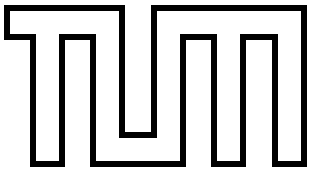


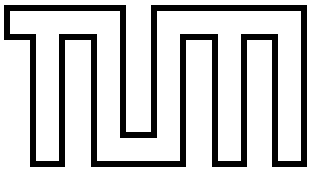
FAKULTÄT FÜR ELEKTROTECHNIK UND INFORMATIONSTECHNIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Electrical and Computer Engineering

Large Scale Anomaly Detection Using Spark

Jan Lauinger



FAKULTÄT FÜR ELEKTROTECHNIK UND INFORMATIONSTECHNIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Electrical and Computer Engineering

Large Scale Anomaly Detection Using Spark

Großflächige Anomalie Erkennung von Schadsoftware mit Spark

Author:	Jan Lauinger
Supervisor:	Prof. Dr. -Ing. Georg Sigl
Advisor:	Huang Xiao
	Ma. George Davis Webster III.
Date:	August 17, 2016

I assure the single-handed composition of this thesis only supported by declared resources.

Munich, August 17, 2016 Jan Lauinger

Abstract

Keeping up with the increase of data set sizes as well as application domains concerning anomaly detection is the challenge of modern big data processing engines. In most cases the development and main usage of these execution engines constrains itself to other application domains than anomaly detection. It is therefore important to clarify and test how a modern large scale processing engine deals in regard to a different application field such as anomaly detection. We implement all intermediate steps of the analytic life cycle of an anomaly detection process during this thesis. Therefore we perform database extractions, raw data preprocessing, anomaly detection through machine learning models, and visualization of model metrics using Spark. The either external developed or in Spark's machine leaning library included data structures compose our models. Spark manages to provide flexibility and solutions to all problem types through analytic processing stages. Especially into Spark's machine learning library included models produce very fast and accurate detection results. The interaction bottleneck of Spark versus external data structures marks the key factor for achieving fast overall detection processes. An extensive use of Spark data structures is thereby mandatory. The study does not cover all detection techniques and application domains of anomaly detection but rather provides general behavior characteristics of Spark in regard to anomaly detection. Additionally it opens the question to further explore data structure patterns that match the Spark advanced execution engine's requirements.

Zusammenfassung

Das Mithalten mit dem stetigen Wachstum von Datensätzen und Anwendungsgebieten der Erkennung von Schadsoftware fordert moderne Big Data Verarbeitungs-Triebwerke heraus. Die Entwicklung und Anwendung solcher Triebwerke beschränkt sich hauptsächlich auf andere Anwendungsgebiete als die Erkennung von Schadsoftware. Deshalb ist die Untersuchung des Verhaltens moderner Verarbeitungs-Triebwerke in Bezug auf die Erkennung von Schadsoftware wichtig, da sie den Anwendungsbereich dieser Triebwerke erweitert sowie in einem anderen Kontext testet. In dieser Bachelorarbeit implementieren wir alle Zwischenschritte eines Schadsoftware Erkennungsprozesses mit Spark. Dazu zählt die Vorverarbeitung von rohen Daten, die Anomalie Erkennung durch maschinelles Lernen und die Visualisierung der Eigenschaften der Erkennungsmodelle. Die Anomalie Erkennungsmodelle bestehen dabei entweder aus von Spark zur Verfügung gestellten oder externen Datenstrukturen. Spark bietet flexible Lösungen zu auftretenden Problemen während der Prozessanalyse. Allem

voran liefern die durch Spark zu Verfügung gestellten Erkennungsmodelle sehr schnelle und genaue Resultate. Der Engpass des Zusammenspiels zwischen von Spark gestellten und externen Datenstrukturen beeinflusst die Gesamtperformance des Erkennungssystems stark. Ein extensiver Gebrauch von Spark Datenstrukturen ist daher notwendig für schnelle und genaue Erkennungsresultate. Diese Arbeit bietet keinen Überblick über alle Techniken und Anwendungsgebiete von Anomalie Erkennung. Vielmehr verdeutlicht sie Charakteristiken und Verhaltensmerkmale von Spark in Bezug auf die großflächige Anomalie Erkennung von Schadsoftware.

Acknowledgements

I take this opportunity to express gratitude to Prof. Dr. -Ing. George Sigl, Prof. Dr. Claudia Eckert, and all faculty members. This project would not have been possible without them. The work was challenging, satisfying, and very educational to me. The biggest acknowledgment goes to Huang Xiao and George Webster who guided me through this project. They supported, inspired and motivated me to master all challenges. They were exceptional advisors. Eventually, I would like to thank my family and close friends for the mental support and love.

