	1	2	0	2	1
neutral	0	2	**23**	4	14	9	11	0	3
snafu	0	0	7	**85**	8	4	52	1	1
watching	0	0	21	0	**87**	5	12	0	10
response	0	2	12	3	6	**3**	1	0	3
complaint	0	1	10	42	9	7	**73**	2	2
refuse to rate	0	9	4	1	2	4	9	**17**	2
happy	0	0	16	6	21	5	30	1	**68**

(h) BinaryNaiveBayesClassifier

Table B.2: Uncompressed, NGram, EriqFilter Classification Confusion Matricies Cont.

Classified \ Actual	undecided	media	neutral	snafu	watching	response	complaint	refuse to rate	happy
undecided	**0**	0	0	0	0	0	0	0	0
media	0	**99**	0	2	0	0	0	0	2
neutral	0	4	**0**	25	13	0	22	0	2
snafu	0	0	0	**97**	6	0	55	0	0
watching	0	1	0	28	**78**	0	23	0	5
response	0	2	1	2	5	**0**	18	1	1
complaint	0	2	0	80	2	0	**59**	0	3
refuse to rate	0	14	0	14	3	0	12	**4**	1
happy	0	0	1	58	10	0	30	0	**48**

(i) BinaryJ48Classifier

Classified \ Actual	undecided	media	neutral	snafu	watching	response	complaint	refuse to rate	happy
undecided	**0**	0	0	0	0	0	0	0	0
media	0	**95**	0	0	0	4	1	1	2
neutral	0	5	**1**	19	3	0	34	0	4
snafu	0	0	1	**73**	1	0	79	0	4
watching	0	0	1	10	**75**	0	32	0	17
response	0	2	1	6	4	**0**	11	0	6
complaint	0	2	1	27	2	0	**107**	0	7
refuse to rate	0	13	0	6	0	2	17	**9**	1
happy	0	0	2	18	9	0	63	0	**55**

(j) BinaryBayesNetClassifier

Classified \ Actual	undecided	media	neutral	snafu	watching	response	complaint	refuse to rate	happy
undecided	**0**	0	0	0	0	0	0	0	0
media	0	**57**	19	11	1	2	7	0	6
neutral	0	0	**23**	14	8	4	10	0	7
snafu	0	0	22	**57**	0	4	72	0	3
watching	0	1	41	12	**42**	4	11	0	24
response	0	2	7	7	4	**0**	1	1	8
complaint	0	0	18	57	3	0	**66**	0	2
refuse to rate	0	0	16	13	2	1	7	**4**	5
happy	0	0	40	17	6	5	9	0	**70**

(k) BinaryKNNClassifier

Classified \ Actual	undecided	media	neutral	snafu	watching	response	complaint	refuse to rate	happy
undecided	**0**	0	0	0	0	0	0	0	0
media	0	**96**	4	0	1	2	0	0	0
neutral	0	4	**37**	2	8	4	3	0	8
snafu	0	0	44	**70**	2	4	37	0	1
watching	0	2	32	1	**84**	5	4	0	7
response	0	2	11	4	6	**0**	0	1	6
complaint	0	2	43	38	4	0	**57**	0	2
refuse to rate	0	10	24	0	1	1	1	**9**	2
happy	0	1	46	1	13	5	3	0	**78**

(l) BinarySMOClassifier

Table B.2: Uncompressed, NGram, EriqFilter Classification Confusion Matricies Cont.

Classified \ Actual	undecided	media	neutral	snafu	watching	response	complaint	refuse to rate	happy
undecided	**0**	0	0	0	0	0	0	0	0
media	0	**82**	10	0	5	4	0	1	1
neutral	0	2	**17**	5	18	18	2	1	3
snafu	0	6	10	**99**	13	7	20	1	2
watching	0	5	26	4	**79**	14	2	0	5
response	0	4	10	2	6	**3**	2	1	2
complaint	0	4	16	62	14	6	**42**	2	0
refuse to rate	0	18	6	2	1	5	0	**16**	0
happy	0	13	22	56	24	15	3	0	**14**

