Amir Hossein Akhavan Rahnama

Real-time Sentiment Analysis of Twitter Public Stream

ABSTRACT

Akhavan Rahnama, Amir Hossein
Sentinel: Distributed Real-time Sentiment Analysis for Twitter Stream
Jyväskylä: University of Jyväskylä, 2015, 84 p.

Sentiment analysis on Twitter public stream has been a topic of research recently. Several *non-commercial* libraries and software were developed to perform sentiment analysis, however none of them performed the analytics in real-time for Twitter data. Performing the same task in real-time can gives us insight of Twitter users public opinions regarding *recent* happenings of the time that analysis was made. In this thesis work, we propose a full-stack architecture with a software prototype that performs real-time sentiment analysis on Twitter public stream. We address the problem using large-scale online learning and specifically online parallel decision trees. Large-scale learning is utilized due to the fact that social media website such as Twitter produce data with high volume (around 5800 tweets per second in 2014) and in addition, there is a high time constraint (up to seconds) in real-time analytics in both learning, processing and query response time. Moreover, Twitter stream data arrives instance-by-instance and therefore we have utilized online learning with incremental and per-instance learning flexibility. SAMOA is a framework that provides support for a set of scalable online learning algorithms such as Vertical Hoeffding Tree. We use SAMOA's VHT learner with Apache Storm as our Stream Processing Engine. However, utilizing *only* VHT and Apache Storm cannot solve the problem at hand. Therefore, we *also* developed an open-source Java library called Sentinel that enables real-time Twitter stream reading, in-memory pre-processing computations and data structures, feature selection, frequent miner algorithms and etc. that completes our architecture. In Chapter 3, we show the architecture of our solution and its applicability and usefulness is shown in chapter 4.

Keywords: sentiment analysis, real-time analytics, social media mining, twitter, large-scale learning, parallel decision tree

ACKNOWLEDGEMENTS

I would like to thank my family for their endless support and love throughout my life. My beloved father, mother and sister: Mohammad Akhavan Rahnama, Maryam Toloo and Mahshid Akhavan Rahnama and my dear uncle, Dr. Mehdi Toloo, who has always supported my academic and scientific efforts throughout my life. I would also like to thank Professor Jari Veijalainen for his supervision. Thanks to Professor Tommi Kärkkäinen and a special thanks to Professor Timo Tihonen for his valuable inputs on proper academic writing practices and the common pitfalls one needs to avoid.

Jyväskylä 13.01.2014
Amir Hossein Akhavan Rahnama

ACRONYMS

API	Application Programming Interface
DDM	Distributed Data Mining
DSRM	Design Science Research Methodology
GFS	Google File System
I/O	Input/Output
IT/IS	Information Technology/Information Systems
PI	Processing Item
SPI	Source Processing Item
SPE	Stream Processing Engine
VFDT	Very Fast Decision Tree
VHT	Vertical Hoeffding Tree

FIGURE

TABLE

CONTENTS

1 INTRODUCTION

1.1 Background research

The trend of advanced data mining, as we know of today, started since 1990s in leading IT industry firms (Mena: 2011). According to Mena (2011), retail and finance companies were collecting data by applying relatively basic data analysis since 1980. Using the knowledge that came from data brought them opportunities, which resulted into their market dominance and customer satisfaction. This trend has accelerated even more. One straightforward explanation of this trend is the availability of vast amounts of data in companies' data warehouses. A lot of companies have explored data for competitive advantage, however due to the huge volume of existing data, processing approaches has been moved away from manual analysis toward computerized analysis. In addition, data volume in increasing and sources are no longer limited to human beings and devices are now a big player in the data generation ecosystem. On the other hand, Internet has become increasingly popular that the current data mining systems now include distributed and time-dependent information. Adding the distributed dimension to data mining systems shall introduce additional complexity to the data mining's processes.

Laney (2001) defined challenges and opportunities brought by increased data with a 3Vs model: the increase of Volume, Velocity, and Variety, which was later used as the basis of definition of big data. Beyer (2012) defined *big data* as

> High-volume, high-velocity and high-variety information assets that demand cost-effective, innovative forms of information processing for enhanced insight and decision-making.

Big data analytics has turned into an important utility for improving both efficiency and quality of different players such as companies, organizations and governments.

According to Murdopo (2013), in terms of *volume*, big data systems need to handle large amounts of data, which is generated at an increasing rate. Gens (2011) stated that

8 billion terabytes of data would be generated by 2015. In terms of variety, sensors and devices along with web applications are generating various types of data. In terms of velocity, big data systems need to process data with high speed of generation that flow into their systems.

1.2 Problem Identification and Motivation

Real-time analytics, a subset of big data applications, is when the newly arriving data needs to be included into the decision-making process within seconds. Meaning that, they need to deal with high volume of *recent* amounts of data to aid timely or instantiations decision-making (Mohammad and Al-Jaroodi, 2014). In intelligent transportation where the sensors are used to control traffics in cities or roads, decision-making needs to include the sensor data within seconds whereas in financial market surveillance, the system needs to be able to make decisions within milliseconds to detect opportunities in the market trend. See Mohammad and Al-Jaroodi (2014) for more examples. After Twitter announced its stream APIs in 2010, performing real-time analytics on Twitter also became possible.

According to Pang and Lee (2008), Sentiment analysis is "computational treatment of, sentiment, opinions or subjectivity in a source text". Sentiment analysis is generally defined as a classification machine learning task and we follow the classification scheme of the problem throughout this thesis work.

Sentiment analysis on Twitter public stream[1] has been of interest in research community. One reason can be the fact that users are always encouraged to write a post about their current happenings in life and as a result there is potential for users to review on ideas, products. Twitter is one of the important sources of dynamic and massive data among social media websites. In October 2012, Obama and Romney's presidential election debate created almost 10 million tweets in less than two hours. During the debate, Twitter users were posting their sentiment about different topics that were interesting to them such as healthcare[2]. In September 2014, Twitter was receiving around 500 million tweets per day from users[3]. This means that around 5800 tweets are processed every second. The mentioned facts shows that there exists an opportunity to perform real-time analytics on Twitter data to detect trends or perform tasks, namely sentiment analysis in a timely manner for public security, disaster prediction like Swine Flu epidemics in 2009, business disaster protection such as Toyota's Crisis[4], public polling, data driven journalism and marketing to name a few. According to De Francisci Morales (2013) and

[1] https://dev.twitter.com/streaming/public
[2] https://blog.twitter.com/2012/dispatch-from-the-denver-debate
[3] https://about.twitter.com/company
[4] http://en.wikipedia.org/wiki/2009–11_Toyota_vehicle_recalls

as we demonstrate in more details, none of the available non-commercial software can perform sentiment analysis in real-time on Twitter public stream in section 2.3.2.

1.3 Research question

As mentioned earlier, Twitter Stream data has high velocity and volume in real-time. Also, the instances arrive with non-uniform arrival rates that can vary over different time period. This means that a complete dataset is not ready before the learning starts and learning process should be incremental.

Our main research questions are the followings:

- In terms of feasibility of the task, is it possible to develop a software prototype according to a paradigm with a set of libraries to perform real-time sentiment analysis on Twitter Public Stream? It should be noted that we assume that a real-time computing engine is assuring and supporting the real-time communication among nodes in a distributed environment.
- In terms of learning in real-time, is it possible to use supervised learners to learn from real-time data with acceptable performance and accuracy in classifying sentiments?

1.4 Proposed Solution

It is apparent that real-time Twitter sentiment analysis can by tackled by large-scale online machine learning due to number of reasons: firstly, the stream data has time constraint while processed in real-time. By parallelization, processing and query time becomes shorter and more efficient. In addition, as mentioned earlier, the Twitter data is rather substantial in terms of size. In addition, learning models and algorithms with high-accuracy are complex and therefore expensive in terms of computation and in case it is distributed among several processing units, it can be computationally efficient. Also, the data arrives instance-by-instance and using online learning can provide incremental learning in our model. Online learning algorithms are faster compared to batch learners (Le Cun and Bottou, 2004). Specifically, online decision trees such as Hoeffding Tree has shown high performance and accurate results in classification task (Pfahringer et. al, 2007). In addition, according to Ben-Haim and Tom-Tov (2010) and Amado, Gama and Silva (2002), parallel decision trees reduce the time a tree is generated and therefore they are equally accurate than their serial counter parts and therefore highly applicable in this context. Proposed by Murdupo et al. (2013), Vertical Hoeffding Tree (VHT) *keeps* the accuracy measures of serial Hoeffding tree, improves its memory usage by parallelism.

In this research, we present a functional software prototype that enables the usage of online parallel decision trees in distributed environments. We follow SAMOA[5]'s scalable online learner, namely Vertical Hoeffding Tree. Apache Storm is our selected real-time computing engine, due to the fact that it ensures real-time computation in the stream data model (Gray et al: 2014). However with all the aforementioned selections, SAMOA lacks components that process Twitter Public stream and adapters that connect data, learning and processing stream engine. Not only we develop adaptors to connect learner and the engine, but also we present Sentinel[6] in this study. Sentinel is an open-source Java library that makes our architecture complete by enabling pre-processing, feature selection, frequent miner, sliding window models, stream abstraction classes and etc. While Sentinel is part of our research, it is not the only purpose. Sentinel is combined with SAMOA and Apache Storm to enable the real-time sentiment analysis. Therefore, Sentinel a *mean* to answer our research question.

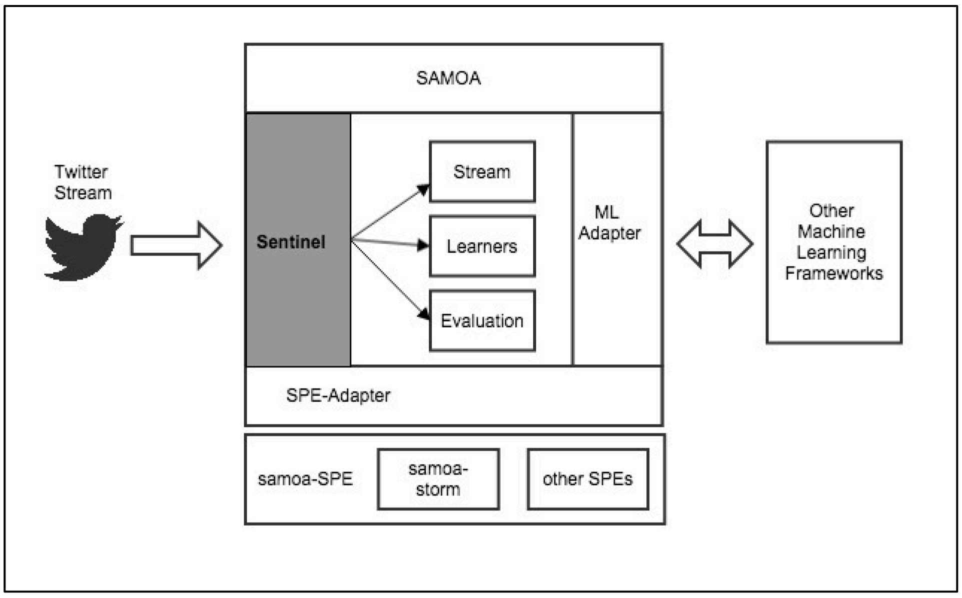


FIGURE 1. A bird-eye view of our solution

It should be mentioned that our large-scale online parallel decision tree approach is only one of the possible solutions that addresses real-time Twitter sentiment analysis problem, however at the time of writing this thesis, there are no other, open source solutions

[5] Apache SAMOA (Scalable Advanced Massive Online Analysis) is a framework and programming paradigm for scalable machine learning: http://samoa.incubator.apache.org/
[6] Full source code and documentation available at: http://ambodi.github.io/sentinel

to the research question with similar capabilities to sentinel. Furthermore, Sentinel is currently being merged with the SAMOA code base. It may be possible to consider other possible solutions and approaches but discussion of sentinel compared to other approaches is left out of the scope of this thesis work and can be a topic of further research.

1.5 Objectives of the solution

In this study, the followings are our solution's objectives:

- *Real-time analytics*: It guarantees to perform online analytics on stream data.
- *Accuracy and Performance in Learning*: We present a hybrid approach that can improve both accuracy and performance in Vertical Hoeffding Tree compared to its serial version. In other words, since VHT uses memory more efficiently, we present an approach that can use that free memory to improve accuracy measures of VHT compared to its serial version.
- Real-world dataset*:* The case study of this thesis work should be based on using Twitter Public Stream in real-time directly.

1.6 Overview of this thesis

This thesis work is organized as follows: In chapter 1, problem identification and motivation along with the proposed solution and its objectives were presented. In chapter 2, data stream mining along with large-scale learning and parallel decision trees are discussed. A section on related concludes this chapter. In chapter 3, each core component in our architecture with its corresponding algorithms is discussed. In chapter 4, the usefulness of our approach is shown in a case study and further research is proposed. We have an appendix at the end of this document to provide some basic necessary definitions, only to present enough information to ease reading of this thesis work.

1.7 Publications

Some of the results from chapters 1-3 of this thesis work are documented in the following publication:

- Rahnama, Amir Hossein Akhavan, (2014, Nov). Distributed real-time sentiment analysis for big data social streams, 2014 International Conference on (pp. 789 -

794). IEEE. In Control, Decision and Information Technologies (CoDIT), DOI: 10.1109/CoDIT.2014.6996998

2 EXTENDED BACKGROUND AND RELATED RESEARCH

2.1 Data Stream Mining

2.1.1 Data streams

Han and Kamber (2006) stated that continuous in-and-out flow of stream with varying update rates makes data streams different than other type of data models. According to Bifet et al. (2010), main sources of stream data are sensors and devices, traffic management systems, computer networks, user click and activity logs in front-end web applications, manufacturing systems, emails, blogs, social media (Twitter, Facebook, Wikipedia) to name a few.

According to Bishop (2006, pp. 234), stream data is classified into two different categories: stationary and non-stationary. In the *stationary* case, the data evolves in time, but the distribution from which it is generated stays the same. In *non-stationary* case, which is generally more complex to mine, the distribution that generates data also evolves with time.

2.1.2 Stream Data Model

A formal representation of stream can be denoted as

$$(a_1, uv_1), \ldots., (a_n, uv_n), \ldots$$

where at each point in time, t, and stream has a fixed length, a positive integer denoted as n. However, it is assumed that the value of length of the stream is not fixed and limits to infinity. Each instance of stream data is an element of an ordered list [7]called a *tuple*. In the representation, each instance is a key/value pair where a_i $(1 \leq i \ll n)$ is the attribute key that can hold a unique identifier and uv_i $(1 \leq i \ll n)$ is the corresponding attribute value(s), also called an *update values* that can hold values of types string, bits or integers (Liu et. al: 2011).

2.1.3 Stream Data Processing

According to Rajaraman and Ullman: (2012, pp. 132), in a data-stream management system, data instances arrive at a system with different arrival rates. It should be noted that arrival rates between instances are not uniform. One significant difference between data stream management system with regular databases is that data stream management systems have no control over arrival instances and rates in contrast with a database that controls the reading process from a disk by SQL insert commands, bulk loaders or etc. In contrast, data stream management systems should be developed in such a way that they can deal with different arrival rates for incoming instances. In practice of data streams processing however, there are chances of data being lost during that processing. Due to the assumption of potentially infinite number of available instances in a stream data, the processing on each instance should be fast enough so that instances are not lost due to latency in processing[8]. In the stream model, it is expected that processing on each instance be done in only few passes over the data. In this research, we follow the usage of one-pass algorithms. In *one-pass algorithms*, each arriving instance is examined once only and gets discarded by algorithm after that. One-pass algorithms keep the required space smaller compared to algorithms that use few passes over data.

As it can be seen in Figure 2, summaries of a subset of streams are in a working store and streams are archived into archival store. Based on Rajaraman and Ullman: (2012, pp. 132) and Han and Kamber (2006, pp. 468), we follow the common assumption in stream mining literature that queries[9] are answered based on the working store and the system *only* uses the archival store for answering queries in special conditions, such as when there is no interest in timely decision-making (Rajaraman and Ullman, 2012, pp. 134). In most cases, working store is either on disk or in main memory. In case it exists in main memory, it will return queries much faster with a trade-off that it will be smaller in size and therefore more compact with less data. The archival storage is so massive that we just store the input streams. It cannot be assumed that system can answer queries fast enough with that data, therefore the working set is the source of data for real-time analytics and archival set is the source to perform CRUD[10] operations in databases, batch processing and etc.

[7] Ordered by the timestamp which data was generated
[8] We also exclude cases where computing engine has large memory such as in massively scaled clouds or distributed environments such as in large companies such as Twitter.
[9] Queries in stream mining are discussed in section 2.1.6 in more details.
[10] Create, Read, Update and Delete are main functions while working with databases.