Contents

1 Introduction

1.1 Motivation

Anomaly detection finds patterns in data sets that do not match the expected behavior. The increase of applications domains of anomaly detection indicate the rising importance of anomaly detection solutions. The usage of anomaly detection systems expands to domains such as fraud detection for credit cards, insurance, health care, intrusion detection for cyber-security, fault detection in safety critical systems, and military surveillance for enemy activities [9]. Most application domains deal with large data sets. This requires fast engines for large-scale data processing. Anomalies appear differently because of diverse motives and context. In order to handle the diversity in demand, many new anomaly detection techniques have been developed by numerous research communities. As such, their exists many specifically developed techniques which supply solutions to their applicable domain [9].

The importance of tools for optimizing the anomaly detection process is obvious. The following example provides an insight into the new dimensions of damage caused by cybercrime. In 2014 the FBI revealed losses of $226 million due to compromised business email accounts. In the same year JPMorgan experienced the largest cybercrime hack which affected 100 million financial services of 75 companies[25]. Those new dimensions of attacks lead to new approaches by governments and corporations to tackle criminal enterprises.

Concerning the amount of data today, it is desired to pass data through an analytic life cycle to gain necessary information. The structured and modular schema of the analytic life cycle provides an overview of information retrieval stages. This clarifies intermediate processing steps and allows the modular exchange of processing techniques. Since large scale execution engines have been used to perform analytic data processing, processing systems mingle processing stages when they do not take clarifying processing schemes into account. To work hand in hand with the analytic life cycle processing steps become separated which leads to easier optimization arrangements within each stage. This approach leads to general progress within the analytic system.

The following systems implement existing solutions. These systems automatically handle difficult aspects of distributed computing such as fault tolerance, scheduling, synchronization, and communication. MapReduce [14] and Dryad [20] represent systems that enable to cover a wide range of algorithms through flexible implementation possibilities. On the other side they are awkward or inefficient to use concerning other algorithms. Problems typically arise with iterative machine learning algorithms which require more powerful execution en-

gines [29]. This is due to the lack of support to processing structures that contain branches and cycling. Cyclic data flow allows to detect load imbalance and repartitioning of processing tasks. For instance local minimums of complex iterative machine learning calculation can provoke reorientation of parameter convergence. This requires more flexible arrangements which cyclic data flow supplies. Furthermore the mentioned systems use shared memory concurrency which is compared to faster deterministic in memory computation not the optimal solution concerning computation expensive iterative algorithms[45].

Apache Storm [42] and Apache Spark [41] are powerful execution engine solutions for the analysis of big data. Commonalities of both engines are scalability, fault tolerance, simplicity and the property that both can be used in various programming languages. Additionally Spark provides faster data processing and incorporates developer friendly packages. The machine learning library targets large-scale learning settings that benefit from data-parallelism or model-parallelism. Due to support of general execution graphs and in memory computation Spark's optimized execution engine satisfies large scale processing requirements [27]. Regarding anomaly detection, the listed problems still occur.

- The limited machine learning library prevents the detection of specific large scale anomaly detection problems where missing algorithms are demanded.
- Spark's data structures do not always provide fluent accessibility and thereby restrict the flexibility of reacting to unknown tasks.
- The infrastructure of Spark might slow down external integrated algorithms.

This thesis investigates large scale anomaly detection using Spark. Thereby it compares different large scale anomaly detection algorithms with regard to speed, accuracy, and interaction possibilities of external and through Spark provided data structures.

1.2 Research Questions

The reason for tackling the large scale anomaly detection problem using Spark is the existence of an analytic system composed of different software. Additionally this systems stores a big amount of maleware. As there are the following research questions, this system enables the possibility of writing a thesis exploring these questions. The large scale data processing engine of the analytic system is Spark. Large scale anomaly detection using Spark is relatively novel as not many data analysts utilize Spark for large scale anomaly detection. Therefore the basic idea is to explore large scale anomaly detection with Spark.

The first general question asks how Spark can be used for large scale anomaly detection. Spark includes a limited machine learning library (MLlib) [27] which could solve and attribute to the implementation of large scale detection models. Besides that the question arises of how Spark behaves concerning other processing stages. Taking a closer look at large scale anomaly detection problems the question becomes more complicated. It is not clear if Spark supplies the demanded flexibility on all different aspects of the large scale anomaly

detection problem. It is interesting to figure out how Spark's architecture and code design handles intermediate tasks and stages of the large scale anomaly detection process. Because all intermediate analytic steps during the large scale anomaly detection process affect the final performance regarding speed and accuracy of the model. One more point is how effective Spark's data structures are with processing gained information concerning evaluation or visualization.