(a) Non-Weka NaiveBayesClassifier

Classified \ Actual	undecided	media	neutral	snafu	watching	response	complaint	refuse to rate	happy
undecided	**0**	0	0	0	0	0	0	0	0
media	0	**101**	0	0	2	0	0	0	0
neutral	0	15	**10**	7	13	21	0	0	0
snafu	0	15	4	**98**	13	8	18	2	0
watching	0	10	13	6	**92**	13	1	0	0
response	0	11	7	4	4	**4**	0	0	0
complaint	0	17	8	59	11	13	**38**	0	0
refuse to rate	0	30	1	2	1	2	0	**12**	0
happy	0	18	12	56	36	7	2	0	**16**

(b) Non-Weka BPNBClassifier

Classified \ Actual	undecided	media	neutral	snafu	watching	response	complaint	refuse to rate	happy
undecided	**0**	0	0	0	0	0	0	0	0
media	0	**66**	3	1	2	6	0	24	1
neutral	1	0	**20**	2	11	20	3	0	9
snafu	0	0	7	**96**	4	10	24	0	17
watching	0	0	23	6	**72**	12	3	0	19
response	0	2	15	2	3	**5**	0	0	3
complaint	0	1	7	39	10	18	**49**	0	22
refuse to rate	0	4	6	4	0	3	1	**29**	1
happy	6	0	16	17	15	10	6	0	**77**

(c) NaiveBayesClassifier

Classified \ Actual	undecided	media	neutral	snafu	watching	response	complaint	refuse to rate	happy
undecided	**0**	0	0	0	0	0	0	0	0
media	3	**45**	7	1	1	1	0	45	0
neutral	0	0	**10**	9	14	16	0	8	9
snafu	0	0	4	**86**	8	11	11	3	35
watching	0	0	10	8	**81**	16	0	0	20
response	0	2	6	4	5	**6**	0	5	2
complaint	0	0	11	42	7	14	**16**	4	52
refuse to rate	0	7	11	6	0	1	0	**22**	1
happy	0	0	7	37	20	7	2	7	**67**

(d) BayesNetClassifier

Table B.3: Uncompressed, TweetFSG, NoFilter Classification Confusion Matricies

Actual / Classified	undecided	media	neutral	snafu	watching	response	complaint	refuse to rate	happy
undecided	**0**	0	0	0	0	0	0	0	0
media	0	**97**	0	0	1	0	1	2	2
neutral	0	3	**12**	7	14	4	9	0	17
snafu	0	0	5	**89**	2	1	41	2	18
watching	0	1	10	7	**90**	1	6	1	19
response	0	2	6	6	5	**0**	4	1	6
complaint	0	2	3	42	9	2	**61**	3	24
refuse to rate	0	12	1	6	1	1	8	**16**	3
happy	0	1	8	11	22	2	20	1	**82**

(e) J48Classifier

Actual / Classified	undecided	media	neutral	snafu	watching	response	complaint	refuse to rate	happy
undecided	**0**	0	0	0	0	0	0	0	0
media	0	**81**	0	0	3	2	1	13	3
neutral	0	7	**4**	4	11	4	7	15	14
snafu	0	2	2	**45**	4	4	53	19	29
watching	0	6	10	3	**80**	4	7	9	16
response	0	3	4	5	5	**0**	2	4	7
complaint	0	6	3	37	9	1	**50**	15	25
refuse to rate	0	7	0	0	1	2	0	**33**	5
happy	0	3	5	5	15	7	10	13	**89**

(f) KNNClassifier

Actual / Classified	undecided	media	neutral	snafu	watching	response	complaint	refuse to rate	happy
undecided	**0**	0	0	0	0	0	0	0	0
media	0	**94**	0	0	2	2	1	4	0
neutral	0	1	**14**	2	11	6	9	2	21
snafu	0	0	3	**92**	3	4	40	0	16
watching	0	2	16	1	**84**	7	4	0	21
response	0	2	9	4	5	**0**	0	1	9
complaint	0	1	7	44	7	1	**64**	2	20
refuse to rate	0	9	2	0	1	2	1	**28**	5
happy	0	1	10	2	17	6	7	1	**103**

(g) SMOClassifier

Actual / Classified	undecided	media	neutral	snafu	watching	response	complaint	refuse to rate	happy
undecided	**0**	0	0	0	0	0	0	0	0
media	0	**84**	3	2	2	1	0	10	1
neutral	1	0	**20**	2	14	18	3	0	8
snafu	0	0	7	**99**	5	8	23	0	16
watching	0	0	19	7	**85**	9	2	0	13
response	0	2	15	2	4	**4**	0	0	3
complaint	0	1	7	41	11	17	**46**	0	23
refuse to rate	0	10	5	5	0	3	1	**23**	1
happy	6	0	14	19	22	9	6	0	**71**

(h) BinaryNaiveBayesClassifier

Table B.3: Uncompressed, TweetFSG, NoFilter Classification Confusion Matricies Cont.