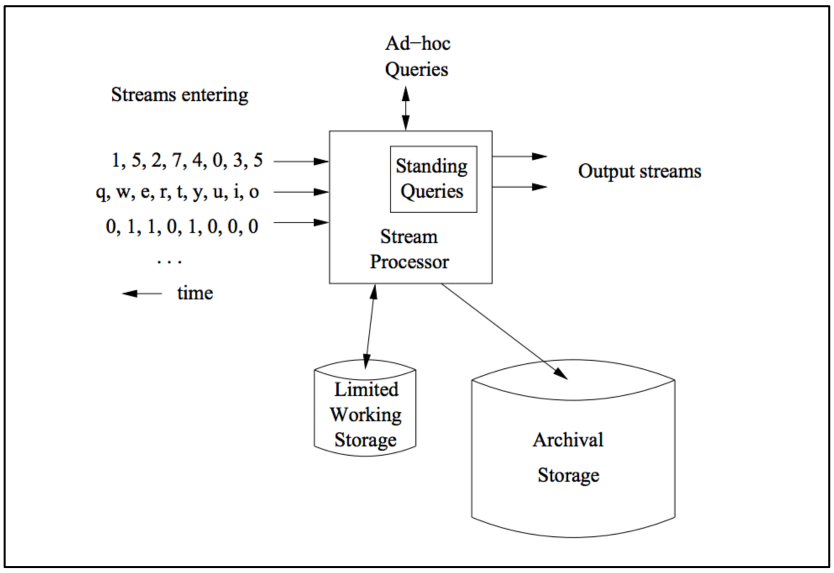


FIGURE 2. Anatomy of a data-stream management system (Rajaraman and Ullman, 2012, pp. 132)

2.1.4 Synopses

Gibbons and Matias (1999) introduced a type of data structures called synopsis data structure, which is substantially smaller than the base data sets. Synopsis resides in main memory and is used to answer queries faster where timely responses are required. As mentioned earlier, answering queries with archival set is time-consuming. Synopsis is the in-memory working sets in stream data processing. Stream processing algorithms use this data structure to keep an optimum usage of memory. *Synopses* are of utmost important, since they can efficiently hold summaries of the stream data. The relationship between queries and synopsis is explained in chapter 3.

2.1.5 Sliding window

Sliding window is one of widely used approaches in stream mining in order to obtain a sample of data stream. The sliding-window model works based on recent data and therefore it is optimized for real-time stream processing. In this model, the processing and analysis of the instances is only done on recent data within a fixed slice of time. In many applications areas, such as financial forecasting, predicting the next value in a time series is dependent on observations of the previous values. Generally in such cases, to predict future values, it is expected that recent observations are likely to be better predictors (i.e. more descriptive) than historical observations. Here, we discuss the fixed and adaptive sliding windows, although other approaches such as Landmark and Damped are exist.

2.1.5.1 Fixed Sliding Window

According to Han and Kamber (2006, pp. 470) in *fixed-size sliding window model* with size N. With arrival of each instance, time step value is increased by one (zero is its starting value). Each data instance will expire after exactly N time steps. The data that is used for answering queries is the last N instances that have arrived.

2.1.5.2 Adaptive Window

Bifet and Gavalda (2007) showed that setting the length of the window is not an easy task. This is because stream data flow dynamically resulting in different rate of instance arrival at different points in time. To use fixed-size sliding-window more efficiency, three approaches can be followed: First is choosing a small size to increase accuracy of the model. Second is to choose a large size so instances will be accessible longer which results in stability of recall. Third is to use a *decay function*[11] to assign a weight value to instances with regards to the time they were generated. In the third solution, the user still needs to set the decay value in a way that it matches the rate of change, which is not known in non-stationary data streams (Datar and Motwani, 2007).

Bifet and Gavalda (2007) proposed ADWIN that dynamically sets the size of the window based on incoming data. As you can see in Figure 3, ADWIN needs a confidence value as input (δ) and initializes an empty list of buckets and keeps most recent instances along with simple statistics measures such as width, variance and total number of instances in the window. With coming of new instances, old instances are dropped from the window, however statistics of the entire stream is stored in memory. ADWIN splits the window into two sub-windows in all possible ways to find that two sub-windows show noticeably large difference in mean values of their corresponding instances, based on a statistical measure called ε_{cut}. When this happens, ADWIN drops the old window.

The statistical measure used in ADWIN, ε_{cut} (Hoeffding bound) was proposed by Hoeffding (1963) and it can be calculated from the confidence value (δ) for sub-windows W_0 and W_1. At each time, number of instances that belong to W, n, is the sum of number of instances in W_0 (n_0) and W_1 (n_1) or formally ($n = n_0 + n_1$):

$$m = \frac{1}{{}^1\!/_{n_0} + {}^1\!/_{n_1}} \qquad (1)$$

$$\acute{\delta} = \frac{\delta}{n} \qquad (2)$$

[11] i.e. exponential decay decreases at a rate proportional to its current value. Exponential decay can be formulated as $\frac{dN}{dt}$. It should be noted that N is the quantity and λ is a positive rate, which is called the *exponential decay* constant.

$$\varepsilon_{cut} = \sqrt{\frac{1}{2m} \cdot \ln\frac{4}{\delta}} \quad (3)$$

ADWIN’s processing time for each instance is $O(log|W|)$ and uses $O\left(\frac{1}{\varepsilon_{cut}} logW\right)$ memory words. See Bifet and Gavalda (2007) for more theorems and details.

```
Input: Confidence value ($\delta$), Stream ($S$), Instance ($I$)
Output: Observed Mean Value of window ($\tilde{\mu}_W$)

initialize $W$ = empty list
    while ($S$ has $I$)
        insert I to head of W
            repeat (delete I from the tail of W and Update
                    statistical measures)
            until ($\left|\tilde{\mu}_{W_0} - \tilde{\mu}_{W_1}\right| \geq \varepsilon_{cut}$)
        for every possible sub-window of W ($W = W_0 \cdot W_1$)
```

FIGURE 3. ADWIN algorithm (Bifet and Gavalda: 2007)

2.1.6 Stream Queries

Two types of queries can be made over stream data: ad-hoc and standing. *Ad-hoc query* is made one time and *standing query* is made continuously in a loop on the stream data. Iceberg queries and top-k queries are two common forms of queries in data streams. We follow the definitions of these queries from Liu et al (2011):

Definition 1: (iceberg query) Suppose *S* is an input stream of length *n*. Also, consider a support threshold g ∈ (0, 1). Return the instances, which their frequencies, i.e. number of appearances are not smaller than $\lfloor \varphi n \rfloor$[12].

Definition 2 (top-*k* query): Suppose *S* is the input stream, return an ordered list of *k* with most frequent instances.

[12] The notation $\lfloor$ corresponds to floor operation

2.1.7 Frequent Item Algorithms

Liu et al. (2011) categorized frequent item algorithms into three unique categories, namely sampling-based, counting based and hashing based algorithms. In this research work, we only focus on counting-based algorithms, therefore for more details see Liu et al. (2011).

Liu et al. (2011) stated that counting-based algorithms have counters that monitor instances in data stream. In case the algorithm selects an instance, a counter has the responsibility of counting each of its exact occurrences. Since memory is limited, clearing methods are used to keep the memory clean of infrequent instances or features[13]. If not, such instances will occupy the limited memory space, which can potentially become a shortcoming of this type of algorithms. In case clearing methods are performed inconsistently, some potentially frequent instances that are currently infrequent can be deleted by mistake. This has a negative effect on the results. In these types of algorithms, the goal is to design algorithms that can handle and adapt to changes efficiently.

Metwally et al. (2005) proposed *Space-Saving* (Figure 4) for finding most popular k instances, namely top-k instances that occur more than a predefined threshold (*min*). The space-saving algorithm (Figure 3) is a counter-based algorithm and it has a synopsis data structure (D) that keeps the summary of a subset of m stream instances in a stream with length n, where m is noticeably smaller than n. Since even in most optimal cases, some of the stream instances will be lost during stream processing, Space Saving *estimates* the frequency of instance, $F(i_j)$ where $1 \leq j \leq m$ by its estimated value, namely $count(i_j)$. The algorithm adds m distinct instances into D. After that with arrival of each instance, Space Saving increments the count value of that instance, if the instance is monitored. If not, it replaces it with the instance that has at least *min* as it *count* value. Since Space Saving overestimates the frequency of the element that is not monitored, it keeps an error rate (ε) value as the "maximum over-estimation error". The error rate for each instance is used to answer *top-k* queries where an instance is considered frequent if its corresponding sum of values of $count$ and ε is greater than or equal to the product of threshold and length of the stream. The space required for Space Saving is $O(m)$. See Metwally et al. (2005) for proofs regarding space usage and more details.

[13] In most cases, features in text mining correspond to words and token. This is explained in chapter 3 and 4 in our case in more details.

```
Input: Stream (S) with length $n$, minimum frequency threshold ($min$), number of monitored elements ($m$)
Output: Frequent items with threshold $min$

initialize a dictionary synopsis D in form of ($i$, $count(i)$, $\varepsilon(i)$) where $count(i)$ is the estimation of frequency of $i$ and $\varepsilon(i)$ is the error rate of instance $i$

for each instance ($i$) in S
    if $i$ is monitored then
        $count(i) = count(i) + 1$;
    else if D has an empty counter
        add new counter ($i$, 1, 0) to $D$
    else
        replace $i$ with $i_m$ with $min$ frequency in $D$
        $count(i) = count(i) + 1$
        $\varepsilon(i) = min$
    $length = length + 1$;

Output $i$ in $D$ where $count(i) + \varepsilon(i) \geq min * length$
```

FIGURE 4. Space Saving algorithm (Metwally et. al: 2005)

2.1.8 Stream Data Classification

For stream data classification, we follow the definition by Dietterich (2002). Suppose a stream x and let $\{(e_i, uv_i)\}_{i=1}^{n}$ be a set of n instances. The goal of stream data classification is creating a classifier h that can accurately predict a new category label $y = h(x)$.

According to Mena-Torres and Aguilar-Ruiz (2014) classifying data streams can be performed with two approaches: single classifier-based and ensemble-based approach. In *single classifier-based* approach, the main goal is to build a model that is from small portions of the data stream and incrementally update the model using new instances that are arriving to the model. For *ensemble-based approach*, many base classifiers are built for different parts of the data stream, and then all base models are merged to form an ensemble of classifiers.

2.2 Large-scale machine learning

According to Bekkerman et al (2011), there are several reasons that machine-learning algorithms need to scale up:

- Data is growing is size that puts barrier into single machine processing. Scaling up machine learning improves performance compared to single machine learning algorithms.
- Features in data are growing in number. This particularly happens in text mining that it can reach to more than thousands of features. One way to handle this is exclude features that have less impact on the analysis. However, scalable algorithms partition instances in several processing units. This not only improves performance but also allows us to keep all features available in data to gain a better representation of data.
- In order to improve accuracy, learning algorithms are growing in complexity. Scaling up these algorithms give more freedom to work with complex algorithms especially in online learning approaches with incremental learning curves.
- Real-time analytics puts time constraint on the time the data was generated. The constraint can vary from milliseconds to seconds (Mohammad and Al-Jaroodi, 2014) and latency in processing and learning can result in loss of valuable information. Scaling up those algorithms by parallelization can greatly benefit the analytics with faster processing and less data loss.

To be able to use scalable algorithms we need to utilize parallel or distributed systems that are able to execute machine-learning task in concurrence. To achieve this, research has suggested two main approaches: data parallelism and task parallelism. In former, multiple partitions of data are fed into the algorithm and they are processed concurrently. In the latter, different parts of algorithm are divided into independent segmented that can run concurrently (Bekkerman et. al, 2011, pp. 7-8).

In following sub-sections, we discuss possible distributed processing engines that have potential to be used for online scalable machine learning approaches, namely modified version of MapReduce and SAMOA. After that, we discuss parallel decision trees and different type of parallelism in decision trees. We also discuss vertical parallelism as a potentially reliable solution since we deal with texts that can include high number of features. After that, Vertical Hoeffding Tree is discussed, as it is our parallel learning algorithm that we have selected to apply on Twitter public stream data. We conclude this section on Adaptive Bagging ensemble classification approach that combines several Vertical Hoeffding Tree classifiers to obtain a better accuracy with a trade-off in using more memory and performance. Since we follow a large-scale approach by parallelization in a distributed environment, the proposed trade-off can be neglected in our approach.

2.2.1 MapReduce & Hadoop

Dean and Ghemawat (2004) proposed MapReduce model as a programming model for processing datasets that are both large in scale and distributed. MapReduce works with a parallel-distributed algorithm, which runs on many clusters. MapReduce has two main functions, *map* that filters and sorts data, and *reduce* that summarizes the result of mapped function. Apache Hadoop[14], the most widely adapted and popular implementation of MapReduce, is a platform with high reliability and is widely utilized in big data systems[15]. However MapReduce programming paradigm follows the batch-processing model. It should be mentioned that MapReduce paradigm supports scalability, and offers fault tolerance mechanisms, which makes it a sound platform in distributed environments.

MapReduce programming model is based on batch processing model that is not the model we focus in our study. Several attempts have been made to make Hadoop and its MapReduce model a candidate for stream processing and adding feature to make it applicable to online processing model. Aly et al (2012) suggested M3 prototype to overcome issues of pre-storing data before Hadoop jobs in Hadoop Distributed File System (HDFS) that causes a delay time. The authors suggested an approach in M3 that bypasses HDFS and handles the processing of stream data in the memory. M3 also supports continuous MapReduce jobs that make M3 applicable for real-tine online stream processing. M3 is just a closed-source laboratory based project and is not yet published and therefore inapplicable in a production environment. Condie et al (2013) proposed *Hadoop Online* that allows “online aggregation” during batch processes that results in the ability to monitor and control the execution of the jobs in real-time. Their prototype also supports continuous queries that make Hadoop Online a sound platform for stream mining. According to Hadoop Online homepage[16], Hadoop Online is an early work-in-progress prototype and no support is provided for utilizing it in neither external projects nor production environments.

2.2.2 Apache Storm

Apache Storm is a distributed real-time computing platform that enables real-time pro-

[14] http://hadoop.apache.org/
[15] A long list of DDM projects that use Hadop can be found at: http://wiki.apache.org/hadoop/PoweredBy
[16] https://code.google.com/p/hop/

cessing of data. Marz (2011) proposed Storm as a computing engine for distributed stream data mining systems. According to Apache Storm documentation, *Apache Storm*[17] is a distributed real-time computation system that facilitates reliable processing of unlimited streams of data. Real-time analytics and online machine learning are just a few domain of Storm's multiple use cases. Gray et al. (2014) has proven that Apache Storm ensures real-time computation by following acyclic graph of transformation for each processing task. The limitation of using Apache Storm for all real-time big analytics scenarios is that Storm *only* guarantees the real-time computing if stream data model is used.

2.2.3 SAMOA

De Francisci Morales (2013) proposed SAMOA as an abstraction layer and a scalable machine-learning Java library. SAMOA is released by Yahoo and provides flexibility and abstraction to interact with stream processing engines. We follow SAMOA's conventions in the upcoming sections while using Apache Storm since it adds flexibility and re-usability while working with stream processing engines such as Apache Storm. It also has parallel learning algorithms that can be used on top of our Stream Processing Engine.

2.2.3.1 SAMOA conventions

SAMOA defines some key concepts for real-time distributed stream mining applications (Figure 5). More comprehensive concepts are discussed in details in Murdopo et al (2013).

In SAMOA, each distributed system has a topology. *Topology* is a construct, which contains a set of processing items and stream. A *processor* is in charge of executing parts of algorithm on a specific Stream Processing Engine (*SPE).* Processors contain the logic of the algorithms. *Processing Items* (PI) implement processors. Systems pass different content events via streams. It should be noted that processing items could be inter-connected.

A *stream* is an unlimited continuity of dynamically typed and serialized list of values[18] called *tuples*. Streams can be seen as connectors between PIs and other components that send different content events between PIs. A *content event* is a wrapper object around the data that is passed on from one PI to another.

[17] http://storm.incubator.apache.org/

[18] Serializing the tuple ensures that the object is in a state that it can be used in different processors even if they are on different physical machines. Dynamically typed listed list do not put any limits on data type that can be stored and retrieved from the list and therefore add more flexibility while working with them.

There are two types of node in SAMOA's paradigm: master and worker. Master node is where the topology is submitted and it takes the responsibility of distributing code for other nodes. A worker node is a Java Virtual Machine process that executes parts of the topology.

A source PI is called a spout. A *spout* passes content events through stream and reads data from external sources. Each stream has at least one spout. A *bolt* is consumer of spout(s) and it can join, filter or aggregate different instances. It can also communicate with other data sources such as Database Management Systems (DBMS).

Bolts need *grouping* mechanisms, which sets how will the stream routes the content events. In *shuffle* grouping, stream passes the content events in a round-robin manner to target PIs. In *all* grouping, after replicating the content events, stream is passed to all PIs. In *Key* grouping, the stream is passed according to its key, meaning that content events are passed to the same PI if they have equal key identifiers.