The limited machine learning library of Spark sets the next question of the project. Due to the limitation of the machine learning library, the included techniques and algorithms do not solve all possible large scale anomaly detection problems. On account of this the thesis investigates how effective external algorithms for missing large scale anomaly detection techniques work together with Spark's code design, and also how easily they can be integrated. Thereby sacrificing beneficial data structures of Spark misses the goal. It is rather efficient trying to keep the system as fast as possible with the frequent use of advantages of Spark's architecture.

The third and last question deals with optimization suggestions. It is valuable for future developers to know benefits and drawbacks about the realizations of large scale anomaly detection algorithms using Spark. For that reason this project finds strengths and weaknesses of models implemented with external algorithms, and then compares those algorithms with existing predictive algorithms in Spark. Finally this work explains how and where to look deeper into detail for improving the overall detection models. Apart from that, it is a goal to solve the question if there are indicators that cause variance regarding accuracy and speed using different sizes of input data sets.

1.3 Contribution

The exploration of large scale anomaly detection with Spark was done through several implementations of anomaly detection algorithms. Thereby building unsupervised as well as supervised learning algorithms allowed to react flexibly on either labeled or unlabeled input data instances. In the case of large scale anomaly detection algorithms with a supervised learning algorithm for detecting anomalies, this thesis first processed data instances with Spark's data structures and methods to label, normalize, and conform the instances. The general next stage was processing the valid input data samples with the large scale detection algorithms. Finally calculating evaluation metrics and visualizing the accuracy of the detection algorithms set the last processing stage. Apart from this the thesis measured and compared runtimes of each detection model. Time measurements referred to data preprocessing, training and prediction calculations of detection algorithms, and evaluation metric calculations.

With the comparison of the implemented large scale detection models optimization suggestions were provided. The comparison of evaluation metrics of Spark's machine learning library included and external developed algorithms revealed benefits and drawbacks about

the detection models. The input data sets varied in sizes and provoked diverse accuracy results of the detection models. The external developed algorithms contained as much code design of Spark as possible due to the beneficial large scale processing properties of Spark's data structures. Runtimes of Spark's data structures in contrast to regular Scala code were tested. This thesis highlighted adverse properties of Spark's code design as well. Moreover it discussed the difficulties of incorporating external code structures and methods into Spark's data structures.

With the mentioned arrangements and discussions this thesis provided insights into the behavior and flexibility of Spark to large scale anomaly detection problems. Spark had no pre-defined data structures and techniques to satisfy all diverse problem characteristics of large scale anomaly detection. This work therefore clarified the behavior of Spark to more remote application domains of large scale processing engines.

To conclude:

- The thesis implemented different large scale anomaly detection algorithms with Spark
- The thesis processed different sizes of data sets and visualized evaluation metrics of the algorithms
- The thesis measured runtimes of data preprocessing, training and prediction calculations of detection algorithms, and evaluation metric calculations
- The thesis discussed pros and cons of interaction difficulties between Spark and external data structures

1.4 Organization

The following two chapters explain terms, concepts, and techniques which are used throughout the study. Moreover they provide an overview of the basic infrastructure setting and the analytic setup of the project. The implementation chapter afterwards sums up all actions that were taken during the different implementation stages. The following three chapters evaluate gained results, discuss findings and flaws of the study, and illustrate occurring limitations. Based on the achieved results the last two chapters further explore findings and trends of the study. Optimization suggestions of critical flaws indicate future work to extend and further clarify the outcomes of the project. Apart from that the project becomes classified in regard to recent and current related areas of research.

2 Fundamentals

2.1 Anomaly detection

Basic anomaly detection proceedings follows a certain analytic schema. Anomaly detection techniques establish a profile of the characteristics and properties of the inspected sample or instance. The findings about instances refers to normal behavior. Afterwards anomaly detection techniques compare characteristics of unknown samples with regard to the normal profile on which they had been trained on before. In the case of significant distinctions from a new instance to the normal profile, anomaly detection techniques predict unknown samples as malicious [51]. Apart from that the anomaly detection problem consists of several intermediate process solutions.

The nature of the input data, the availability of labeled data instances, and the requirements and constraints which depend on the application domain formulate a specific anomaly detection problem [9]. All these problems together form and compose the anomaly detection problem. The next points explain the problems more accurately:

- The nature of input data as the first intermediate problem clarifies diverse structures and occurrences of data instances. Various examples therefore are forms such as vector, event, object, and pattern structures. Each instance is characterized by values of attributes that indicate different aspects and characteristics of the instance [48]. So processing raw instances first for receiving the desired pattern and information is necessary.
- Secondly based on the nature, context, behavior or cardinality, anomalies are classified into following three categories of Point Anomalies, Contextual Anomalies and Collective Anomalies [18]. Therefore each section requires different treatment.
- Another point is the property of either labeled or unlabeled data instances. Labeled data sets are often available only in small quantities whereas unlabeled data sets can be abundant. Labeled data sets though are essential for supervised learning [54]. Therefore it is necessary to provide both processing options with supervised and unsupervised learning algorithms.
- Lastly the outputs produced by anomaly detection techniques are either labels or scores [9]. Thresholds enable final outcomes in the case of scores while labels deliver results immediately.