Classified \ Actual	undecided	media	neutral	snafu	watching	response	complaint	refuse to rate	happy
undecided	**0**	0	0	0	0	0	0	0	0
media	0	**95**	2	1	1	0	1	0	3
neutral	0	2	**5**	10	18	0	13	1	17
snafu	0	0	3	**99**	4	0	36	4	12
watching	0	1	7	14	**96**	0	3	0	14
response	0	2	2	6	9	**0**	9	0	2
complaint	0	2	2	50	6	0	**52**	4	30
refuse to rate	0	14	0	11	0	0	8	**11**	4
happy	0	1	5	33	25	0	13	1	**69**

(i) BinaryJ48Classifier

Classified \ Actual	undecided	media	neutral	snafu	watching	response	complaint	refuse to rate	happy
undecided	**0**	0	0	0	0	0	0	0	0
media	0	**78**	8	0	1	1	0	15	0
neutral	0	2	**6**	12	19	2	0	18	7
snafu	0	0	1	**91**	2	2	7	15	40
watching	0	0	2	10	**101**	1	1	6	14
response	0	3	3	6	8	**1**	0	8	1
complaint	0	2	6	41	7	3	**20**	19	48
refuse to rate	0	9	3	5	0	3	0	**27**	1
happy	0	0	7	38	24	0	0	11	**67**

(j) BinaryBayesNetClassifier

Classified \ Actual	undecided	media	neutral	snafu	watching	response	complaint	refuse to rate	happy
undecided	**0**	0	0	0	0	0	0	0	0
media	0	**81**	0	2	3	2	1	11	3
neutral	0	8	**4**	5	14	5	6	14	10
snafu	0	3	3	**52**	13	5	51	17	14
watching	0	7	12	2	**87**	5	6	6	10
response	0	3	5	5	5	**0**	1	4	7
complaint	0	7	4	40	10	4	**49**	11	21
refuse to rate	0	7	0	1	3	3	0	**31**	3
happy	0	3	6	9	23	7	9	9	**81**

(k) BinaryKNNClassifier

Classified \ Actual	undecided	media	neutral	snafu	watching	response	complaint	refuse to rate	happy
undecided	**0**	0	0	0	0	0	0	0	0
media	0	**89**	9	0	1	2	1	1	0
neutral	0	1	**31**	1	8	5	8	0	12
snafu	0	0	38	**64**	1	4	44	0	7
watching	0	2	33	1	**80**	5	4	0	10
response	0	2	10	4	5	**0**	1	1	7
complaint	0	1	35	38	4	0	**57**	0	11
refuse to rate	0	10	8	0	1	1	1	**25**	2
happy	0	1	45	0	17	4	1	0	**79**

(l) BinarySMOClassifier

Table B.3: Uncompressed, TweetFSG, NoFilter Classification Confusion Matricies Cont.

Classified \ Actual	undecided	media	neutral	snafu	watching	response	complaint	refuse to rate	happy
undecided	**0**	0	0	0	0	0	0	0	0
media	0	**93**	5	0	0	4	0	1	0
neutral	0	8	**16**	6	14	18	1	3	0
snafu	0	8	6	**100**	17	7	15	5	0
watching	0	10	16	5	**74**	23	3	1	3
response	0	6	8	1	3	**8**	0	3	1
complaint	0	12	9	57	9	12	**43**	3	1
refuse to rate	0	30	4	0	1	9	0	**3**	1
happy	0	17	9	32	28	19	1	1	**40**

(a) Non-Weka NaiveBayesClassifier

Classified \ Actual	undecided	media	neutral	snafu	watching	response	complaint	refuse to rate	happy
undecided	**0**	0	0	0	0	0	0	0	0
media	0	**102**	0	0	1	0	0	0	0
neutral	0	25	**8**	7	9	15	2	0	0
snafu	0	21	1	**99**	10	7	20	0	0
watching	0	29	9	9	**78**	9	0	0	1
response	0	16	2	1	4	**7**	0	0	0
complaint	0	29	2	59	6	10	**40**	0	0
refuse to rate	0	40	1	1	2	3	0	**0**	1
happy	0	23	6	38	26	9	1	0	**44**