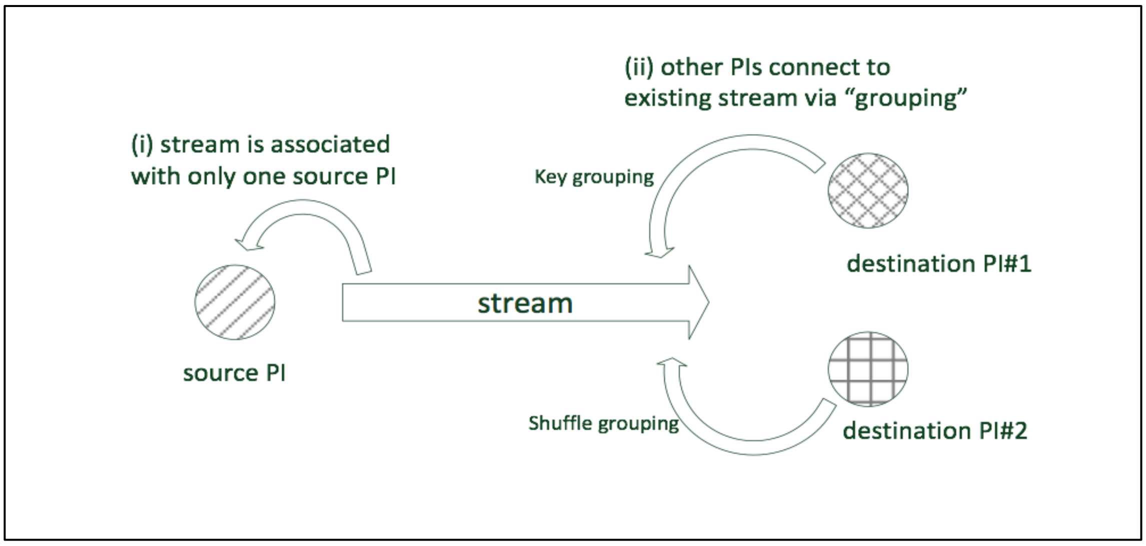

(ii) other PIs connect to existing stream via "grouping"
(i) stream is associated with only one source PI
Key grouping
destination PI#1
stream
source PI
Shuffle grouping
destination PI#2

FIGURE 5. Instantiation of a Stream and Examples of Groupings (Murdopo et al: 2013)

Transformation of stream between spouts and bolts follows the *pull* model, i.e. the corresponding bolt needs to pull tuples in the content event from the source processing items. Therefore the incoming instances are lost in spouts, in cases that bolts are unable to process instances fast enough to capture all incoming instances from them.

2.2.4 Parallel Decision Tree Induction

Decision trees are reliable and efficient methods for classification (Ben-Haim and Tom-tov 2010; Amado, Gama and Silva, 2002). One of their key features is that they include readable rules for humans. One of the main drawbacks of decision trees while working with large data is that they need to sort numerical attributes which causes computational overhead. Parallel decision trees like serial decision trees perform sorting in advance, or distributed or they use approximation techniques. Parallel decision trees have shown to improve performance significantly in text mining scenarios (Murdopo, 2013 and Ben-Haim and Tom-tov, 2010).

According to Ben-Haim and Tom-tov (2010), parallel decision trees have several approaches for parallelization: horizontal, vertical, task and hybrid. We briefly mention each type along with their advantages and disadvantages and for more details on each parallelism type, see from Murdopo et al (2013).

In *horizontal* parallelism, each instance will be sent to one and only one processor. The advantage of horizontal parallelism lies in the fact that it is suited for very high arrival rates of data. This type of parallelism is also flexible in the fact that it allows the user to add more processing power by increasing the parallelism level. The drawback is excessive usage of memory as it keeps several duplicated model instances[19] in each processor.

In *vertical* parallelism (Figure 6), each attribute of incoming instance will be sent to one and only one processor. Vertically parallelizing a decision tree introduces several drawbacks. First shortcoming is that increasing level of parallelism cannot result in any performance gain in most cases, because splitting and distributing nodes requires a lot of computing power. Another drawback is that it is not optimized for cases where numbers of attributes are not high enough. The main application area of vertical parallelism is in text mining where number of attributes within each instance is high up to thousands. Vertical parallelism is suitable when working with data types such as texts with high number of attributes. Another advantage of vertical parallelism is less usage of memory due to the fact that the there is only one instance of the model for all processors.

In *task* parallelism, decision tree nodes will be distributed to processors. Task parallelism is only useful in case the model is complex up to a level that it has a bigger size than available memory. The disadvantage of task parallelism is that it needs a longer period of time to efficiently use the parallel processing power.

[19] In this context, the model is the decision tree model

Hybrid parallelism is a combination of vertical and horizontal parallelism. It uses each of parallelism types to overcome their shortcomings to achieve better efficiency in terms of performance.

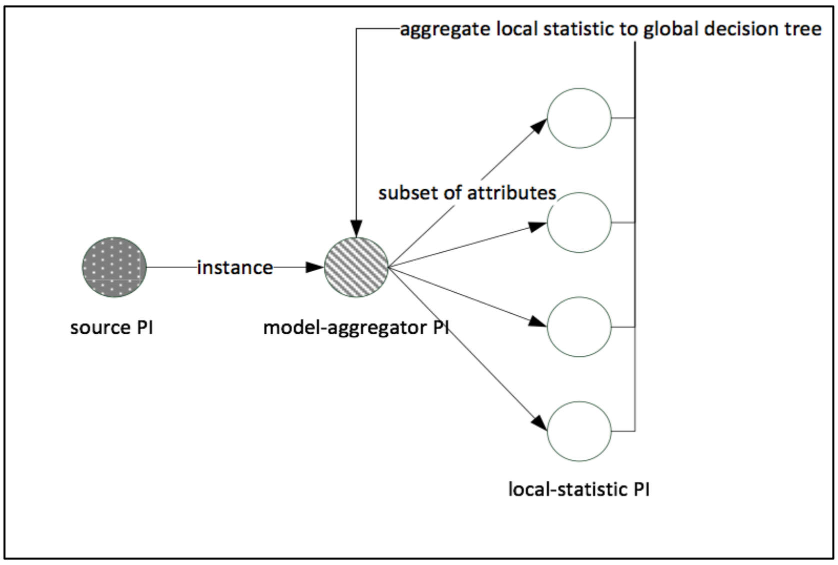


FIGURE 6. Vertical Parallelism (Murdopo et al: 2013)

2.2.5 Vertical Hoeffding Tree

Murdopo et al. (2013) proposed Vertical Hoeffding Tree (VHT), a parallel decision tree algorithm based on VFDT learner. Domingos and Hulten (2000) proposed VFDT for mining high-speed data streams. Cohen et al. (2008) and Last (2002) presented impressive results with performance and accuracy of VFDT in online classification of data streams scenarios.

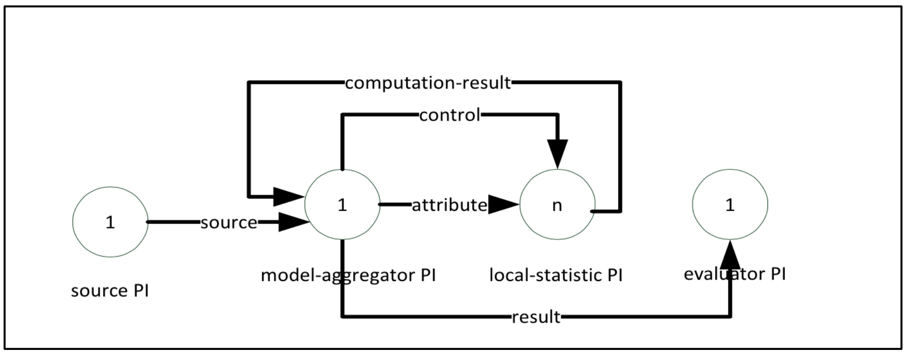


FIGURE 7. Vertical Hoeffding Tree components and streams (Murdopo et al: 2013)

As you can see in Figure 7, VHT has four key processors: source, model-aggregator, local statistic and evaluator along with four messaging streams: computation-result, control, attribute and result. Source fetches the data from a stream data source, instance by instance. It then sends the instance to model aggregator, which then splits the instances based on their attribute. After that, the attribute is sent through attribute stream to local-statistic processor. As with Hoeffding Tree algorithm, the nodes in the model-aggregator of VHT are created dynamically with every n_{min} of new arriving instances and in case they belong to different classes. Model-aggregator node continuously sends instance information to local-statistic PIs and once it is time to split, it uses the compute-event stream to send the information about the leaf that needs splitting to local-statistic nodes. This information includes list of attribute with their update values, the class that the instance belong to along with the estimated accuracy of the classification, i.e. instance weight (Figure 8).

```
Input:
Vertical Hoeffding Tree (VHT) in its current state
Instance (Inst) =$\{\{(a_i, uv_i,)\}_{i=1}^{total}, class_{inst})\}$ where total is number of attributes
Output: compute tuple

initialize list of splitting nodes ($list_{split}$)
sort Inst into a leaf ($l$) by VHT
set $l_{id}$ to a unique identifier
$l_{num}$ = 0 (number of instances seen at leaf $l$)


send attribute = $(l_{id}, \{(a_i, uv_i,)\}_{i=1}^{total}, class_{inst}, instance\_weight)$ to local-statistic node
$l_{num}$ = $l_{num}$ + 1;
grow = ($l_{num}$ mod $n_{min}$ === 0);

if (grow = $true$)and $\left|\left\{class_{inst_j}\right\}_{j=1}^{n_{min}}\right| \geq 1$
  add $l$ to $list_{split}$
  send $compute = (l, l_{id})$ to local-statistic with compute stream
```

FIGURE 8. Model-aggregator – Phase 1 (Murdopo et al., 2013)

Local-statistic PI keeps a dictionary of leaf IDs and their corresponding attributes (*dict*) and updates the values upon receiving the *attribute* tuple. When the compute tuple is sent, $\bar{G_l}$ is computed for all attributes in l_{id} that are in *dict*. First two highest attributes with highest $\bar{G_l}$ are sent to model-aggregator node (see Figure 9).

```
Input: attribute or compute tuple
Output: local-result tuple

initialize hash table of $< l_{id}, \{(a_i, uv_i,)\}_{i=1}^{total} >$
while (attribute event is received) do
        update $dict = < l_{id}, \{(a_i, uv_i,)\}_{i=1}^{total} >$ with attribute and
        class value and instance weight.

while (compute event is received) do
        for each $a_i$ $(1 \leq i \leq total)$ in $l_{id}$ where total is the num-
        ber of attributes, compute $\bar{G_l}(a_i)$ if and only if $l_{id} \in dic$

        find highest and second highest values of $\bar{G_l}(a_i)$
        In $a_i$ $(1 \leq i \leq total)$

        send    $local - result = (a_{max_1}^{local}, a_{max_2}^{local})$    to    model-
        aggregator via computation-result stream
```

FIGURE 9. Local statistic PI, (Murdopo et al., 2013)

Model-aggregator (Figure 10) sets the new highest and second highest attribute in the splitting node. When the local-result tuple is received, after computing the Hoeffding bound and checking if the difference of those attributes are greater than the bound, it will perform an split on the node with the highest $\bar{G}$ (information gain) value[20].

There are two important remarks that should be mentioned: first is that an upper bound for tie breaking (τ) is set as input. According to Domingos and Hulten (2000), Hoeffding Tree can exhaust performance by splitting on nodes with very similar values of the first and second highest values of $\bar{G}$. To prevent this, an upper bound is set for tie breaking and therefore the algorithm performs the splitting function more efficiently. Second is that model-aggregator has a built-in time-out mechanism that in case local-result tuples from *all* the local-statistic nodes are not received, it will perform the splitting by statistics from local-statistic nodes that it has received.

[20] see Appendix for details on information gain

```
Input: local-result
Upper bound on tie breaking ($\tau$)
$\bar{G}$, Information gain function
Vertical Hoeffding Tree (VHT) in its current state

for $l$ in $list_{split}$ do
            $a_{max_1} = a_{max_1}^{local}$
            $a_{max_2} = a_{max_2}^{local}$

while (local-result is received or time-out)
    compute $\varepsilon$ (Hoeffding bound)
    $a_{\emptyset} = \bar{G}(l)$, // $\bar{G}$ of no-split scenario
    if $a_{max_1} \neq a_{\emptyset}$ and $\bar{G}(a_{max_1}) - \bar{G}(a_{max_2}) < \varepsilon$ and $\varepsilon < \tau$
        replace $l$ and $l_{new}$ = the split node on $(a_{max_1})$
        for each branch in $l_{new}$ do
            insert leaf $l_{new}'$
            set $l_{new}'$(stats) = $l_{new}$ (stats)
        end
    end if
end while
```

FIGURE 10. Model-aggregator – Phase II (Murdopo et al., 2013)

While the model is executed in testing, VHT predicts the value of classes in incoming instances via model-aggregator node. Model-aggregator sorts input instances into the leaf and predict their class value. The result will be sent in *result* stream and sends it to evaluator processor.

2.2.6 Adaptive Bagging of VHT

According to Bifet et al. (2009), adaptive bagging methods are ensembles methods that combine several models prediction value to provide a final value that is more accurate and they introduce mechanisms that can handle concept drifts.

Bifet et al. (2009) proposed the usage of ADWIN bagging using Hoeffding trees. In our study, we use bagging ensemble methods with Vertical Hoeffding Tree. In this approach, each node in VHT has its own ADWIN. During the execution of algorithm, error rates in classification at each node are stored. Any sudden increase in classification errors indicates change in data. At this stage, a new tree is cloned without splitting any attribute that has its own ADWIN. After ensuring that the new tree shows decrease error values, the new tree is replaced by the old one. Bifet et al (2009) showed that ADWIN bagging using Hoeffding Trees could increase accuracy with a trade-off in runtime memory usage.

Although adaptive bagging is an exhaustive process for learning algorithm, the improved accuracy is of great value in our scalable online learning approach, because online learning algorithms have less accuracy over their batch counter-parts (Le Cun and Bottou, 2004). Since we follow a parallel learning approach, the working memory is less exhausted. This means that additional memory and computation power can be added with more freedom to the system to improve accuracy as well as it performance in our parallel online learning process.

2.3 Related research

Numerous studies have been conducted on sentiment analysis on Twitter, however we only mention the ones with a direct influence on this thesis work. In the first section, we discuss sentiment analysis research on Twitter and in the second section, we mention the available non-commercial software and libraries for Twitter sentiment analysis to show the need for our research in real-time analytics on Twitter stream data that with support for large datasets.

2.3.1 Twitter Sentiment Analysis

Go et al. (2009) presented a batch learning maximum entropy approach on sentiment analyzing the Twitter data. The authors have used Twitter Search API instead of Twitter Stream API[21]. They created a sample dataset that has both a training set and a test set with Twitter search API. It includes 1 million and six hundred tweet instances that is *balanced*, meaning half of tweets have positive and the other half have negative sentiment. The study considers the Twitter data in context of n-gram language models, specifically unigram and bigram models and the accuracy of keyword-based, Naïve Bayesian, Maximum Entropy and SVM are compared. The authors proposed their solution as a useful tool for buyers who want to search for the opinions of other customers regarding the same products. Their solution is also useful for companies that want to monitor specific products or brands. In the study, Naïve Bayes, Maximum Entropy and SVM were showing significantly accurate measures up to 80%.

Bifet and Frank (2010) conducted a research on comparing Multinomial Naive Bayes, SGD and Hoeffding tree in terms of time, accuracy and Kappa[22] measures to perform

[21] https://dev.twitter.com/docs/api/1/get/search
[22] see appendix for more details on Kappa

sentiment analysis on a the sample that was provided in Go et al. (2009). Authors recommended the SGD-based models along with the usage of sliding window models for Go et al (2009)'s dataset. This study also showed high potentials in using Hoeffding Tree for sentiment analysis with acceptable performance and accuracy rates.

2.3.2 Stream Mining Software for Stream mining

There have been mainly two non-commercial software and libraries proposed for the Sentiment analysis for Twitter stream, namely MOA and RapidMinder. We discuss both of these solutions along with their shortcomings for our research in real-time Twitter stream sentiment analysis.

Bockermann and Blom (2012) proposed a Twitter plugin for RapidMiner[23], which uses Twitter Stream Data. The modeling of the stream follows the single-pass paradigm. The authors proposed a data and control flow (see Figure 11-12) with a prediction layer on top of a batch Naïve Bayes Learner. Prediction Service component add a prediction value to each instance. Moreover, Prediction Error has the responsibility of computing the error of prediction and labeling process. The RapidMiner plugin processes stream instances in different file formats such as CSV files and after that, user can perform learning on the stored Tweet instances. One of the shortcomings of RapidMiner plugin is the fact that it only offers classification with the serial Naïve Bayes learner. The other one is the fact that there is a limit for data files that it can process (see documentation for more details)[24].