2.2 Analytic Life Cycle

The analytic life cycle which is also known as intelligence cycle embodies a general methodology for information identification and analysis[44]. Regarding large scale anomaly detection the following six different modules compose the processing schema.

- The requirement module defines the goal of the project or system. Large scale anomaly detection aims to identify anomalies in large scale data sets.
- The information sources identification module cares about gathering suitable information. This study finds appropriate data sets with the help of the the virustotal[2] and PEINFO [1] services.
- The information collection module truly performs data collection.
- The information processing module creates valid data sets through preprocessing for further processing.
- The module of information analysis and intelligence production builds and runs large scale anomaly detection techniques during this thesis.
- The reporting module makes determinations based on the gained results.

The modular composition of processing stages allows the exchange of diverse implementations within a module. Application domains indeed have to resemble for enabling module transfer. Moreover the structural approach of data processing brings analytic compliance.

2.3 Big Data Processing

Companies nowadays collect unprecedented amounts of data with the help of the internet. Not surprising seem facts that Google processes hundreds of Petabytes of data per month. Among others Facebook generates log data of over 10 Petabytes per month [10]. If companies do not have the resources to build or buy large scale data processing infrastructures they remain holding tremendous amounts of data. Depending on how dependent these companies are to data processing they lose money. As a consequence the demand for cheaper and better data processing systems intensifies the competition of building improved large scale processing systems.

Building even faster and more advanced processing infrastructures brings challenging problems. First there is the integration and collection of massive data sets from widely distributed data sources. Second the quantity of data and its mutual relations will far surpass the capacity of the information technology architectures and infrastructure of existing companies. Moreover real-time requirements stress the available computing capacity of current companies. Third it is difficult to store and manage huge heterogeneous data sets with moderate requirements on hardware and software infrastructure. Last in consideration

of heterogeneity, scalability, real-time, complexity and privacy, reducing data sets at different levels during the analysis, modeling, visualization and forecasting improves decision making [10].

To sum up big data processing addresses and unifies domains such as parallel programming, distributed systems, and load balancing for scalable performance [38]. In big data processing diverse software application have to work hand in hand to provide decent results.

2.3.1 Apache Hadoop

The Apache Hadoop software library allows distributed processing of large scale data sets across multiple clusters of computers with its framework. This can be done with simple programming models. The Hadoop infrastructure scales up from single computers to thousands of machines. Each computer provides local storage and computation power. The software library is designed to detect and handle failures at the application layer. Hadoop is a distributed computing framework and marks the outermost infrastructure of a collection of diverse software frameworks. However, the standard Hadoop software setup consists of the following main modules [15]:

- Hadoop Common provides common utilities that support other Hadoop modules.
- Hadoop Distributed File System(HDFS) is a distributed file system providing high-throughput access to application data.
- Hadoop Yet Another Resource Negotiator(YARN) is a framework for job scheduling and cluster resource management.
- Hadoop MapReduce is a YARN-based system for parallel processing of large data sets.

The two fundamental and core pieces of Hadoop are the MapReduce framework and the Distributed File System [19].

The MapReduce execution engine is in charge of the fault tolerant distributed computing system. It processes large data sets stored in the cluster's distributed file system. The MapReduce paradigm follows separate map and reduce steps. It processes these steps in parallel and each step operates sets of key value pairs. The data transfer between nodes in the clusters interrupts the program execution of the map and reduce stage. Therefore the MapReduce paradigm divides its program execution [43].

HDFS is the file system component of the Hadoop ecosystem. It stores file system metadata and application data separately. Servers called NameNodes store metadata while servers called DataNodes store application data. The fully connected servers communicate using TCP-based protocols. HDFS does not use data protection mechanisms but rather replicate file content on multiple DataNodes for reliability. This allows more opportunities for locating computation near the processed data next to providing data durability [40].

2.3.2 Spark

Apache Spark [41] is a general engine for large scale data processing. Spark sits on top of Hadoop. It was originally designed to be a better processing engine then MapReduce on the Hadoop system. Due to its advanced Directed Acyclic Graph(DAG) execution engine that supports cyclic data flow and in-memory computation, Spark allows fast data processing. Spark runs programs up to 100x faster than Hadoop MapReduce. Additionally Spark generalizes two stage MapReduce and provides fast data sharing between operations [49]. Parallel applications are easy to build with Spark's high level operators and these applications can interactively be built through Java, Scala, Python or R shells. With built-in libraries Spark provides an analytic flexibility within applications built by Spark. A combination of different libraries in one app is possible. Spark is compatible with various storage systems. Hence, it can access data from HDFS, Cassandra, HBase, Hive, Tachyon or any Hadoop data source. One last feature of Spark is the fact that it can run in the standalone cluster mode or on distributed systems such as Hadoop.