(b) Non-Weka BPNBClassifier

Classified \ Actual	undecided	media	neutral	snafu	watching	response	complaint	refuse to rate	happy
undecided	**0**	0	0	0	0	0	0	0	0
media	0	**66**	5	3	1	3	0	24	1
neutral	1	1	**24**	9	9	10	4	3	5
snafu	0	0	6	**91**	7	10	27	1	16
watching	0	0	21	11	**63**	8	2	2	28
response	0	2	14	3	2	**5**	0	2	2
complaint	0	1	10	44	10	12	**50**	7	12
refuse to rate	0	4	5	5	5	5	0	**23**	1
happy	6	0	15	19	19	6	5	1	**76**

(c) NaiveBayesClassifier

Classified \ Actual	undecided	media	neutral	snafu	watching	response	complaint	refuse to rate	happy
undecided	**0**	0	0	0	0	0	0	0	0
media	2	**29**	1	0	3	1	3	64	0
neutral	0	0	**1**	10	14	9	2	21	9
snafu	0	0	3	**65**	10	13	22	7	38
watching	0	0	7	13	**55**	14	3	10	33
response	0	0	1	3	4	**11**	0	9	2
complaint	0	0	3	32	11	13	**20**	15	52
refuse to rate	0	5	4	5	5	8	0	**20**	1
happy	0	0	1	19	25	8	11	10	**73**

(d) BayesNetClassifier

Table B.4: Uncompressed, TweetFSG, EriqFilter Classification Confusion Matricies

Classified \ Actual	undecided	media	neutral	snafu	watching	response	complaint	refuse to rate	happy
undecided	**0**	0	0	0	0	0	0	0	0
media	0	**99**	0	1	0	0	1	1	1
neutral	0	4	**11**	7	11	5	8	4	16
snafu	0	0	1	**77**	6	1	48	3	22
watching	0	1	10	7	**87**	0	9	2	19
response	0	2	6	3	6	**1**	2	0	10
complaint	0	2	8	42	7	2	**59**	2	24
refuse to rate	0	13	4	7	1	0	7	**13**	3
happy	0	1	7	16	15	3	8	4	**93**

(e) J48Classifier

Classified \ Actual	undecided	media	neutral	snafu	watching	response	complaint	refuse to rate	happy
undecided	**0**	0	0	0	0	0	0	0	0
media	0	**83**	0	1	3	2	0	14	0
neutral	0	8	**3**	2	9	4	3	24	13
snafu	0	1	0	**43**	10	3	44	36	21
watching	0	2	9	7	**59**	5	5	29	19
response	0	3	4	3	5	**0**	2	6	7
complaint	0	3	2	35	5	0	**45**	29	27
refuse to rate	0	13	0	2	1	1	1	**27**	3
happy	0	3	4	10	15	5	3	30	**77**

(f) KNNClassifier

Classified \ Actual	undecided	media	neutral	snafu	watching	response	complaint	refuse to rate	happy
undecided	**0**	0	0	0	0	0	0	0	0
media	0	**95**	1	0	1	2	1	3	0
neutral	0	3	**13**	3	9	6	7	4	21
snafu	0	0	4	**80**	2	4	47	6	15
watching	0	2	13	7	**85**	6	3	4	15
response	0	2	5	5	5	**0**	3	4	6
complaint	0	2	4	43	7	2	**65**	9	14
refuse to rate	0	11	2	3	1	2	1	**23**	5
happy	0	1	10	3	14	7	9	2	**101**

(g) SMOClassifier

Classified \ Actual	undecided	media	neutral	snafu	watching	response	complaint	refuse to rate	happy
undecided	**0**	0	0	0	0	0	0	0	0
media	0	**78**	3	3	2	3	0	13	1
neutral	1	2	**23**	9	10	10	4	2	5
snafu	0	0	6	**97**	7	9	22	1	16
watching	0	0	18	15	**75**	4	2	2	19
response	0	2	14	4	2	**5**	0	2	1
complaint	0	1	10	44	10	12	**50**	7	12
refuse to rate	0	9	4	5	5	5	0	**19**	1
happy	1	0	15	20	20	5	3	1	**82**

(h) BinaryNaiveBayesClassifier

Table B.4: Uncompressed, TweetFSG, EriqFilter Classification Confusion Matricies Cont.