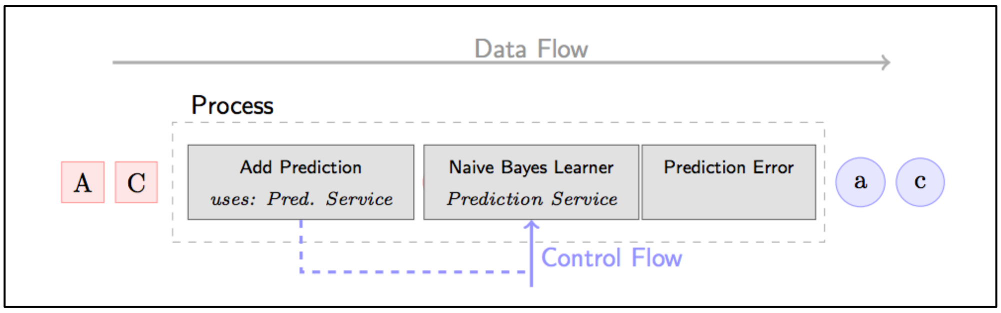


FIGURE 11. Control and data flow for data processing in RadpidMiner Twitter plugin (Bockermann and Blom: 2012)

[23] The plugin works on top of Rapidminer package, more details at http://rapidminer.com/
[24] https://rapidminer.com/documentation/cloud/twitter/

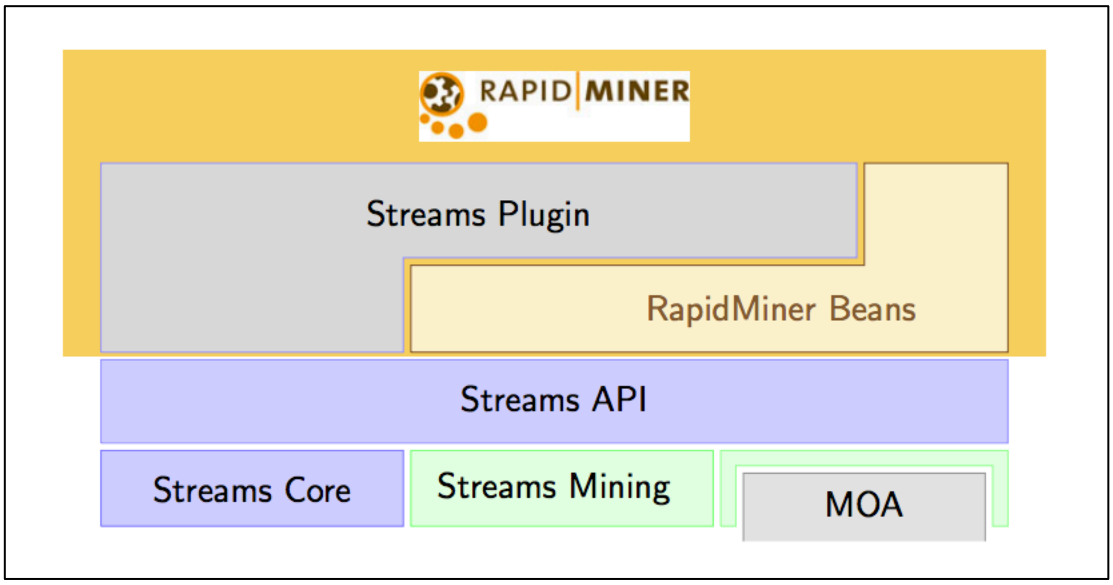


FIGURE 12. Architecture of the Streams Plugin in Twitter Plugin for RapidMiner (Bockermann and Blom, 2012)

Bifet et al. (2011) proposed a plugin for mining tweets called MOA-TweetReader with a set of serial online learning algorithms. The authors showed the usage of MOA-TweetReader with Go et al (2009)'s dataset in Bifet et al. (2011). The solution[25] includes an adaptive filter and a change detector that has the responsibility of detecting sudden and gradual changes. MOA-TweetReader runs a batch processor and stores the instances in an ARFF file at first. After that, user executes the learning process that includes a set of online learners that run machine-learning tasks over the stored dataset. It should be mentioned that the software package has been non-functional since 2013, due to several reasons such its incompatibility with Twitter APIs[26] and several considerable software bugs in its components. Above that, none of the algorithms or processing components supports any scalability (or other approaches) by parallelization of tasks or algorithms. This makes MOA and its Twitter plugin, MOA-TweetReader applicable in working on small datasets only and inapplicable for large datasets.

The most important shortcoming of both previously mentioned plugins is the fact that they read their data from a file. This means that the system has full control over the reading process and there are no chances that instances are lost. Therefore, performance is never a critical issue for such systems. As mentioned in section 2.1.3, this is in contrast with real-time stream mining cases, where loss of data is an important factor to discard algorithms for their low performance rate. This makes both solutions inapplicable in Twitter's real-time stream mining or more specifically sentiment analysis.

[25] Available at http://sourceforge.net/projects/moa-datastream/files/MOA-TweetReader/
[26] https://groups.google.com/forum/#!topic/moa-users/RZ4EbG_8xco

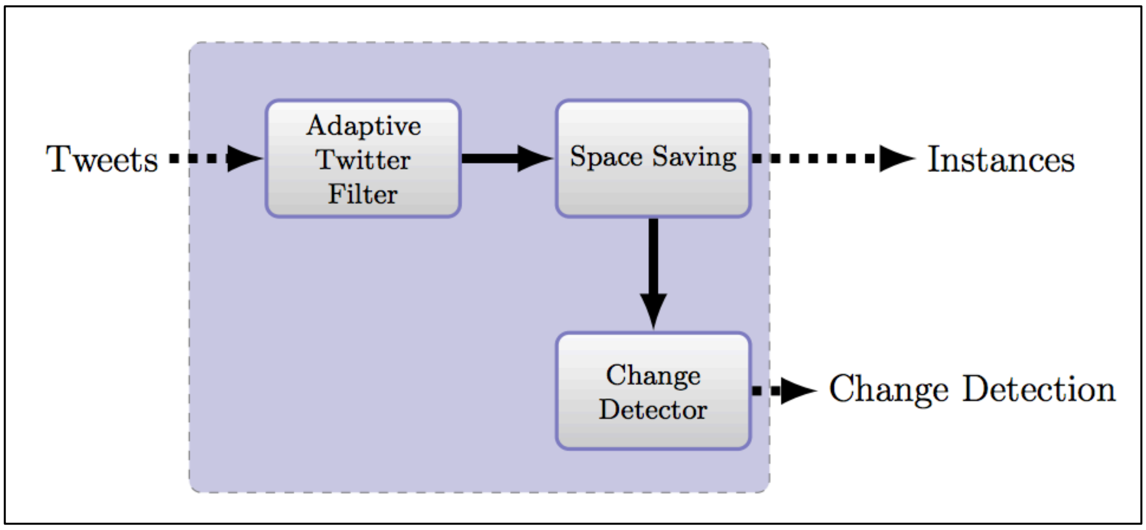


FIGURE 13. MOA-TweetReader (Bifet et al, 2011)

3 ARCHITECTURE

In this chapter, we propose a full-stack architecture of a software prototype that can perform real-time sentiment analytics on Twitter public stream (Figure 1). We divide the architecture into three components of a pyramid (Figure 14). First is Apache Storm, our stream processing engine that is responsible for communication among processors and nodes in a distributed environment. Second is SAMOA, our scalable machine-learning library that comprises a set of learning algorithms that can be used as with an API. On top of that is Sentinel, our open-source Java library that offers stream readers, query monitoring, windowing models, frequent item algorithms, feature selection that is missing to make the actual data digestible with SAMOA learners and evaluators for the whole architecture to perform the task efficiently.

We already have discussed SAMOA and Storm's conventions in section 2.2.2 – 2.2.3. In this chapter, we analyze each part of the pyramid separately and in more details. We also provide some pseudo-like sample Java code of some parts of each component.

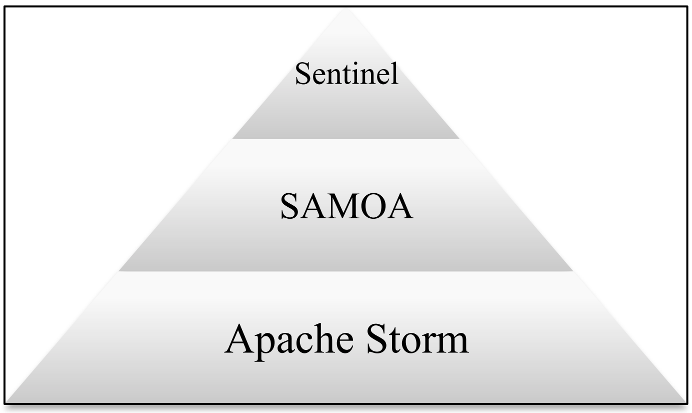


FIGURE 14. Full stack architecture pyramid

3.1 SAMOA-SPE

SPEs offer different and through fault-tolerance mechanisms that handle possible issues in node communications, since any such deficiencies can become a bottleneck for the system. Our selection of SPE is Apache Storm that has the capability to offer real-time processing of stream data model (Gray et al, 2014). As mentioned earlier, Storm includes a set of stream, spouts and bolts. You can see our Storm topology in Figure 15.

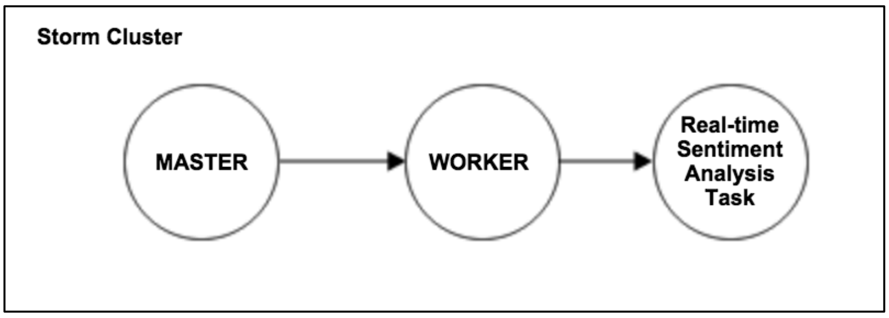


FIGURE 15. Our Storm Cluster's topology

3.1.1 Master

As it can be seen in Figure 16, we submit the topology to the master node and our worker node is where we initialize data and resources that are needed to perform our real-time sentiment analysis task. Since one master can serve several workers, having several workers that correspond to multiple sentiment analysis tasks can add another level of parallelism, i.e. a hybrid parallelism approach with both task parallelism along with data parallelism. This is however out of the scope of this thesis work.

```
StormTopology stormTopology = StormSamoaU-
tils.argsToTopology(args);
String topologyName = stormTopology.getTopologyName();

Config conf = new Config();
conf.setMaxTaskParallelism(numWorker);

backtype.storm.LocalCluster cluster = new back-
type.storm.LocalCluster();
cluster.submitTopology(topologyName, conf,
stormTopo.getStormBuilder().createTopology());

backtype.storm.utils.Utils.sleep(30000);

cluster.killTopology(topologyName);
cluster.shutdown();
```

FIGURE 16. Master nodes submits the topology and fires up worker nodes

3.1.2 Worker

As you can see in Figure 17, the worker node creates and executes the real-time sentiment analysis task and sets some option in the task, the limit for maximum number of instances is set to 10^{10} instances. Vertical Hoeffding Tree is stated as the classifier with 4 parallel local statistic processors. It should be mentioned that parallelism level is set by trial and error on each cluster. To find the optimum number of parallelism, one can increase the level up to a point that no significant different in performance measures are seen in the learning algorithm and halt this process with the least number achieved so that the rest of runtime memory can be used in other parts of the tasks such as adaptive sliding windows, frequent item miners or real-time labeling of data. At last, Sentinel's TwitterStreamInstance class as set as the source of data. We will explain Sentinels internal in the section 3.2.

```
RealTimeSentimentAnalysis rTimeSentiment = new RealTimeSenti-
mentAnalysis();

rTimeSentiment.setFactory(new SimpleComponentFactory());

rTimeSentiment.instanceLimitOption.setValue(10000000000);
rTimeSentiment.sampleFrequencyOption.setValue(5);
rTimeSenti-
ment.learnerOption.setValueViaCLIString("classifiers.trees.Ve
rticalHoeffdingTree -p 4");
rTimeSenti-
ment.streamTrainOption.setValueViaCLIString(TwitterStreamInst
ance.class.getName());

rTimeSentiment.init();
```

FIG-
URE 17. Worker node executes the real-time sentiment analysis task

3.1.3 RSentimentTask (real-time sentiment analysis task)

`RSentimentTask` is where Sentinel, SAMOA's Vertical Hoeffding Tree, query monitoring and evaluator process are connected. Figure 18 shows a diagram of all components involved in the task.

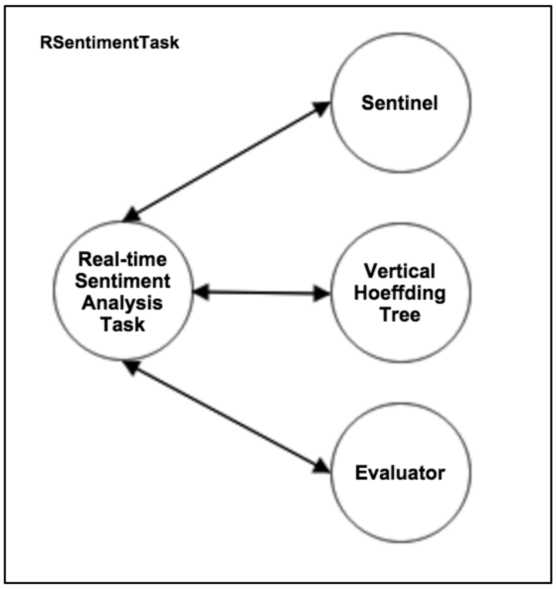
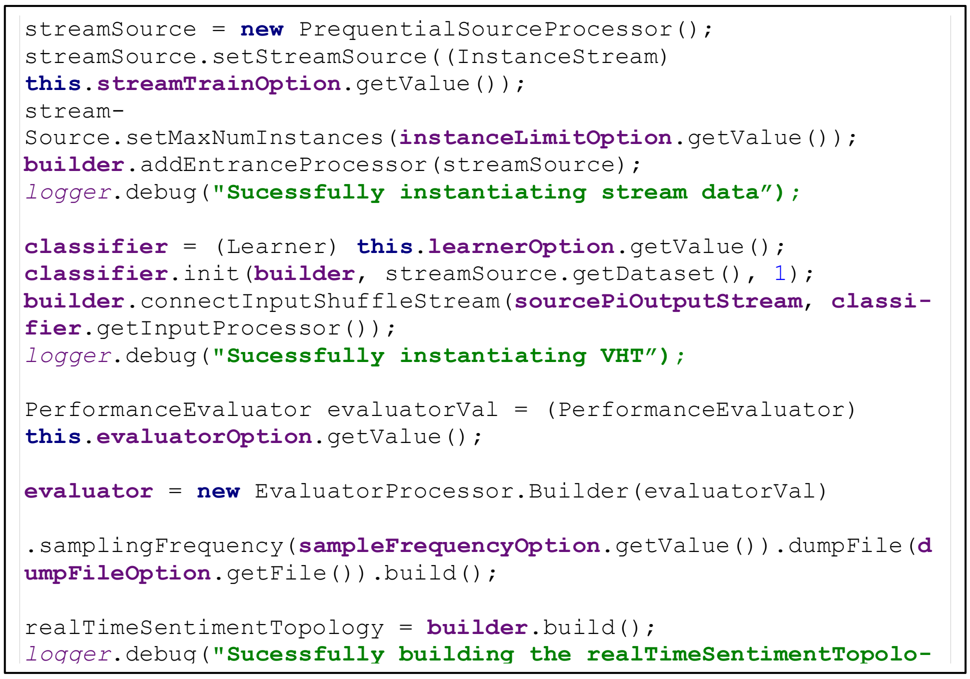


FIGURE 18. `RSentimentTask` interacts with Sentinel, VHT and Evaluator

The `RSentimentTask` starts off by setting Sentinel, as the source of Stream data, meaning that all content events will be read from Sentinel's readers. After that, VHT is initialized. Here, we feed data that comes from Sentinel to VHT. It should be mentioned that we set *shuffle* grouping as VHT grouping mechanism. This is because at this level we do not sort instances based on any identifier, therefore we cannot follow *key* groupings. Selecting *all* mechanism exhausts extra limits to working memory due to having replicated instances. Since we will be using adaptive bagging approach, we prefer to keep as much as memory possible at this stage and therefore all grouping is not followed neither. After that, we instantiate our evaluator that keeps track of measures that comes from VHT. Finally, we execute the topology.



```
streamSource = new PrequentialSourceProcessor();
streamSource.setStreamSource((InstanceStream)
this.streamTrainOption.getValue());
stream-
Source.setMaxNumInstances(instanceLimitOption.getValue());
builder.addEntranceProcessor(streamSource);
logger.debug("Sucessfully instantiating stream data”);

classifier = (Learner) this.learnerOption.getValue();
classifier.init(builder, streamSource.getDataset(), 1);
builder.connectInputShuffleStream(sourcePiOutputStream, classi-
fier.getInputProcessor());
logger.debug("Sucessfully instantiating VHT”);

PerformanceEvaluator evaluatorVal = (PerformanceEvaluator)
this.evaluatorOption.getValue();

evaluator = new EvaluatorProcessor.Builder(evaluatorVal)

.samplingFrequency(sampleFrequencyOption.getValue()).dumpFile(d
umpFileOption.getFile()).build();

realTimeSentimentTopology = builder.build();
logger.debug("Sucessfully building the realTimeSentimentTopolo-
```

FIGURE 19. `RSentimentTask` logic

3.1.3.1 Unbalanced Data Case

A common problem in classification of unbalanced data streams such as Twitter's public stream is that classifiers have high accuracy, close to 90% due to the fact that a large portion of instances fall into one of the classification classes. While working with sample datasets such as Go et al. (2009) data, this feature is removed because the sample data set is balanced, i.e. it has same number of instances for each category of sentiment.