2.3.3 Zeppelin

Apache Zeppelin [16] is a web based notebook which enables interactive data ingestion, data discovery, data analytics, data visualization, and data collaboration. It provides the development of data driven, interactive and collaborative documents. The multipurpose Zeppelin notebook allows with its interpreter concept any language and data processing back-end to be plugged to Zeppelin. Additionally it supports interpreters such as Apache Spark, Python, JDBC, Markdown and Shell. In particular, Zeppelin is compatible with Apache Spark and does not need to build a separate module or library for it. One of Zeppelin's most attractive features is data visualization. Basic visualization charts are already included but Zeppelin recognizes and visualizes output from any language back-end. Per simple drag and drop Zeppelin creates pivot charts and one more property is dynamic creation of input form to the notebook.

2.4 Machine Learning

Machine learning algorithms try to accurately predict actions based on past observations. Algorithms therefore approximate a function which is set together of all different observation cases from the input data. An easy example is to let a machine learning algorithm distinguish spam emails from emails. The approach taken by a machine learning algorithms is to start by gathering samples of spam and emails. In this case it is beneficial assuming labels that indicate whether an email is spam or not. Feeding the machine leaning algorithms with the data samples and the labels the algorithm is able to learn a classification rule on the given data. Afterwards the algorithm is able to predict and classify unknown emails [36].

The goal is to build a machine learning algorithms as accurate as possible. The vast number of different aspects of a central concern affect machine learning situations in many

ways. With that there is no single formula to solve a learning problem. There are rather many different and complementary views on a learning problem. The main characterizations of a machine learning problem are the learning tasks, the types of interactions between the learner and the environment and the forms of learning [11].

The learning task is generally left implicit in a learning system because it is determined by the type of input data and the learning goal. Learning goals define the relevance, acquirability, and predictability of knowledge of the input data.

Supervised learning and unsupervised learning are the main types of interactions between the learner and the environment. Besides that batch and online learning methods complete the different learning methods in learning systems. This work explains the main types of interactions in short because explaining all methods in detail is beyond the scope of this thesis. Data analysts use either supervised learning when there are labeled instances or unsupervised learning when there are unlabeled instances. Learning through the increase of knowledge or through the transformation of representation of knowledge mark different learning forms in machine learning. Explaining further details on learning forms is beyond the scope of this thesis.

To sum up machine learning algorithms are located in these described domains to provide accurate results to a learning problem. Providing predictions on unknown samples, machine learning algorithms allow analysts to evaluate data instances without knowing the actual behavior of the sample. The following sections provide properties of machine learning algorithms and explain them in more detail.

2.4.1 Classification and Clustering

In machine learning the classification task is an instance of supervised learning. Whereas the corresponding clustering task belongs to methods of unsupervised learning. In supervised classification labeled data is available. The classification method maps this labeled data to a finite set of discrete class labels. The expression of binary classification stands for a mapping to a set of just two class labels. A mathematical function with adjustable parameters models the mapping process. An optimization algorithm determines these parameters. The classification process is capable of classifying new data instances after the parameters are set [50].

In unsupervised clustering no labeled data is available. Clustering separates unlabeled data into a finite and discrete set hidden data structures [50]. All clustering techniques explore and cluster the nature of unknown data. Especially large scale analysis depend on cluster analysis because it provides a compressed representation of the data.

2.4.2 Naive Bayes

The Naive Bayes algorithm is a supervised learning method for classification and predictor selection. It calculates the conditional probability distribution of each feature given the label. Afterwards it computes the conditional probability distribution of label given an input instance with the Bayes theorem [21]. The prediction as a result of the calculation determines the assignment of the input instance. A single data pass determines the probabilities and the final prediction [28]. In the case of an instance having a fixed number of independent features with discrete binary values of input, choosing the Bernoulli Naive Bayes model is more appropriate because it simplifies the calculation [24]. Regarding large scale data processing less computational effort leads to faster processing results.

2.4.3 DBSCAN

The Density-Based Spatial Clustering of Application with Noise(DBSCAN) algorithm identifies clusters in large spatial data sets by looking at the local density of input instances. Therefore the DBSCAN algorithm counts as a method of unsupervised learning. The user sets only one input parameter. Furthermore the user needs minimal knowledge about the process because the algorithm provides a suggestion of the input parameter through execution of itself. Additionally DBSCAN is capable of determining which information to classify as noise or outliers. For these reasons the DBSCAN algorithm scales with the size of the input instances [6].