Actual Classified	undecided	media	neutral	snafu	watching	response	complaint	refuse to rate	happy
undecided	**0**	0	0	0	0	0	0	0	0
media	0	**99**	0	0	0	0	1	0	3
neutral	0	4	**0**	15	11	0	19	0	17
snafu	0	0	0	**102**	7	0	32	0	17
watching	0	1	0	19	**82**	0	16	0	17
response	0	2	1	3	6	**0**	9	0	9
complaint	0	2	1	63	5	0	**54**	0	21
refuse to rate	0	13	0	4	1	0	12	**9**	9
happy	0	0	1	44	11	0	27	0	**64**

(i) BinaryJ48Classifier

Actual Classified	undecided	media	neutral	snafu	watching	response	complaint	refuse to rate	happy
undecided	**0**	0	0	0	0	0	0	0	0
media	0	**74**	3	0	0	0	0	26	0
neutral	0	5	**0**	10	4	2	1	33	11
snafu	0	0	0	**70**	1	2	8	22	55
watching	0	0	1	13	**66**	2	0	22	31
response	0	3	0	4	4	**4**	0	13	2
complaint	0	3	0	22	2	3	**20**	33	63
refuse to rate	0	11	0	4	0	2	0	**29**	2
happy	0	0	0	19	8	2	3	26	**89**

(j) BinaryBayesNetClassifier

Actual Classified	undecided	media	neutral	snafu	watching	response	complaint	refuse to rate	happy
undecided	**0**	0	0	0	0	0	0	0	0
media	0	**83**	0	1	4	2	0	13	0
neutral	0	8	**3**	3	13	4	3	21	11
snafu	0	1	0	**52**	16	4	44	24	17
watching	0	4	9	6	**68**	5	4	24	15
response	0	3	5	3	5	**0**	1	6	7
complaint	0	5	2	36	12	0	**44**	25	22
refuse to rate	0	14	0	2	1	1	0	**27**	3
happy	0	3	4	14	22	5	3	29	**67**

(k) BinaryKNNClassifier

Actual Classified	undecided	media	neutral	snafu	watching	response	complaint	refuse to rate	happy
undecided	**0**	0	0	0	0	0	0	0	0
media	0	**89**	9	0	1	2	1	1	0
neutral	0	3	**39**	1	8	4	5	0	6
snafu	0	0	39	**68**	1	3	45	0	2
watching	0	2	33	2	**84**	3	3	0	8
response	0	2	12	3	5	**0**	1	2	5
complaint	0	2	47	31	3	0	**58**	4	1
refuse to rate	0	10	17	0	1	1	1	**16**	2
happy	0	1	62	1	9	5	4	0	**65**

(l) BinarySMOClassifier

Table B.4: Uncompressed, TweetFSG, EriqFilter Classification Confusion Matricies Cont.

Classified \ Actual	media	snafu	other
media	**48**	2	53
snafu	0	**198**	106
other	9	58	**359**

(a) Non-Weka NaiveBayesClassifier

Classified \ Actual	media	snafu	other
media	**93**	4	6
snafu	6	**261**	37
other	29	72	**325**

(b) Non-Weka BPNBClassifier

Classified \ Actual	media	snafu	other
media	**85**	1	17
snafu	1	**215**	88
other	9	36	**381**

(c) NaiveBayesClassifier

Classified \ Actual	media	snafu	other
media	**87**	1	15
snafu	2	**158**	144
other	13	53	**360**

(d) BayesNetClassifier

Classified \ Actual	media	snafu	other
media	**95**	1	7
snafu	2	**211**	91
other	20	55	**351**

(e) J48Classifier

Classified \ Actual	media	snafu	other
media	**56**	0	47
snafu	0	**201**	103
other	5	23	**398**

(f) KNNClassifier

Classified \ Actual	media	snafu	other
media	**98**	0	5
snafu	2	**236**	66
other	17	35	**374**

(g) SMOClassifier

Classified \ Actual	media	snafu	other
media	**98**	0	5
snafu	1	**220**	83
other	17	41	**368**

(h) BinaryNaiveBayesClassifier

Classified \ Actual	media	snafu	other
media	**97**	3	3
snafu	2	**158**	144
other	20	88	**318**

(i) BinaryJ48Classifier

Classified \ Actual	media	snafu	other
media	**90**	0	13
snafu	1	**281**	22
other	15	214	**197**