According to, Bifet and Frank (2010), in classification of unbalanced data common accuracy measures or estimations that are used in small datasets such as cross-validation fail to provide an accurate measure in accuracy. The authors have proven that selection of Kappa as measure of accuracy can help neglect the mentioned error. In addition to that, the authors advised to use a method called "Prequental Evaluation" in which each instance is used to test the model and *then* to train the model. This approach is sometimes referred to as *test-then-train*. We follow Bifet and Frank (2010) approach in using Kappa and Prequential Evaluation in our real-time sentiment analysis task.

3.2 Sentinel

Sentinel is responsible for stream reading, pre-processing, query response, feature selection, and frequent item mining specifically for real-time sentiment analysis of Twitter public stream API. We have summarized important components of Sentinel into 5 main components. As with the theme of this chapter, this is a bottom-up procedure. We start by `TwitterStreamInstace` that acts as the connector of all the rest of components and we provide details of each of the components in subsequent sections. You can see the overall picture in Figure 20.

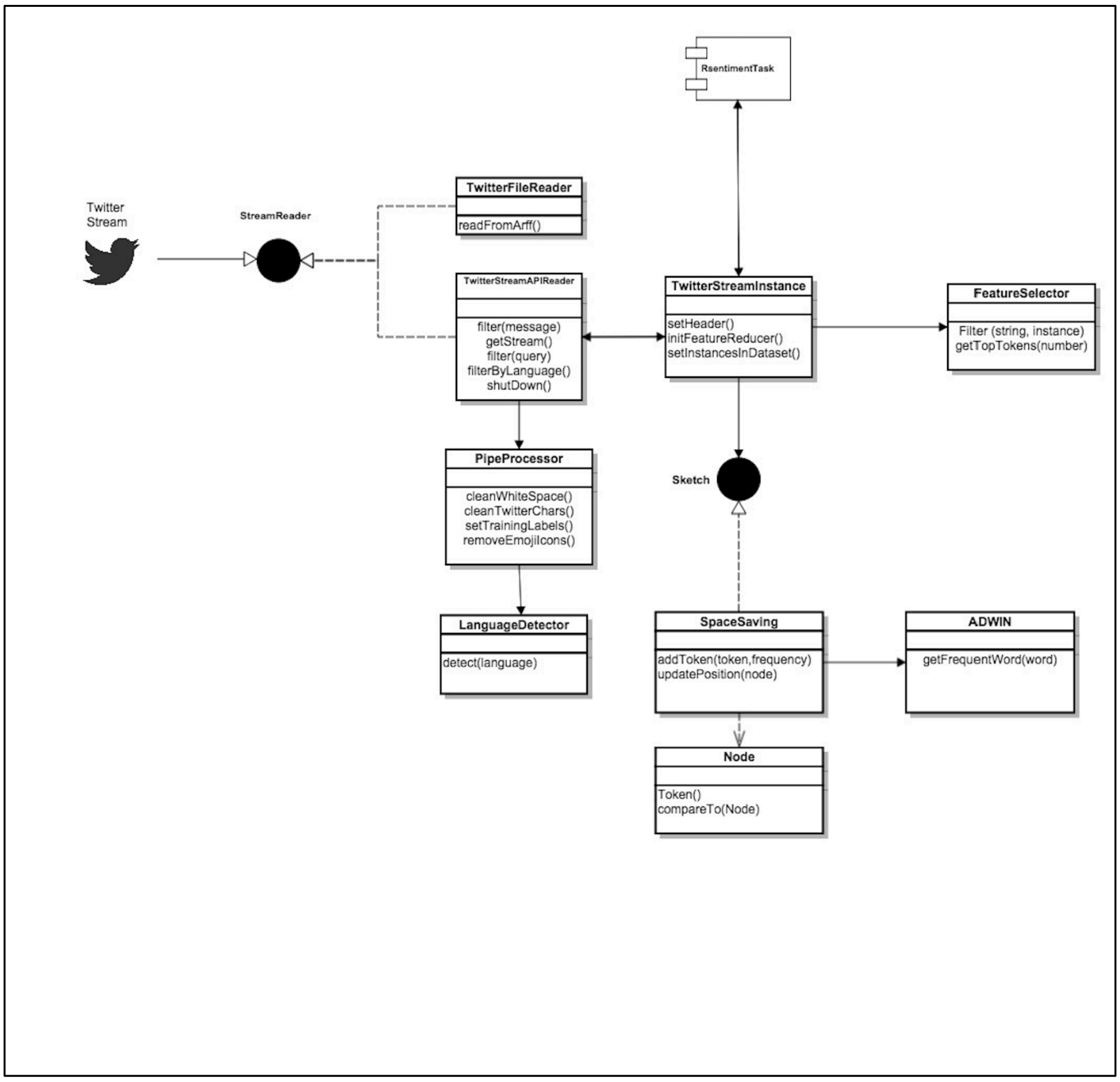


FIGURE 20. Sentinel Class Diagram

3.2.1 TwitterStreamAPIReader

`TwitterStreamAPIReader` (Figure 21) reads the stream directly from Twitter Public Stream[27]. In `TwitterStreamAPIReader`, we use the JSON data. According

[27] https://dev.twitter.com/streaming/public

to Nurseito et al. (2009), JSON encoded objects are transmitted faster than XML objects in Java applications, while using more CPU resources. In addition, both JSON and XML consume the same memory. Instances are read from the random sample of user statuses that Twitter API represents[28]. Using the sample has the benefit that in case several API consumers are connected with same API key, they will receive the same data. This is important because the reading process is a multi-thread process in Sentinel's stream reader that with this approach, synchronization of the instances in the synopsis will be handled. `TwitterStreamAPIReader` keeps track of arrival rate of incoming instances based on how fast they arrived after their last previous instance. We will use this as a metric for our performance analysis in our case study.

`TwitterStreamAPIReader` keeps a synopsis in-memory data structure that can remove or add the incoming instances to it with the maximum threshold of the instances that is set in the worker node and passed to it. Synopsis is implemented as a hash table that keeps IDs of tweets so that it can follow *O(1)* insertion and delete time complexity. First, Stream reader receives the query and performs a primitive filtering based on the existence of the query token in the tweet instance. After the query value is set and checked, the reader captures it. It then uses the open source language detect library called `LangDetect`[29] to set the language of tweet onto incoming instances. The `LangDetect` library uses a Naïve Bayesian classifier that ensures *over* 99% precision in its detection (See the library homepage form ore details). If an instance cannot guarantee to be of English text for 100% probability, it gets discarded. When it is ensured that the text is in English, `TwitterStreamAPIReader` sends the instance to `PipeProcessor` component that performs some text processing on the tweet status text and once it gets the normalized instance back, it keeps them. Whenever `TwitterStreamInstance` asks for new instance(s), Stream reader will respond with the instances it has stored in its synopsis hash map.

3.2.2 PipeProcessor

Tweet texts in stream instances can include information that is excessive to our process. These are Twitter hyperlinks and specific characters such as @, RT, MT that are discussed in details in section 4.1.

Similar to suggestion that was proposed in Go et al. (2009) to shrink the feature space, `PipeProcessor` sets the constant token *USER* for the combination of @ symbol and the user ID. In addition, it replaces hyperlinks within the tweet message by the token URL. Data pipeline keeps the history of repeated letters in distinct forms, meaning that

[28] https://dev.twitter.com/streaming/reference/get/statuses/sample
[29] https://code.google.com/p/language-detection/

the word “awaaaaaaay” gets converted to “awaay” to distinguish it from “away” to keep a more precise feature space that results in more accurate query responses.

```java
public void initStream()
{
    setUpStreamListener();
    twitterStream.sample();
}

public void filter(String[] query) {
    twitterStream.filter(getFilterQuery(query));
}

public FilterQuery getFilterQuery(String[] trackAll) {
    FilterQuery filterQuery = new FilterQuery();
    filterQuery.track(trackAll);

    return filterQuery;
}

public void add(Status status) {
    if (synopsis.size() < sentinel.maxInstances) {
        insert(status);
    } else {
        removeOldInstances();
    }
}

public void insert(Status status) {
    Tweet tweet = soap.processTweets(status.getText(), language);
    String tweetMessage = tweet.getCleanedMessage();

    if (tweetMessage != null && !tweetMessage.trim().equals("")) {
        synchronized (listOfTweets) {
            synopsis.add(tweet);
            sizeTweetList++;
        }
    }
}

@Override
public void setLanguage(String language)
{
    this.language = language;
}

@Override
public String getAndRemove(int position)
{
    // Get the instance and remove it afterwards

public void shutdown() {
    this.twitterStream.shutdown();
}

private void setUpStreamListener() {
    // multi-thread connection to twitter API
}
```

FIGURE 21. `TwitterStreamAPIReader` logic

`PipeProcessor` ensures that such texts are removed from the instance status text and instance is kept as compact as possible, without losing the context. One of core features that `PipeProcessor` does is that it labels the instance data for the training phase of our learning algorithms. We follow Go et al (2009)'s approach however we perform such task in real-time: while processing the tweet and based on the emoji icon used (sad or happy), we add an extra attribute to the tweet object called emotion type and after that we remove the emoji icons. It should be emphasized that this attribute is only used in training phase. This is an affordable approach due to the fact that processing time per instance is extremely limited. Other more sophisticated labeling approaches that take context into the approach are more expensive in computation and therefore are excluded in this study.

```java
public Tweet processTweets(String tweetMessage, String langFilter) {
    tweet = null;
    tweet = new Tweet();
    tweet.setOriginalMessage(tweetMessage);
    message = tweetMessage;

    cleanWhiteSpace();
    removeEmotionsFromMessage();
    takeOutTwitterSpecificCharacters();
    setFilteredMessage();

    return tweet;
}
```

FIGURE 22. `PipeProcessor` logic

3.2.3 FeatureSelector

According to Kogan (2014), one of the challenges in real-time stream mining is the task of feature selection in distributed environment. Kogan (2014) stated that feature selection requires a very high communication overhead. As Bifet et al (2011) suggested, we follow a straightforward feature reducer algorithm, namely term frequency-invert document frequency (TF-IDF)[30].

In this component, data instance are treated in a bag-of-words model. First, based on their frequency, they are presented in a vector of words. After that, its corresponding *tf-idf* values are calculated for each word or token.

[30] See appendix section 5.3 for definition of TF-IDF

Feature reducer has a `SpaceSavingADWIN` instance that is used for responding to the top-k queries of our feature space. We provide more details on `SpaceSavingADWIN` component in the next section.

3.2.4 SpaceSavingADWIN

In this thesis work, based on the extraordinary precision and recall results of Space Saving in top-k queries compared to different count-based algorithms in Liu et al. (2011), we propose the usage of *Space Saving* algorithm (Figure 23).

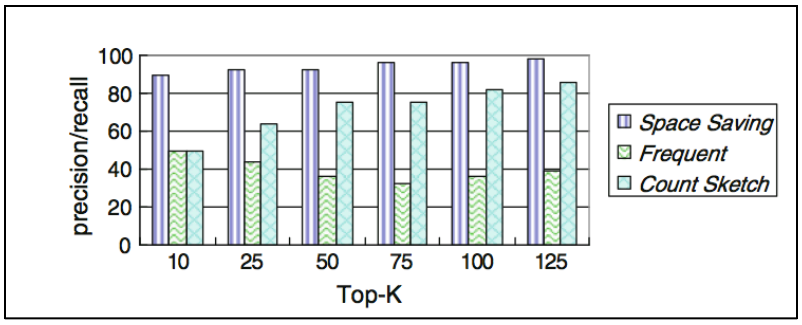


FIGURE 23. Recall/Precision for top-k items with different Ks (Liu et. al: 2011)

As Cormode and Hadjieleftheriou (2008) and Bifet and Frank (2010) showed, Space-Saving algorithm cannot adapt to changes. If a sudden change happens in data, the frequent items will become infrequent, since the algorithm removes the instances with lower frequencies. In this study, we follow Bifet and Frank (2010) and Bifet et al (2011) suggestion of using ADWIN's to make space saving adaptable to change efficiently.

`SpaceSavingADWIN` (Figure 25) improves the performance by keeping only the most relevant features within a document into our windowing model. As mentioned above, `SpaceSavingADWIN` is an adaptive approach keeps relevant instances into its synopsis. `SpaceSavingADWIN` has a synopsis hash map that keeps tokens as the key and a Node object as its value. Node object comprise of corresponding *token, index* and *frequency* of the token. `SpaceSavingADWIN` fills up its hash map with most popular tokens and drop the old ones if it exceeds the capacity of its hash map, which is default to 10000. `SpaceSavingADWIN` also uses an ADWIN frequency counter to be able adapt to new changes and handle concept drift. See Bifet and Frank (2010) for more details. You can see a summary of `SpaceSavingADWIN` logic in Figure 23.

3.2.5 TwitterStreamInstance

`TwitterStreamInstance` (Figure 25) is the interface component between Sentinel and Real-time Sentiment Analysis task. It also acts as a controller that glues different parts of Sentinel together. It has several key functionalities. It checks for new incoming instances inside reader's synopsis on a non-stop loop and receives them instance-by-instance. After that it creates the stream data object that is digestible for VHT along with its header object and context model information. `TwitterStreamInstance` instantiates feature selector component and can decide to work with different feature selection mechanism[31]. It also keeps the `SpaceSavingADWIN` component to keep the sliding window and being able to send the query and return the response back. It also takes care of receiving query from `RSentimentTask` component and passing it on to the reader to use it as a keyword filter. It should be noted that `TwitterStreamIn-stance` is responsible to make sure that during the training phase, model can *only* make use of our `emotionType` feature that was added in `PipeProcessor`. If this is not done, the classifier mode will over-fit the stream.

3.3 Vertical Hoeffding Tree

SAMOA has a VHT algorithm component that performs the learning algorithms on instances that are made by `TwitterStreamInstance`. Since data instances are processed to adapt to VHT's data type, VHT just performs online learning on top of instances that are received from Sentinel. In VHT component, we also make use of its adaptive bagging version as well.

3.4 Hoeffding Tree

[31] In our study, we only use it with TF-IDF appraoch however it can work with n-gram models as well.

For our case study, in order to compare VHT with its serial version, we use MOA's Hoeffding Tree algorithm component and we connect it through SAMOA-to-MOA extension library[32].