2.4.4 Logistic Regression

Logistic regression follows the same principles of linear regression but varies in the end as it returns a classification value. In linear regression input features together with adjustable parameters calculate the strength of relationship between these values through a linear function. The result of the calculation is the prediction to the given input features. Again an optimization algorithm determines the adjustable parameters. The optimization algorithm in this supervised learning case takes the given labels to optimize a error function [26]. The difference with logistic regression is the fact that logistic regression passes the outcome of linear regression through a sigmoidal function. The sigmoidal function produces a threshold that can be used to classify the input instance.

2.4.5 Support Vector Machine

Support Vector Machines solve classification and regression problems. In addition they are a set of related methods for supervised learning. In the classifying procedure the SVM classifiers build a maximum margin hyperplane. This hyperplane lies in a transformed input space and splits the classes given by input instances. Commonly similarity matching functions create the transformed input space. Moreover the hyperplane maximizes the distance to the nearest cleanly split features. In the case of a linear SVM, there is no transformed input space because the hypothesis or classifying function straightforwardly

calculates the result with the adjustable parameters and the input features. A quadratic programming optimization problem determines the adjustable parameters of the solution hyperplane [39].

3 Methodology

An infrastructure for data analysis has to manage the interaction of various applications, machines, and networks. For receiving the results this study provides it is necessary to set up a similar environment. Small distinctions in implementation details affect the complexity of infrastructures. This easily changes and distorts evaluation measurements. Additionally the choice of components for setting up an general data processing infrastructure is very important. Services, applications, and frameworks highly depend on each other to supply demands such as scalability or resilience.

3.1 Infrastructure

The whole data analysis set up is based on the Scalable Architecture for Feature Extraction, Multi-User Analysis, and Real-Time Information Sharing(SKALD) [47]. Skald provides the required architecture for asynchronous feature extraction. The scale of feature extraction handles increasing amount of information and is resilient to system failures. Moreover it is flexible enough to incorporate the latest technology trends. Skald's infrastructure allows advanced analytics over extracted features. In addition it permits preferred tools for information retrieval and sharing. Sharing data through Skald with others does not require the release of raw data. Skald is the outermost architecture and consequently the highest system abstraction. It incorporates the following software components into its design.

3.1.1 Software

Infrastructure

The infrastructure to extract and handle data is based on an implementation of Skald. This software framework is Holmes Processing [46]. Holmes Processing depends on at least two external services. The first is the HTTP file server and the second is a queueing server. The next paragraphs describe important dependencies of the Holmes Processing framework.

Holmes Processing extracts data using hardware with 5 servers with each having 8GB of RAM and 4 CPU cores. During the data extraction the services Virustotal [2] or PEINFO [1] scrap for the results of a sample. After data extraction through these services the Holmes Processing framework stores gained information in different databases. These databases interact as components of the distributed computing framework of Hadoop. The large scale

processing engine of Spark replaces the MapReduce execution engine which is usually been utilized within the Hadoop ecosystem. It is now possible to load data from the Hadoop File System into Spark data structures due to the fact that Spark runs on top of the Hadoop File System. As a front end user interface Holmes Processing uses Zeppelin which allows users to interact with all software components.

The resilient, scalable and fast software framework of Holmes Processing manages and connects diverse applications and services. With that it represents an efficient framework for complex large scale data-oriented analytics. Due to these reasons it sets the basic software infrastructure of this thesis.

Transport

RabbitMQ [32] runs on a server with 8GB of RAM and 2 CPU cores. This messaging service embodies the transport layer in the Skald architecture. The reasons for the integration of RabbitMQ to the Holmes Processing framework are the properties of robust messaging, the simplicity to use, and the compatibility to all major operation systems such as Windows, Linux/Unix, and MAC OS X.

Database

Holmes Processing stores data on two different storage engines. The first storage engine is a distributed database solution called RIAKCS [3] which runs on 4 servers with each having 64GB of RAM, 8 cores, and 2TB storage. The purpose of the RIAKCS storage engine is to store PE32 object files. The second storage engine is Apache Cassandra [4] which stores other data instances. Cassandra runs on 3 servers with each having 32GB or RAM, 8 cores, and 4TB SSD storage. Scalability and fault tolerance properties support the choice to use the cloud storage infrastructure of Cassandra.

File System

The Hadoop [15] Distributed File System(HDFS) represents the into the Homes Processing framework integrated file system. It runs on 9 servers with each having 30GB of RAM and 6 cores. The Holmes Processing framework pulls data locally to the HDFS. Spark enables to fetch data from the HDFS for further processing. Into the HDFS stored data consist of 4 single files. Two files are virustotal [2] withdrawal results containing 200k and 1m samples. The other two files are PEINFO [1] extracted data sets of again 200k and 1m samples.