(j) BinaryBayesNetClassifier

Classified \ Actual	media	snafu	other
media	**57**	0	46
snafu	0	**201**	103
other	5	21	**400**

(k) BinaryKNNClassifier

Classified \ Actual	media	snafu	other
media	**92**	1	10
snafu	1	**204**	99
other	15	19	**392**

(l) BinarySMOClassifier

Table B.5: Compressed, NGram, NoFilter Classification Confusion Matricies

Classified \ Actual	media	snafu	other
media	**59**	3	41
snafu	1	**203**	100
other	9	45	**372**

(a) Non-Weka NaiveBayesClassifier

Classified \ Actual	media	snafu	other
media	**88**	7	8
snafu	8	**262**	34
other	26	61	**339**

(b) Non-Weka BPNBClassifier

Classified \ Actual	media	snafu	other
media	**81**	0	22
snafu	1	**249**	54
other	7	76	**343**

(c) NaiveBayesClassifier

Classified \ Actual	media	snafu	other
media	**94**	1	8
snafu	2	**283**	19
other	20	233	**173**

(d) BayesNetClassifier

Classified \ Actual	media	snafu	other
media	**99**	2	2
snafu	2	**228**	74
other	21	58	**347**

(e) J48Classifier

Classified \ Actual	media	snafu	other
media	**58**	16	29
snafu	0	**248**	56
other	3	94	**329**

(f) KNNClassifier

Classified \ Actual	media	snafu	other
media	**98**	0	5
snafu	2	**258**	44
other	20	47	**359**

(g) SMOClassifier

Classified \ Actual	media	snafu	other
media	**95**	0	8
snafu	1	**252**	51
other	13	77	**336**

(h) BinaryNaiveBayesClassifier

Classified \ Actual	media	snafu	other
media	**99**	2	2
snafu	2	**291**	11
other	21	232	**173**

(i) BinaryJ48Classifier

Classified \ Actual	media	snafu	other
media	**95**	1	7
snafu	2	**286**	16
other	20	216	**190**

(j) BinaryBayesNetClassifier

Classified \ Actual	media	snafu	other
media	**57**	18	28
snafu	0	**252**	52
other	3	101	**322**

(k) BinaryKNNClassifier

Classified \ Actual	media	snafu	other
media	**96**	0	7
snafu	2	**202**	100
other	19	19	**388**

(l) BinarySMOClassifier

Table B.6: Compressed, NGram, EriqFilter Classification Confusion Matricies

Classified \ Actual	media	snafu	other
media	**82**	0	21
snafu	10	**223**	71
other	42	78	**306**

(a) Non-Weka NaiveBayesClassifier

Classified \ Actual	media	snafu	other
media	**101**	0	2
snafu	32	**213**	59
other	84	78	**264**

(b) Non-Weka BPNBClassifier

Classified \ Actual	media	snafu	other
media	**66**	1	36
snafu	1	**208**	95
other	6	44	**376**

(c) NaiveBayesClassifier

Classified \ Actual	media	snafu	other
media	**45**	1	57
snafu	0	**155**	149
other	9	66	**351**

(d) BayesNetClassifier

Classified \ Actual	media	snafu	other
media	**97**	1	5
snafu	2	**233**	69
other	19	84	**323**

(e) J48Classifier

Classified \ Actual	media	snafu	other
media	**81**	1	21
snafu	8	**185**	111
other	26	43	**357**

(f) KNNClassifier

Classified \ Actual	media	snafu	other
media	**94**	1	8
snafu	1	**240**	63
other	15	30	**381**

(g) SMOClassifier

Classified \ Actual	media	snafu	other
media	**84**	2	17
snafu	1	**209**	94
other	12	47	**367**

(h) BinaryNaiveBayesClassifier

Classified \ Actual	media	snafu	other
media	**95**	2	6
snafu	2	**237**	65
other	20	120	**286**

(i) BinaryJ48Classifier

Classified \ Actual	media	snafu	other
media	**78**	0	25
snafu	2	**159**	143
other	14	72	**340**

(j) BinaryBayesNetClassifier

Classified \ Actual	media	snafu	other
media	**81**	3	19
snafu	10	**192**	102
other	28	44	**354**

(k) BinaryKNNClassifier

Classified \ Actual	media	snafu	other
media	**89**	1	13
snafu	1	**203**	100
other	16	21	**389**