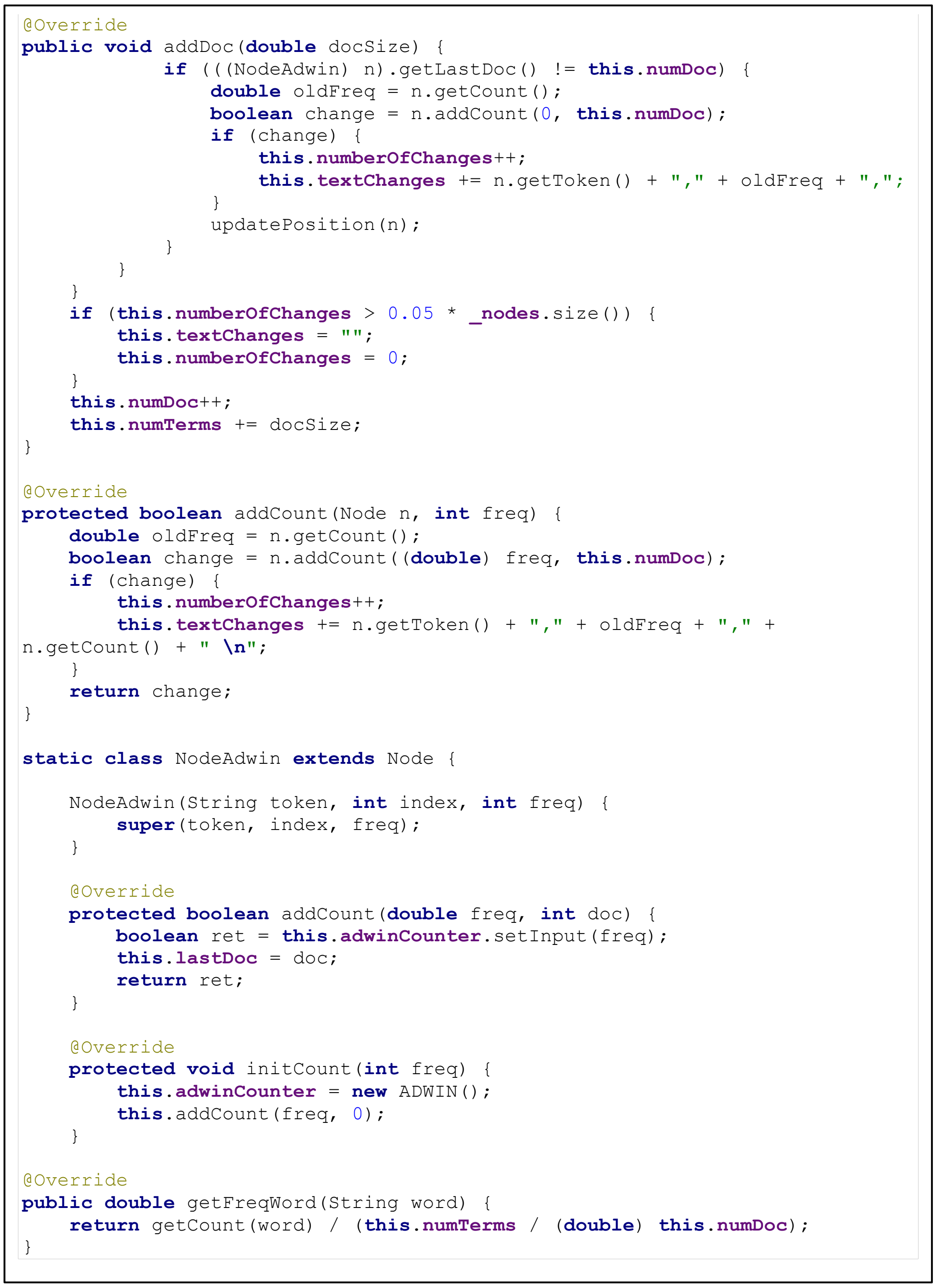
3.5 Evaluator

Our evaluator component takes care of performing measures such as processing time per instance, Kappa and Query Response Time during the learning process. The measures are updated on every *seed* number of instances.

```
  private void setStreamHeader()
  {
     ArrayList<Attribute> wekaAtt = new ArrayList<Attribute>();
     wekaAtt.add(classAtt);

     this.instances = new Instances(getCLICreationString(InstanceStream.class), wekaAtt, 0);
     this.instances.setClassIndex(0);
  }

  private void initFilterTfIdf()
  {
     if (filterTfIdf == null)
     {
        SpaceSavingADWIN sketch = new SpaceSavingADWIN();
        sketch.prepareForUse();

        filterTfIdf = new FilterTfIdf(sketch);
     }
  }

  @Override
  public InstanceExample nextInstance()
  {
     if (this.lastInstanceRead == null) {
        getNextInstance();
     }

    InstanceExample prevInstance = this.lastInstanceRead;

   if (this.twitterStreamReaderSynopsis.size().newData)
     {
        m = this.twitterStreamReader.getAndRemove(0);
        inst = this.filterTfIdf.filter(m, this.getHeader());
        if (this.writer != null)
        {
           try
           {
              writer.write(m);
              writer.write("\n");
           }
           catch (Exception ex)
           {
              throw new RuntimeException(
                 "Failed writing to file ", ex);
           }
        }
     }
     return result;
  }

  public String[] getSketch(int n)
  {
     return filterTfIdf.getTopTokens(n);
  }
}
```

[32] Availab

FIGURE 25. `TwitterStreamInstance` logic



```java
@Override
public void addDoc(double docSize) {
            if (((NodeAdwin) n).getLastDoc() != this.numDoc) {
                double oldFreq = n.getCount();
                boolean change = n.addCount(0, this.numDoc);
                if (change) {
                    this.numberOfChanges++;
                    this.textChanges += n.getToken() + "," + oldFreq + ",";
                }
                updatePosition(n);
            }
        }
    }
    if (this.numberOfChanges > 0.05 * _nodes.size()) {
        this.textChanges = "";
        this.numberOfChanges = 0;
    }
    this.numDoc++;
    this.numTerms += docSize;
}

@Override
protected boolean addCount(Node n, int freq) {
    double oldFreq = n.getCount();
    boolean change = n.addCount((double) freq, this.numDoc);
    if (change) {
        this.numberOfChanges++;
        this.textChanges += n.getToken() + "," + oldFreq + "," +
n.getCount() + " \n";
    }
    return change;
}

static class NodeAdwin extends Node {

    NodeAdwin(String token, int index, int freq) {
        super(token, index, freq);
    }

    @Override
    protected boolean addCount(double freq, int doc) {
        boolean ret = this.adwinCounter.setInput(freq);
        this.lastDoc = doc;
        return ret;
    }

    @Override
    protected void initCount(int freq) {
        this.adwinCounter = new ADWIN();
        this.addCount(freq, 0);
    }

@Override
public double getFreqWord(String word) {
    return getCount(word) / (this.numTerms / (double) this.numDoc);
}
```

FIGURE 24. `SpaceSavingADWIN` logic

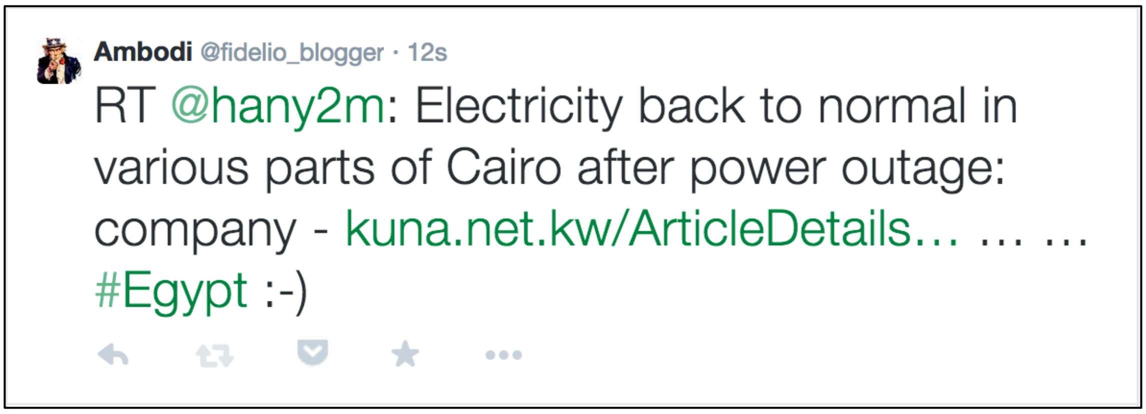
4 CASE STUDY: TWITTER PUBLIC STREAM API

In this chapter, we show application of Sentinel on Twitter Public Stream API. First, we discuss some basic conventions about Twitter and its APIs. In section 4.3, we present the results. We followed the pattern of discussing result as in Bifet et al (2011), meaning that first we show popular feature keywords in our feature space and after that we discuss sentiment positive polarity result. We conclude that section with discussing performance, accuracy and query response time. In section 4.5, application areas that our software prototype might contribute in are mentioned and lastly further research is suggested.

4.1 About Twitter

4.1.1 Conventions

In Twitter terminology, messages contain 140 characters and describe the "status" of a user. In Figure 26, the user by the name of *fidelio_blogger* has re-tweeted a message, i.e. posted a copy of another user's message, by the name of *hany2m*. The token RT is an acronym for Re-Tweet which shows that it is not an original message and source of the message is the user mentioned after the @ token. The tweet message itself includes a text about a piece of news that happened in Egypt, which is followed by a hyperlink to the news page and a hashtag (#) with the text Egypt. The usage of hashtag can bring extra context or can make the text searchable by other users that search news about Egypt. Companies and News agencies use such hashtag to create marketing campaigns, new and media to create live debates and opinion gatherings and etc.



Ambodi @fidelio_blogger · 12s

RT @hany2m: Electricity back to normal in various parts of Cairo after power outage: company - kuna.net.kw/ArticleDetails… … … #Egypt :-)

FIGURE 26. A Tweet Message

4.1.2 Twitter Application Programming Interfaces

Twitter currently provides a Stream Application Programming Interface (API) and two HTTP based REST APIs. Twitter public stream API[33] is our source of social stream data in this case study that is a stream data of type non-stationary. As of September 2014, Twitter receives around 500 million tweets per day. Meaning that around 5800 tweets are processed in every second. As the second quarter of 2014, Twitter has 271 million registered users, and 271 million unique monthly visitors[34]. With the huge volume of data and fast speed of data generation, Twitter public stream API is one of the best choices of publicly available real-time stream data within social media websites. To be able to use the API, one needs a verified Twitter. API Data can be retrieved as XML or JSON format.

4.2 Case Study question

On September 2nd, 2014, the so-called Islamic State of Iraq and Levant (ISIS) released a YouTube video of a beheading of a man whom they identified as the American journalist Steven Sotloff. This created a debate over Twitter. We wanted to show the real-time positive sentiment of Twitter users who mentioned the happening. In addition, to present some measures of performance, accuracy and response time of our software prototype.

[33] https://dev.twitter.com/docs/streaming-apis/streams/public
[34] https://about.twitter.com/company

4.3 Result and Discussions

For the experiment, we ran sentinel on a digital ocean[35] cluster with 14GB of RAM and quad-core Intel 2.90GHZ CPU. Sentinel captured around 1017344 twitter instances with English text from August 31st to September 4th containing the keyword ISIS. The dataset size including the metadata that is needed for learning algorithms in SAMO is approximate 1.4GB compressed, if it was stored in disk. Arrival time rates that were captured during the experiment are represented in Table 1 and Figure 27.

Time Period in Seconds	**Percentage of Total Instances**
0 to 1	4.76
1 to 2	5.31
2 to 5	4.81
5 to 10	13.53
10 to 60	9.15
60 to 3600	25.19
> 3600	37.25

Table 1. Arrival rates between two consecutive stream instances

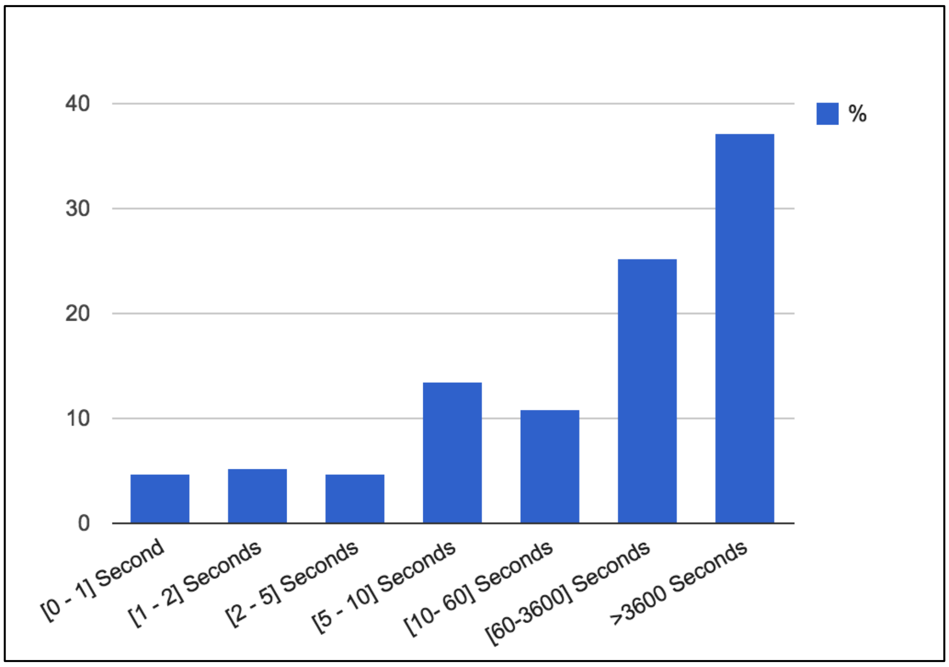


FIGURE 27. Arrival rates between two consecutive stream instances

[35] https://www.digitalocean.com/

4.3.1 Top-k features

Using `SpaceSavingADWIN` component, Table 2 shows the features are the result of the top-k query (k=7) during our experiment:

Feature
isis
#terrorism
beheading
united states
iraq
massacre
syria

Table 2. Top-k query result

4.3.2 Sentiment Analysis

We utilized the VHT adaptive bagging with 4 parallel processors for sentiment analysis of tweets. In Figure 27, a sudden decrease in positive sentiments can be seen on September the 2nd, which was the day that the YouTube video was published on YouTube. This shows that our approach captured some aspects of the event successfully by revealing common disappoint or negative sentiment toward the topic.

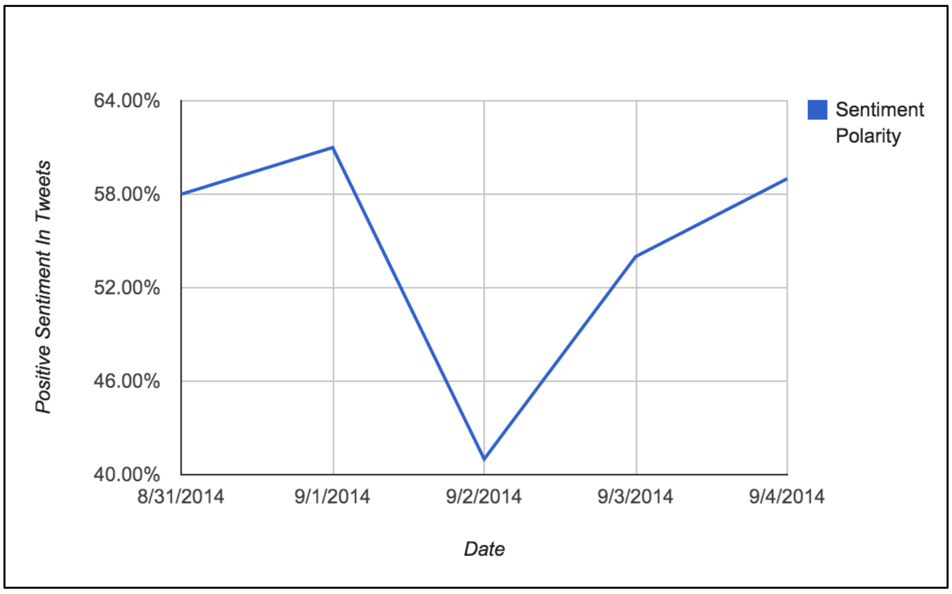


FIGURE 28. Positive Sentiment detection in English tweets with keyword “ISIS”