Execution Engine

Apache Spark [41] is the large scale processing execution engine living on top of the HDFS. This means that Spark servers are the same as the HDFS servers. Spark supports libraries to realize complex analysis, provides interactivity through various programming languages, and is fast due to in memory computation and application parallelism. It supports cyclic data flow which is an important feature regarding iterative computing. All these different facts are beneficial for large scale anomaly detection and explain the choice of Spark over MapReduce. The analytic system uses Spark version 1.6.

Interface

Apache Zeppelin [16] is the front end user interface running on top of all applications of the Holmes Processing framework. Zeppelin's web based notebooks enable interactive data analytics with multiple languages and other plugins. The integration of Spark is possible. This allows data analysis and visualization with Spark data structures in Zeppelin. The data analysis infrastructure uses Zeppelin version 0.6.

3.2 Data Set

The services of Virustotal [2] and PEINFO [1] create the data set samples. The PEINFO [1] service uses a custom tool to extract meta information from the PE32 file format [31]. This service is a multi-platform module to parse and access PE32 files. PE instances are data structures that contain all attributes of the Windows PE32 file format [31] as accessible instances. The PEFILE service enables to interact with attributes of the PE32 instance through capabilities such as the reading and writing standard header members, iterating through the inner structure sections, listing imported or exported symbols, and dumping or analyzing information. Among others Virustotal [5] uses the service of PEINFO. Virustotal is a free malware detection software service. It allows the detection and fast analysis of all sorts of suspicious malware. Moreover it stores the anlytic results into useful JSON [12] data structures.

Samples are extracted with the SHA256 hash values [17]. The hash value is the pivot that matches a specific PEINFO and Virustotal result. This allows to deliberately extract data samples. Collections of samples are stored into the HDFS in the form of data sets containing either 200k or 1m samples.

The binaries of these data sets represent diverse malware types. Traditional criminal malware, highly advanced state-sponsored malware, and highly suspicious programs are examples for different malware types. Approximately 19% of these samples are packed and obfuscated Which is a large volume of the samples. Hence, it is necessary to mention that there is no unpacking or deobfuscation during the extraction procedure. This text paragraph is used with permission from George Webster. These statistics are from a paper in review.

The figure 3.1 on the next page pictures the structure of a virustotal data instance. The large amount of detection scans within a sample justifies the omission of some scan results.

```
{
  "scans":{
    "Bkav":{
      "detected":false,
      "version":"1.3.0.7133",
      "result":null,
      "update":"20150822"
    },
    "MicroWorld-eScan":{
      "detected":true,
      "version":"12.0.250.0",
      "result":"Adware.Linkury.M",
      "update":"20150822"
    },
    "nProtect":{
      "detected":true,
      "version":"2015-08-21.01",
      "result":"Trojan/W32.Agent.86016.DRW",
      "update":"20150821"
    },
                .
                .
                .
    "CMC":{
      "detected":false,
      "version":"1.1.0.977",
      "result":null,
      "update":"20150819"
    },
  "scan_id":"909
     ecb0561b25cfa132c851dd57bf2b375bfa61d6aa6275abe75cb1b
     33bb3e48-1440261007",
  "sha1":"2cd49035a22801e0b4878487fc2406632f2a0b1f",
  "resource":"24827004ab7ff06dc0569705b680ae50",
  "response_code":1,
  "scan_date":"2015-08-22 16:30:07",
  "permalink":"https://www.virustotal.com/file/909
     ecb0561b25cfa132c851dd57bf
     2b375bfa61d6aa6275abe75cb1b33bb3e48/analysis/1440261007/",
  "verbose_msg":"Scan finished, information embedded",
  "sha256":"909ecb0561b25cfa132c851dd57bf2b375bfa61d6aa6275abe75
     cb1b33bb3e48",
  "md5":"24827004ab7ff06dc0569705b680ae50"
}
```


Figure 3.1: VirusTotal Raw Data Sample

4 Implementation

The first step was to clarify the anomaly detection procedure. The process was therefore partitioned into the components. After considering the initial situation, the goals for the evaluation of the anomaly detection system using Spark were to measure the system's speed, accuracy, and interaction competence of external and into Spark included algorithms. The decomposition of the process helped to more accurately evaluate the large scale anomaly detection system at different stages.