(l) BinarySMOClassifier

Table B.7: Compressed, TweetFSG, NoFilter Classification Confusion Matricies

Classified \ Actual	media	snafu	other
media	**93**	0	10
snafu	20	**215**	69
other	71	49	**306**

(a) Non-Weka NaiveBayesClassifier

Classified \ Actual	media	snafu	other
media	**102**	0	1
snafu	50	**218**	36
other	133	59	**234**

(b) Non-Weka BPNBClassifier

Classified \ Actual	media	snafu	other
media	**66**	3	34
snafu	1	**212**	91
other	7	58	**361**

(c) NaiveBayesClassifier

Classified \ Actual	media	snafu	other
media	**29**	3	71
snafu	0	**139**	165
other	5	66	**355**

(d) BayesNetClassifier

Classified \ Actual	media	snafu	other
media	**99**	2	2
snafu	2	**226**	76
other	21	74	**331**

(e) J48Classifier

Classified \ Actual	media	snafu	other
media	**83**	1	19
snafu	4	**167**	133
other	29	38	**359**

(f) KNNClassifier

Classified \ Actual	media	snafu	other
media	**95**	1	7
snafu	2	**235**	67
other	19	44	**363**

(g) SMOClassifier

Classified \ Actual	media	snafu	other
media	**78**	3	22
snafu	1	**213**	90
other	13	62	**351**

(h) BinaryNaiveBayesClassifier

Classified \ Actual	media	snafu	other
media	**99**	1	3
snafu	2	**251**	51
other	20	168	**238**

(i) BinaryJ48Classifier

Classified \ Actual	media	snafu	other
media	**74**	0	29
snafu	3	**120**	181
other	19	54	**353**

(j) BinaryBayesNetClassifier

Classified \ Actual	media	snafu	other
media	**83**	1	19
snafu	6	**176**	122
other	32	39	**355**

(k) BinaryKNNClassifier

Classified \ Actual	media	snafu	other
media	**89**	1	13
snafu	2	**202**	100
other	18	21	**387**

(l) BinarySMOClassifier

Table B.8: Compressed, TweetFSG, EriqFilter Classification Confusion Matricies

Glossary

AWS Amazon Web Services. Cloud computing offerd by Amazon.

EC2 Elastic Compute Cloud. Instance based cloud computing machines offered through AWS.

Netflix Inc. [NASDAQ: NFLX] is the world's leading Internet subscription service for enjoying movies and TV series with more than 23 million streaming members in the United States, Canada, Latin America, the United Kingdom and Ireland[**?**].

Real Time Some of Netflix's services stream to customers in real time which means the users expect to get immediate responses from those services. So when they go down, the customers want the problem to be fixed immediately. These analysis methods need to have real time responses that are as close to immediate detection as possible. This means that the system needs to use whatever information it has available to it up to right before the outage to detect the event and alert Netflix engineers.

SPOONS Swift Perception Of Online Negative Situations. The name of the system presented in this paper.

Time Series Analysis The analysis of a series of data points over time. In this work those data points are the volume or estimated sentiment of a subset of the traffic about Netflix on Twitter during a time period.

Tweet A micro-post to a Twitter service. Tweets are limited to 140 characters.

Twitter Twitter is a social media service that allows users to post tweets (micro-posts) about any topic.

Bibliography

[1] Innodb table and index structures.

[2] E. Augustine, C. Cushing, A. Dekhtyar, M. Tognetti, and K. Paterson. Outage detection via real-time social stream analysis: Leveraging the power of online complaints. In *WWW 2012: Proceedings of the 21st World Wide Web Conference*. ACM, 2012.

[3] L. Chu. Research on chinese text categorization method oriented to imbalanced corpus. may 2012.

[4] C. Cortes and V. Vapnik. Support-vector networks. *Mach. Learn.*, 20(3):273–297, Sept. 1995.

[5] A. Culotta. Detecting influenza outbreaks by analyzing twitter messages. In *KDD Workshop on Social Media Analytics*, 2010.

[6] C. Cushing. Detecting netflix service outages through analysis of twitter posts. Master's thesis, California Polytechnic State University - San Luis Obispo, june 2012.

[7] R. Duda, P. Hart, and D. Stork. *Pattern classification*. Pattern Classification and Scene Analysis: Pattern Classification. Wiley, 2001.

[8] E. Frank and R. R. Bouckaert. Naive bayes for text classification with unbalanced classes. In *Proceedings of the 10th European conference on Principle and Practice of Knowledge Discovery in Databases*, PKDD'06, pages 503–510, Berlin, Heidelberg, 2006. Springer-Verlag.