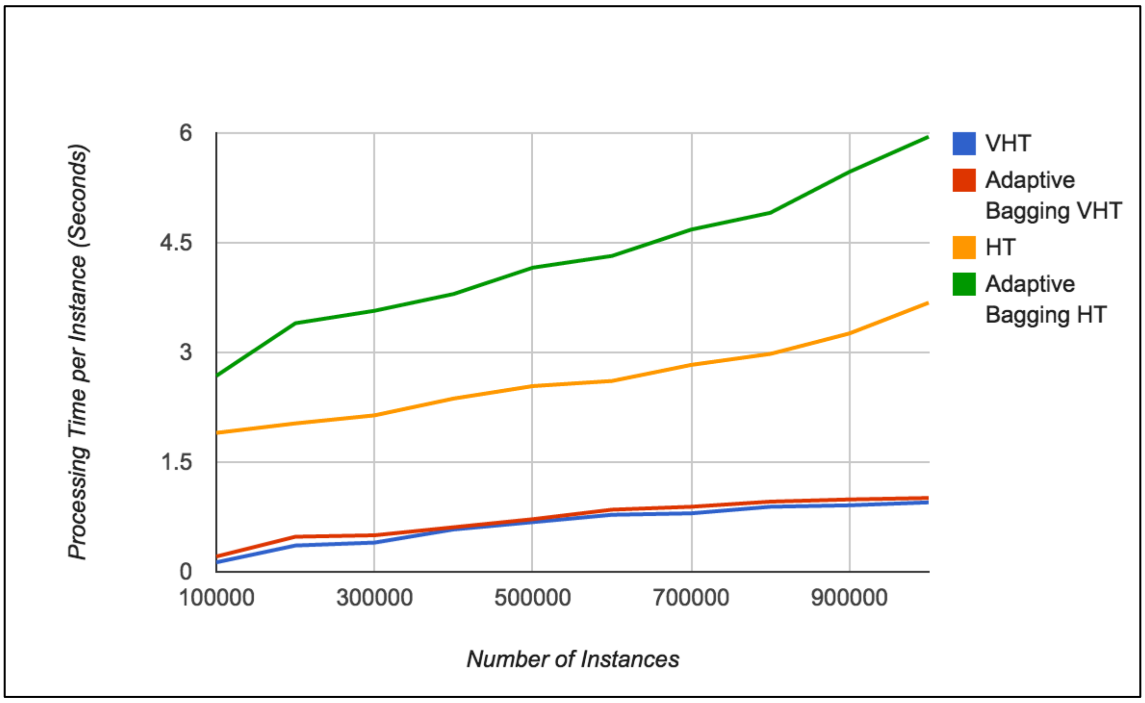


FIGURE 29. Processing per instance measures in seconds

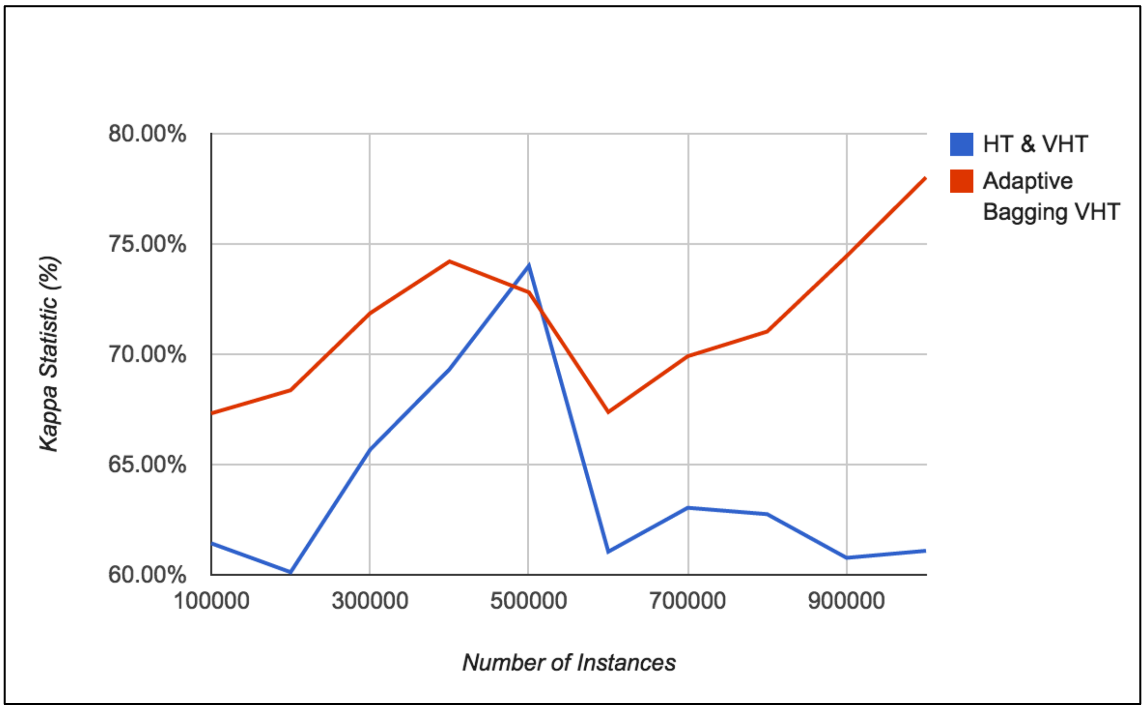


FIGURE 30. Kappa Statistic Measures

4.3.3 Performance Measures

In figure 28 and Table 3, we compared the *performance* of VHT with serial Hoeffding Tree algorithms with and without their adaptive bagging versions. Since all classifiers are learning from the same synopsis data, we used per-instance processing time as our performance measures. In the comparison, VHT showed higher performance compared to its serial counterpart. The speedup value has not improved linearly, meaning that VHT is not performing 4 times better than Hoeffding Tree[36], however, VHT is performing 24% better (in average) than its serial version. In case we use adaptive bagging VHT, 3% of the performance in VHT is reduced. This is due to the fact that VHT has numerous mechanisms such as "node-limiting" and "poor-attribute" removal that makes VHT use less memory and in more efficient ways. See Murdopo (2013) for more details. However, this introduces higher costs in serial Hoeffding Tree. Usage of adaptive bagging on Hoeffding Tree will increase the processing time per instance to 6 in its maximum. Based on Figure 27, this makes adaptive bagging version of Hoeffding Tree to have more than 14.88% data loss, which we believe to be a low performance for real-time analytics.

Stream Instances	**VHT**	**Adaptive Bagging VHT**	**HT**	**Adaptive Bagging HT**
100000	0.13	0.21	1.9	2.68
200000	0.36	0.48	2.03	3.4
300000	0.4	0.5	2.14	3.57
400000	0.58	0.61	2.37	3.8
500000	0.68	0.72	2.54	4.16
600000	0.78	0.85	2.61	4.32
700000	0.8	0.89	2.83	4.68
800000	0.89	0.96	2.98	4.91
900000	0.91	0.99	3.26	5.47
1000000	0.95	1.01	3.68	5.95

Table 3. Processing Time per Instance

4.3.4 Accuracy Measure

In terms of *accuracy* measures (see Figure 30 and Table 4), adaptive bagging VHT shows higher accuracy results compared to its serial version. It should be mentioned that VHT and Hoeffding Tree have the same accuracy measures, since VHT has not made any improvements on accuracy measures in Hoeffding Tree (see Murdopo (2013) for more details) and this is apparent in our experiment as well. In general, you can see that all classifiers are showing less accuracy compared to most classifiers that follow

[36] It should be remembered that we utilize VHT with parallelism set to 4.

batch-learning model. We present three main reasons for general low accuracy results in our experiment:

- Batch-learning algorithms can high accuracy measures even higher than 90%, however online learning algorithms are less accurate in general (Le Cun and Botto, 2004).
- Another reason for having lower accuracy measures in both algorithms in general is because of our approach in labeling the instances. Labeling is a costly operation to perform in real-time and therefore to reduce the computational overhead, we used a rather simple rule-based labeling approach. This of course can reflect in our accuracy measures as well. This is usually the case with online learning that the labeling procedure cannot be as sufficient and therefore computationally complex as in batch learning. This leads to having a less optimally trained model. Our experience is no exception is this case.

- Lastly, using Kappa as our accuracy measure will result with lower accuracy results in unbalanced data. As explained earlier, Kappa eliminates the error in over-accurate learners with unbalances classes and therefore this can result in lower accuracy measures for all classifiers.

Stream Instances	**HT & VHT**	**Adaptive Bagging VHT & Adaptive Bagging HT**
100000	61.43%	67.32%
200000	60.12%	68.37%
300000	65.67%	71.86%
400000	69.32%	74.21%
500000	74.01%	72.81%
600000	61.05%	67.38%
700000	63.04%	69.91%
800000	62.75%	71.03%
900000	60.77%	74.45%
1000000	61.09%	78.02%

Table 4. Kappa Accuracy Measures

As you can see in Figure 30 and Table 4, a sudden decrease of accuracy for both learners is visible when approximately 600000 instances are processed. This is due to noise that is caused by sudden computer bot attacks. Usually computer bots sends batch of tweets that advertises pornography, illegal download websites to perform different attacks on the Twitter Public stream API. Although the keyword filter and language detection rule in our prototype should have reduced the side effects of the attack, however this was not the case. The noisy tweets are meaningless and they either contain tweets with few tokens or tweets with high number of meaningless tokens along with URLs. This confuses the learners, since they train and test on tweets that they are not used classify in their model. Therefore it can decrease the accuracy of the learning algorithm in a rather brief period of time, however after that, both algorithms showed increased accuracy in classification.

4.3.5 Query Response Time Measure

In Figure 31, we can see the Query Response Time (QRT) for both top-k items and Kappa measures. It is apparent that once our synopsis contains more data, the response time will increase, however the change is not that significant and falls into an acceptable range for real-time analytics.

4.3.6 Conclusion

This experiment showed the high potentials of large-scale learning in distributed environment in real-time sentiment analysis. Shifting from serial Hoeffding Tree to Vertical Hoeffding Tree reduced the number of data loss by *more than* 47.26%. Not only the performance was improved, but also efficient usage of memory in the parallel version brought the opportunity to use that extra memory for improving accuracy measures of Hoeffding Tree (or equivalently its parallel version) by adaptive bagging approaches with less cost.

As mentioned in our objectives, our key objective was to have a better combination of accuracy and performance, rather than to focus solely on performance. Based on the results from both accuracy and performance measures, we suggest the usage of adaptive bagging VHT for real-time sentiment analysis in our study. Usage of adaptive bagging VHT introduced up to 74.01% of improvements in accuracy measures (63.93% in average). On the other hand, Adaptive Bagging VHT only lost less than 4.76% of instances in our real-time data. It is apparent that the same setting for the serial version does not hold. Adaptive bagging for Hoeffding Tree showed poor performance result that can have up to 14.88% data loss. We believe that based this result, we do not recommend usage of adaptive bagging in serial Hoeffding tree.

4.4 Application areas

Generally, real-time sentiment analysis can be applied in different organizations, products and brands and therefore the application area is vast. The knowledge of real-time sentiment analysis of social streams helps to understand what social media users think or express “right now”. Applications of real-time sentiment analysis of Twitter stream have brought a lot of opportunities in marketing, news monitoring, customer satisfaction and so on. We showed an example on how news can be monitored in a social media like Twitter and detect sudden changes in real-time and having the ability to perform sentiment analysis in real-time. An example in marketing is when a marketing campaign is launched and analysts can track customer's response to the campaign and feedback in real-time. Another example is to evaluate customer satisfaction when a product is released or a new decision is made in a company and see immediate reactions of customers or fans of the company. It also applies in the prevention of disasters before they turn into crisis such as natural disasters or business disaster. As an example, business crisis such as 2010 Toyota's crisis can become visible and prevented in real-time. Lastly, real-time sentiment analysis can analyze sentiments of users in live debates. Recently a large portion of Twitter users participates in debates, which are mainly organized from TV, radio shows and news agencies. Users send a tweet about their opinion following a certain hashtag (#); therefore in this manner Twitter becomes an open polling platform.

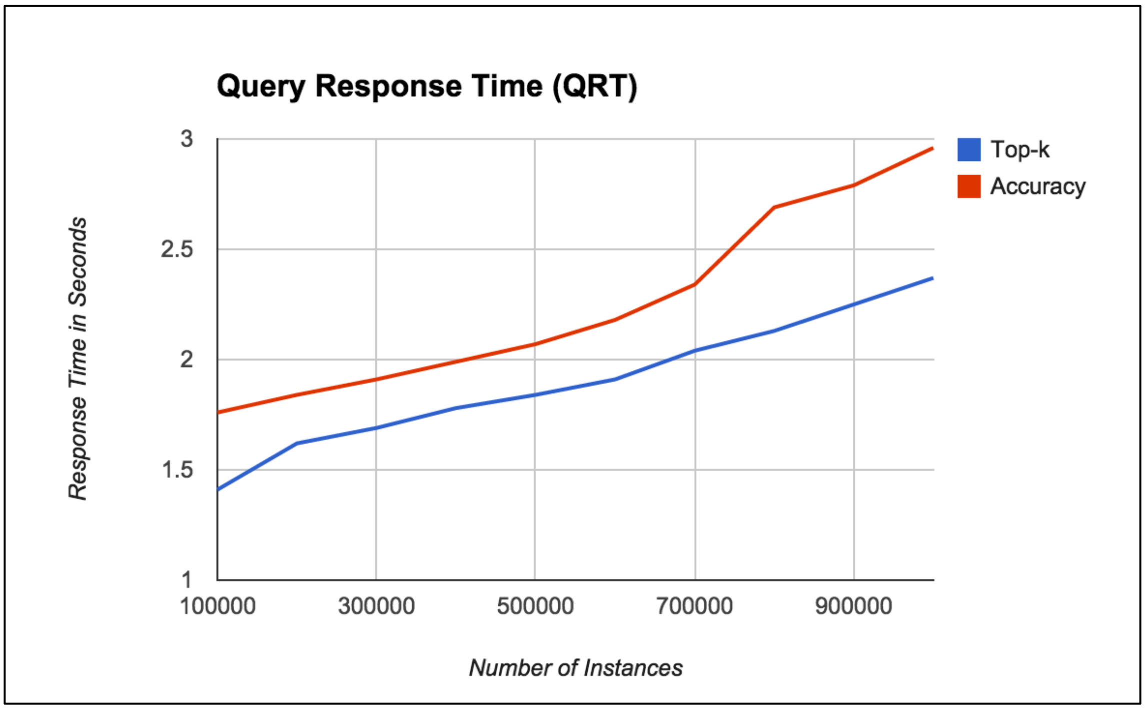


FIGURE 31. Query Response Time

4.5 Further research

In this study we mainly emphasized on distributed supervised learning on social streams in real-time, however there are more interesting sources of data stream that are not yet evaluated. Sentinel can adapt to more stream sources and add functionalities to address the needs for those sources and bring new sentiment results from them. One further research topic can be to enable generic classification task in Sentinel, which are not necessarily related to sentiment analysis. Sentinel can be also developed to be used with more distributed classifiers in future. The real-time clustering of social stream is another young field that Sentinel can be developed in. Lastly, Sentinel can also benefit from having its own real-time visualization component in future to capture reflection of data customized to data stream mining needs.

SUMMARY

Sentiment analysis on Twitter stream has been a topic of research for more than a decade. Several *non-commercial* libraries and software were developed to perform sentiment analysis, however none of them performed the analytics in real-time for Twitter data. We addressed the problem using large-scale online learning and specifically online parallel decision trees. We used SAMOA's VHT learner with Apache Storm as our Stream Processing Engine. However, utilizing *only* VHT and Apache Storm could not solve the problem at hand. Therefore, we *also* developed an open-source Java library called Sentinel that enables real-time Twitter stream reading, in-memory pre-processing computations and data structures, feature selection, frequent miner algorithms and etc. that completes our architecture. In Chapter 3, we showed the architecture of our solution and its applicability and usefulness was shown in chapter 4. The result of this study is communicated in Rahnama (2014) from chapter 1 to chapter 4.

5 APPENDIX

5.1 Kappa Statistic

According to Carletta (1996), *Kappa* is a measure of agreement between k raters. Each of k rater classifies N items into C categories that are mutually exclusive. Kappa can be formulated as follows:

$$\mathrm{K} = \frac{P_a - P_e}{1 - P_e}$$

in which P_a is the number of time the k raters classify items into the same categories and P_e is the number of times they perform it by pure chance.

5.2 Information gain

According to Han and Kamber (2006, pp. 297), *information gain* is a popular method for attribute selection measures in decision trees. Suppose *T* is the tuples of a set called *S*. Also, assume a random variable with k number of probability functions $p_1, p_2, \dots, p_n$ in which p_i is the probability of a tuple in S belonging to $Class_i$ $(i = 1, \dots, k)$. Information gain can be formulated as:

$$Info(S) = -\sum_{i=1}^{k} p_i \ log_2(p_i)$$

When using Information gain as selection measures, the attribute that has the highest information gain value is attribute that we perform the split on. See Han and Kamber (2006) for more details.

5.3 TF-IDF

According to Rajaraman and Ullman (2012, pp. 8), Term Frequency times Inverse Document Frequency (TF-IDF) assigns a value that represents the weight of a token (word) in a document. The *TF-IDF* of the token i in document j can be formulated as:

$$tf_{ij}.idf_i$$

Suppose token i appears in n_{total} number of N number of documents. Then,

$$idf_i = \log \frac{N}{n_{total}}$$

On the other hand, suppose f_{ij} is the number of occurrences of token i in document j. As a result,

$$tf_{ij} = \frac{f_{ij}}{max_t f_{tj}}$$

Meaning that division on number of all tokens in the document normalizes the token frequency of token i in document j.

REFERENCES

Adamic, L. A., & Huberman, B. A. (2002). Zipf's law and the Internet. Glottometrics, 3(1), 143-150.

Afrati, F. N., Sarma, A. D., Salihoglu, S., & Ullman, J. D. (2012). Vision paper: Towards an understanding of the limits of map-reduce computation. arXiv preprint arXiv:1204.1754.

Aggarwal, C. C., Han, J., Wang, J., & Yu, P. S. (2003). A framework for clustering evolving data streams. In Proceedings of the 29th international conference on Very large data bases-Volume 29 (pp. 81-92). VLDB Endowment.

Aggarwal, C. C., Han, J., Wang, J., & Yu, P. S. (2004). On demand classification of data streams. In Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 503-508). ACM.

Aly, A. M., Sallam, A., Gnanasekaran, B. M., Nguyen-Dinh, L., Aref, W. G., Ouzzani, M., & Ghafoor, A. (2012, April). M3: Stream processing on main-memory mapreduce. In Data Engineering (ICDE), 2012 IEEE 28th International Conference on (pp. 1253-1256). IEEE.

Andrews, A. P., Simon, J., Tian, F., & Zhao, J. (2011). The Toyota crisis: an economic, operational and strategic analysis of the massive recall.Management Research Review, 34(10), 1064-1077.

Arndt, C. (2001). Information Measures: information and its description in science and engineering; with 64 Figures. Springer.

Amado, N., Gama, J., & Silva, F. (2001). Parallel implementation of decision tree learning algorithms. In Progress in Artificial Intelligence (pp. 6-13). Springer Berlin Heidelberg.

Aston, N., Liddle, J., & Hu, W. (2014). Twitter Sentiment in Data Streams with Perceptron. Journal of Computer and Communications, 2014.

Babcock, B., & Olston, C. (2003, June). Distributed top-k monitoring. In Proceedings of the 2003 ACM SIGMOD international conference on Management of data (pp. 28-39). ACM.

Bockermann, C., & Blom, H. (2012). Processing data streams with the rapidminer streams-plugin. In Proceedings of the RapidMiner Community Meeting and Conference.

Barbier, G., & Liu, H. (2011). Data mining in social media. In Social Network Data Analytics (pp. 327-352). Springer US.