[9] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: an update. *SIGKDD Explor. Newsl.*, 11(1):10–18, 2009.

[10] L. Hong and B. D. Davison. Empirical study of topic modeling in twitter. In *Proceedings of the First Workshop on Social Media Analytics*, SOMA '10, pages 80–88, New York, NY, USA, 2010. ACM.

[11] F. Jabr. Using twitter to follow trends beats the stock market. *NewScientist*, (2829), Sept. 2011. http://www.newscientist.com/article/mg21128295.900-using-twitter-to-follow-trends-beats-the-stock-market.html.

[12] L. Jiang, M. Yu, M. Zhou, X. Liu, and T. Zhao. Target-dependent twitter sentiment classification. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, HLT '11, pages 151–160, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.

[13] A. M. Kibriya, E. Frank, B. Pfahringer, and G. Holmes. Multinomial naive bayes for text categorization revisited. In *Proceedings of the 17th Australian joint conference on Advances in Artificial Intelligence*, AI'04, pages 488–499, Berlin, Heidelberg, 2004. Springer-Verlag.

[14] D. E. Knuth. *The art of computer programming, volume 2 (3rd ed.): seminu-*

merical algorithms. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

[15] Y. Matsu, M. Okazak, and T. Sakak. Earthquake shakes twitter users: real-time event detection by social sensors. In *WWW 2010: Proceedings of the 19th World Wide Web Conference*, 2010. http://ymatsuo.com/papers/www2010.pdf.

[16] K. McEntee. personal communication, 2011.

[17] S. Mukherjee, A. Malu, B. A.R., and P. Bhattacharyya. Twisent: a multi-stage system for analyzing sentiment in twitter. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, CIKM '12, pages 2531–2534, New York, NY, USA, 2012. ACM.

[18] R. E. Neapolitan. *Probabilistic reasoning in expert systems: theory and algorithms*. John Wiley & Sons, Inc., New York, NY, USA, 1990.

[19] J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

[20] M. Pennacchiotti and A.-M. Popescu. Democrats, republicans and starbucks afficionados: user classification in twitter. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '11, pages 430–438, New York, NY, USA, 2011. ACM.

[21] J. C. Platt. Advances in kernel methods. chapter Fast training of support vector machines using sequential minimal optimization, pages 185–208. MIT Press, Cambridge, MA, USA, 1999.

[22] J. C. Platt. Advances in kernel methods. chapter Fast training of support vector machines using sequential minimal optimization, pages 185–208. MIT Press, Cambridge, MA, USA, 1999.

[23] M. F. Porter. An algorithm for suffix stripping. In K. Sparck Jones and P. Willett, editors, *Readings in information retrieval*, pages 313–316. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997. http://tartarus.org/martin/PorterStemmer.

[24] J. R. Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[25] J. R. Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[26] Y. Raz. Automatic humor classification on twitter. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Student Research Workshop*, NAACL HLT '12, pages 66–70, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics.

[27] H. Saif, Y. He, and H. Alani. Semantic sentiment analysis of twitter. In *Proceedings of the 11th international conference on The Semantic Web - Volume Part I*, ISWC'12, pages 508–524, Berlin, Heidelberg, 2012. Springer-Verlag.

[28] Twitter. #numbers, Mar. 2011. http://blog.twitter.com/2011/03/numbers.html.

[29] Twitter. Terms of service, June 2011. https://twitter.com/tos.

[30] Twitter. Get search/tweets, Oct. 2012. https://dev.twitter.com/docs/api/1.1/get/search/tweets.

[31] X. Wang, F. Wei, X. Liu, M. Zhou, and M. Zhang. Topic sentiment analysis in twitter: a graph-based hashtag sentiment classification approach. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, CIKM '11, pages 1031–1040, New York, NY, USA, 2011. ACM.

[32] D. E. Willard. New trie data structures which support very fast search operations. *J. Comput. Syst. Sci.*, 28(3):379–394, July 1984.

[33] W. X. Zhao, J. Jiang, J. Weng, J. He, E.-P. Lim, H. Yan, and X. Li. Comparing twitter and traditional media using topic models. In *Proceedings of the 33rd European conference on Advances in information retrieval*, ECIR'11, pages 338–349, Berlin, Heidelberg, 2011. Springer-Verlag.