Ben-Haim, Y., & Tom-Tov, E. (2010). A streaming parallel decision tree algorithm. The Journal of Machine Learning Research, 11, 849-872.

Bifet, A. (2013). Mining Big Data in Real Time. Informatica (03505596), 37(1).

Bifet, A., & Frank, E. (2010). Sentiment knowledge discovery in twitter streaming data. In Discovery Science (pp. 1-15). Springer Berlin Heidelberg.

Bifet, A., Holmes, G., & Pfahringer, B. (2011). MOA-TweetReader: real-time analysis in twitter streaming data. In Discovery Science (pp. 46-60). Springer Berlin Heidelberg.

Bifet, A., Holmes, G., Kirkby, R., & Pfahringer, B. (2010). Moa: Massive online analysis. The Journal of Machine Learning Research, 11, 1601-1604.

Bifet, A., & Kirkby, R. (2009). Data stream mining a practical approach.

Bifet, A., Holmes, G., Pfahringer, B., & Gavalda, R. (2009). Improving adaptive bagging methods for evolving data streams. In Advances in Machine Learning(pp. 23-37). Springer Berlin Heidelberg.

Bifet, A., & Gavalda, R. (2007). Learning from Time-Changing Data with Adaptive Windowing. In SDM (Vol. 7, p. 2007).

Bishop C. M (2006), Pattern Recognition and Machine Learning, Springer Science+Business Media, , ISBN-10: 0-387-31073-8

Bravo-Marquez, F., Mendoza, M., & Poblete, B. (2013). Combining strengths, emotions and polarities for boosting Twitter sentiment analysis. InProceedings of the Second International Workshop on Issues of Sentiment Discovery and Opinion Mining (p. 2). ACM.

Bekkerman, R., Bilenko, M., & Langford, J. (Eds.). (2011). Scaling up machine learning: Parallel and distributed approaches. Cambridge University Press.

Carletta, J. (1996). Assessing agreement on classification tasks: the kappa statistic. Computational linguistics, 22(2), 249-254.

Chan, P. K., & Stolfo, S. J. (1998). Toward Scalable Learning with Non-Uniform Class and Cost Distributions: A Case Study in Credit Card Fraud Detection. In KDD (Vol. 1998, pp. 164-168).

Charikar M, Chen K, Farach-Colton M (2002) Finding frequent items in data streams. In: Proceedings of 29th international colloquium on automata, languages and programming, pp 693–703

Cohen, L., Avrahami-Bakish, G., Last, M., Kandel, A., & Kipersztok, O. (2008). Real-time data mining of non-stationary data streams from sensor networks.Information Fusion, 9(3), 344-353.

Condie, T., Conway, N., Alvaro, P., Hellerstein, J. M., Gerth, J., Talbot, J., ... & Sears, R. (2010, June). Online aggregation and continuous query support in mapreduce. In Proceedings of the 2010 ACM SIGMOD International Conference on Management of data (pp. 1115-1118). ACM.

Cormode, G., & Muthukrishnan, S. (2005). What's hot and what's not: tracking most frequent items dynamically. ACM Transactions on Database Systems (TODS), 30(1), 249-278.

Cormode, G., & Hadjieleftheriou, M. (2008). Finding frequent items in data streams. Proceedings of the VLDB Endowment, 1(2), 1530-1541.

Chu, C. T., Kim, S. K., Lin, Y. A., Yu, Y., Bradski, G., Ng, A. Y., & Olukotun, K. (2006, December). Map-reduce for machine learning on multicore. In NIPS(Vol. 6, pp. 281-288).

Dai, W., & Ji, W. (2014). A MapReduce Implementation of C4. 5 Decision Tree Algorithm. International Journal of Database Theory & Application, 7(1).

Datar, M., & Motwani, R. (2007). The sliding-window computation model and results. In Data Streams (pp. 149-167). Springer US.

De Francisci Morales, G. (2013). SAMOA: A platform for mining big data streams. In Proceedings of the 22nd international conference on World Wide Web companion (pp. 777-778). International World Wide Web Conferences Steering Committee.

Demaine, E. D., López-Ortiz, A., & Munro, J. I. (2002). Frequency estimation of internet packet streams with limited space. In Algorithms—ESA 2002 (pp. 348-360). Springer Berlin Heidelberg.

Dietterich, T. G. (2002). Machine learning for sequential data: A review. InStructural, syntactic, and statistical pattern recognition (pp. 15-30). Springer Berlin Heidelberg

Dietterich, T. G. (2000). An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. Machine learning, 40(2), 139-157.

Dean, J., & Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. Communications of the ACM, 51(1), 107-113.

Domingos, P., & Hulten, G. (2000). Mining high-speed data streams. InProceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 71-80). ACM.

Easterbrook, S., Singer, J., Storey, M. A., & Damian, D. (2008). Selecting empirical methods for software engineering research. In Guide to advanced empirical software engineering (pp. 285-311). Springer London.

Ediger, D., Jiang, K., Riedy, J., Bader, D. A., Corley, C., Farber, R., & Reynolds, W. N. (2010, September). Massive social network analysis: Mining twitter for social good. In Parallel Processing (ICPP), 2010 39th International Conference on (pp. 583-593). IEEE.

Eekels, J., & Roozenburg, N. F. (1991). A methodological comparison of the structures of scientific research and engineering design: their similarities and differences. Design Studies, 12(4), 197-203.

Estan, C., & Varghese, G. (2002). New directions in traffic measurement and accounting (Vol. 32, No. 4, pp. 323-336). ACM.

Fagnani, F., Fosson, S. M., & Ravazzi, C. (2014). A distributed classification/estimation algorithm for sensor networks. SIAM Journal on Control and Optimization, 52(1), 189-218.

Fischer MJ, Salzberg SL (1982) Finding a majority among N votes: solution to problem, J Algorithm 3(4),376–379

Fuller, R. B., & McHale, J. (1967). World Design Science Decade, 1965-1975. World Resources Inventory.

Gama, J., Sebastião, R., & Rodrigues, P. P. (2009). Issues in evaluation of stream learning algorithms. In Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 329-338). ACM.

Gama, J., Medas, P., & Rocha, R. (2004, March). Forest trees for on-line data. In Proceedings of the 2004 ACM symposium on Applied computing (pp. 632-636). ACM.

Gens, F. (2011). Idc predictions 2012: Competing for 2020.

Ghiassi, M., Skinner, J., & Zimbra, D. (2013). Twitter brand sentiment analysis: A hybrid system using n-gram analysis and dynamic artificial neural network. Expert Systems with Applications: An International Journal, 40(16), 6266-6282.

Giannella, C., Han, J., Pei, J., Yan, X., & Yu, P. S. (2003). Mining frequent patterns in data streams at multiple time granularities. Next generation data mining, 212, 191-212.

Gibbons, P. B., & Matias, Y. (1999). Synopsis data structures for massive data sets. In Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms (pp. 909-910). Society for Industrial and Applied Mathematics.

Gilbert, A. C., Kotidis, Y., Muthukrishnan, S., & Strauss, M. (2001). Quicksand: Quick summary and analysis of network data. Technical Report, available at: citese-er.nj.nec.com/gilbert01quicksand.html

Go, A., Bhayani, R., & Huang, L. (2009). Twitter sentiment classification using distant supervision. CS224N Project Report, Stanford, 1-12.

Gray, I., Chan, Y., Audsley, N. C., & Wellings, A. (2014). Architecture-Awareness for Real-Time Big Data Systems. In Proceedings of the 21st European MPI Users' Group Meeting (p. 151). ACM.

Guerra, P. C., Meira Jr, W., & Cardie, C. (2014). Sentiment analysis on evolving social streams: how self-report imbalances can help. In Proceedings of the 7th ACM international conference on Web search and data mining (pp. 443-452). ACM.

Guha, S., Mishra, N., Motwani, R., & O'Callaghan, L. (2000). Clustering data streams. In Foundations of computer science, 2000. proceedings. 41st annual symposium on (pp. 359-366). IEEE.

Han, J., Kamber, M., (2006), Data Mining Concepts and Techniques, Second Edition, Morgan Kaufman Publishers Elsevier Inc.

Hevner, A., & Chatterjee, S. (2010). Design science research in information systems (pp. 9-22). Springer US

Hoeffding, W. (1963). Probability inequalities for sums of bounded random variables. Journal of the American statistical association, 58(301), 13-30.

Holmes, G., Richard, K., & Pfahringer, B. (2005). Tie-breaking in Hoeffding trees.

Jiang, D., Chen, G., Ooi, B. C., Tan, K. L., & Wu, S. (2014). epiC: an Extensible and Scalable System for Processing Big Data. Proceedings of the VLDB Endowment, 7(7).

Katakis, I., Tsoumakas, G., & Vlahavas, I. (2006). Dynamic feature space and incremental feature selection for the classification of textual data streams.Knowledge Discovery from Data Streams, 107-116.

Kontopoulos, E., Berberidis, C., Dergiades, T., & Bassiliades, N. (2013). Ontology-based sentiment analysis of twitter posts. Expert systems with applications, 40(10), 4065-4074.

Kopetz, H. (2011). Real-time systems: design principles for distributed embedded applications. Springer.

Kogan, J. (2014). Feature Selection Over Distributed Data Streams. In Data Mining for Service (pp. 11-26). Springer Berlin Heidelberg.

Kumar, A., Kantardzic, M., & Madden, S. (2006). Distributed Data Mining

Last, M. (2002). Online classification of nonstationary data streams. Intelligent Data Analysis, 6(2), 129-147.

Laney D (2001), 3-d data management: controlling data volume, velocity and variety. META Group Research Note, available at: http://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf

Le Cun, L. B. Y., & Bottou, L. (2004). Large scale online learning. Advances in neural information processing systems, 16, 217.

Lee, K. H., Lee, Y. J., Choi, H., Chung, Y. D., & Moon, B. (2012). Parallel data processing with MapReduce: a survey. AcM sIGMoD Record, 40(4), 11-20.

Laptev, N., Zeng, K., & Zaniolo, C. (2012). Early accurate results for advanced analytics on mapreduce. Proceedings of the VLDB Endowment, 5(10), 1028-1039.

Liu, H., Lin, Y., & Han, J. (2011). Methods for mining frequent items in data streams: an overview. Knowledge and information systems, 26(1), 1-30.

Manjhi, A., Shkapenyuk, V., Dhamdhere, K., & Olston, C. (2005, April). Finding (recently) frequent items in distributed data streams. In Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on (pp. 767-778). IEEE.

Manku, G. S., & Motwani, R. (2002, August). Approximate frequency counts over data streams. In Proceedings of the 28th international conference on Very Large Data Bases (pp. 346-357). VLDB Endowment.

Mannisto, T. (2013), Lecture notes in "methods for software engineering research": available at: https://www.cs.helsinki.fi/en/courses/582703/2013/s/k/1

Mark Beyer (2012), The Importance of 'Big Data': A Definition , Douglas. Gartner, available at: https://www.gartner.com/doc/2057415/importance-big-data-definition

Mehta, M., Agrawal, R., & Rissanen, J. (1996). SLIQ: A fast scalable classifier for data mining. In Advances in Database Technology—EDBT'96 (pp. 18-32). Springer Berlin Heidelberg.

Mena, J. (2011). Machine Learning Forensics for Law Enforcement, Security, and Intelligence. CRC Press.

Metwally, A., Agrawal, D., & El Abbadi, A. (2005). Efficient computation of frequent and top-k elements in data streams. In Database Theory-ICDT 2005(pp. 398-412). Springer Berlin Heidelberg.

Michie, D., Spiegelhalter, D. J., & Taylor, C. C. (1994). Machine learning, neural and statistical classification.

Mitchel, T.M. (1997). Machine Learning, Mc-grow Hill Science/Math/Engineerings

Misra, J., & Gries, D. (1982). Finding repeated elements. Science of computer programming, 2(2), 143-152.

Mohammad N. and Al-Jaroodi J. (2014), Real-time big data analytics: applications and challenges, International Conference on High Performance Computing & Simulation (HPCS), DOI: 10.1109/HPCSim.2014.6903700, pp 305-310

Murdopo, A., Severien, A., Morales, G. D. F., & Bifet, A (2013), Developer's Guide, Yahoo Labs, Barcelona, (2013) July

Murdopo (2013), Distributed Decision Tree Learning for Mining Big Data Streams, Master of Science Thesis in Distributed Computing, Available (online) at: http://people.ac.upc.edu/leandro/emdc/arinto-emdc-thesis.pdf

Mena-Torres, D., & Aguilar-Ruiz, J. S. (2014). A similarity-based approach for data stream classification. Expert Systems with Applications, 41(9), 4224-4234.

Nurseitov, N., Paulson, M., Reynolds, R., & Izurieta, C. (2009). Comparison of JSON and XML Data Interchange Formats: A Case Study. Caine, 2009, 157-162.

O'Callaghan, L., Meyerson, A., Motwani, R., Mishra, N., & Guha, S. (2002, February). Streaming-data algorithms for high-quality clustering. In 2013 IEEE 29th International Conference on Data Engineering (ICDE) (pp. 0685-0685). IEEE Computer Society.

Pang, B., & Lee, L. (2008). Opinion mining and sentiment analysis. Foundations and trends in information retrieval, 2(1-2), 1-135.

Pak, A., & Paroubek, P. (2010). Twitter as a Corpus for Sentiment Analysis and Opinion Mining. In LREC.

Park, B. H., & Kargupta, H. (2002). Distributed data mining: Algorithms, systems, and applications.

Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). A design science research methodology for information systems research.Journal of management information systems, 24(3), 45-77.

Pfahringer, B., Holmes, G., & Kirkby, R. (2008). Handling numeric attributes in hoeffding trees. In Advances in Knowledge Discovery and Data Mining (pp. 296-307). Springer Berlin Heidelberg.

Pfahringer, B., Holmes, G., & Kirkby, R. (2007). New options for hoeffding trees. In AI 2007: Advances in Artificial Intelligence (pp. 90-99). Springer Berlin Heidelberg.

Provost, F. J., & Buchanan, B. G. (1995). Inductive policy: The pragmatics of bias selection. Machine Learning, 20(1-2), 35-61.

Quinlan, J. R. (1993). C4. 5: programs for machine learning (Vol. 1). Morgan kaufmann.

Rajaraman, A., & Ullman, J. D. (2012). Mining of massive datasets. Cambridge University Press.

Rahnama, Amir Hossein Akhavan, (2014, Nov). Distributed real-time sentiment analysis for big data social streams, 2014 International Conference on (pp. 789 - 794). IEEE. In Control, Decision and Information Technologies (CoDIT), DOI: 10.1109/CoDIT.2014.6996998

Salton, G., & Buckley, C. (1988). Term-weighting approaches in automatic text retrieval. Information processing & management, 24(5), 513-523.

Sarnovsky, M., & Vronc, M. (2014). Distributed boosting algorithm for classification of text documents. In Applied Machine Intelligence and Informatics (SAMI), 2014 IEEE 12th International Symposium on (pp. 217-220). IEEE

Shafer, J., Agrawal, R., & Mehta, M. (1996, September). SPRINT: A scalable parallel classi er for data mining. In Proc. 1996 Int. Conf. Very Large Data Bases (pp. 544-555).

Shrivastava, N., Buragohain, C., Agrawal, D., & Suri, S. (2004). Medians and beyond: new aggregation techniques for sensor networks. InProceedings of the 2nd international conference on Embedded networked sensor systems (pp. 239-249). ACM.

Silver, M. S., Markus, M. L., & Beath, C. M. (1995). The Information Technology Interaction Model: A Foundation for the MBA Core Course. MIS quarterly, 19(3).

Street, W. N., & Kim, Y. (2001). A streaming ensemble algorithm (SEA) for large-scale classification. In Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 377-382). ACM.

Toivonen, H. (1996). Sampling large databases for association rules. In VLDB (Vol. 96, pp. 134-145)

Wang, H., Fan, W., Yu, P. S., & Han, J. (2003). Mining concept-drifting data streams using ensemble classifiers. In Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 226-235). ACM.

Yang H., Fong S. (2011), Moderated VFDT in stream mining using adaptive tie threshold and incremental pruning, Proceedings of the 13th international conference on Data warehousing and knowledge discovery, (pp 471-483), Springer-Verlag Berlin, Heidelber

Yu, J. X., Chong, Z., Lu, H., & Zhou, A. (2004, August). False positive or false negative: mining frequent itemsets from high speed transactional data streams. In Proceedings of the Thirtieth international conference on Very large data bases-Volume 30 (pp. 204-215). VLDB Endowment.

Zaki, M. J., Ho, C. T., & Agrawal, R. (1999). Parallel classification for data mining on shared-memory multiprocessors. In Data Engineering, 1999. Proceedings., 15th International Conference on (pp. 198-205). IEEE.

Zliobaite, I., Bifet, A., Holmes, G., & Pfahringer, B. (2011). MOA Concept Drift Active Learning Strategies for Streaming Data. Journal of Machine Learning Research-Proceedings Track, 17, 48-55.

Zhao, W., Ma, H., & He, Q. (2009). Parallel k-means clustering based on mapreduce. In Cloud Computing (pp. 674-679). Springer Berlin Heidelberg.