4.1 General Setup

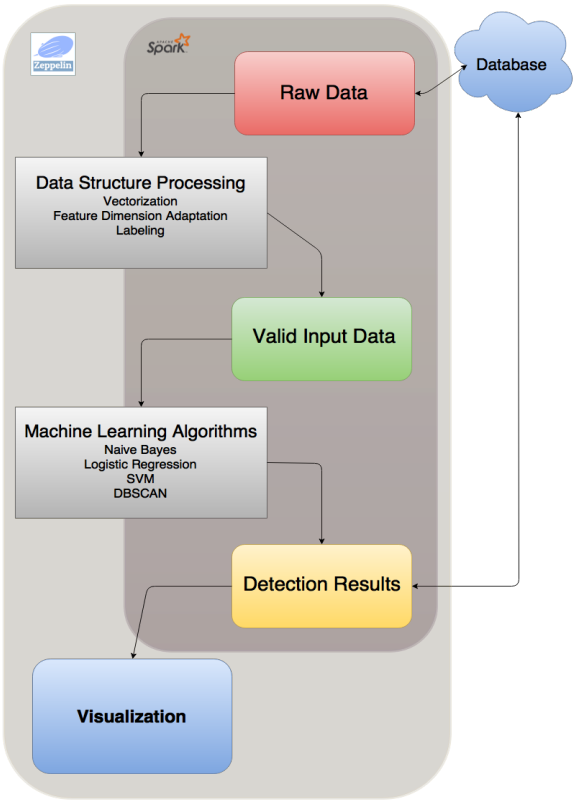


Figure 4.1: Anomaly Detection Process

The figure 4.1 shows the large scale anomaly detection setup for this thesis. The light gray shadow represents Zeppelin as a user front end interface running on top of all applications.

The darker gray shadow indicates the stages where Spark data structures had been used. The large scale anomaly detection process composes of three main stages. Data structure processing, machine learning algorithms, and visualization embody these stages. The red, green, and yellow boxes stand for the processing states of the data sets. The following sections of in this chapter refer to the stages and states which are displayed in the figure 4.1. The arrows clarify the chronology of consecutive stages.

The initialization of a new Zeppelin notebook for the project marked the first step. The Holmes processing framework enabled to extract raw data into the Cassandra storage and with that into the HDFS. Spark afterwards allowed to load the data sets into Spark's resilient distributed dataset(RDD) structures [52]. As the stored data was in JSON format it was necessary to call a conversion method of Spark to transform the initial format to the RDD format. Subsequently the machine learning algorithms and all supporting and necessary data structures from Spark's machine learning library were imported. Other imports facilitated the implementation of external and by ourselves designed algorithms. Scala was the choice of programming language because Spark has been developed in Scala and it has efficiently worked on large data sets.

4.2 Data Structure Processing

During the first stage, the raw data sets had the structure described in figure 3.1. The first stage vectorized, adapted, and labeled all data instances. The goal of the first processing stage was to provide suitable input structures for all machine learning algorithms.

Containing raw data, the first variables in Zeppelin stored instances with Spark's SQL row structure in Spark's RDD data structures. To create valid input data structures for the machine learning algorithms out of Spark's machine learning library, methods of Spark's data structures accessed RDDs and returned new RDDs. This gave the possibility to modify the structure of the data instances, not only the structure of the data collections. Apart from this the machine learning algorithms required Spark's labeled point data structure as the inner data structure. Due to this the function that modeled the inner structures had to return labeled points in a RDD at the end. The labeled point structure again consisted of a double value as label and Spark's linear algebra vector as vector. This linear algebra vector represented the features of the instances but had less functionality than the regular Scala vector structure. Finally the creation of a linear algebra vector required an regular Scala array.

4.2.1 Vectorization

For vectorizing the raw data instances the RDD's map method applied the vectorization function 4.2 to each instance of the RDD. This applied function filtered out the "scans" dictionary of the data instance 3.1. Afterwards the function appended a +1 or -1 to an

regular array whether it detected a "true" or "false" in the different scan dictionaries. In the case of the input for the Naive Bayes algorithm the feature values were set to 0 or 1 depending on the detection of the virus scans. The returned array was temporarily stored in a RDD before the processing of the feature dimension adaptation started.

```
def vectorize(row: Row): Array[Double] = {

    // vectorization for SVM & LogRegrsion:
    var arr: Array[Double] = Array()
    for(word <- row(row.fieldIndex("scans")).toString.split(",")){

        if( word contains "false" )
            arr = arr :+ -1.0
        else if( word contains "true")
            arr = arr :+ 1.0
    }
    return arr
}
val data_array = raw_data_rdd.map(x => vectorize(x))
```

Figure 4.2: Vectorization Function & RDD Map Call

4.2.2 Feature Dimension Adaptation

The vectorization of the data sets did not solve the problem of diverse lengths of the feature vectors. The machine learning algorithms required equal input structures. To solve this problem average calculations averaged the lengths of the arrays in a RDD. Next the RDD's filter method filtered for arrays with that specific average length. At the end RDDs contained arrays of equal length. Even tough this schema shrank the size of a RDD, it was a possible way to conform the feature size.

4.2.3 Labeling

Passing the labeling function 4.3 as input, the RDD'S map method called this function on every data instances. The return type of that function was Spark's labeled point structure. An adjustable threshold parameter determined the label of the instance. This threshold parameter could range from zero to the size of the features and decided if an instance was either malicious or benign. The comparison value to that threshold was a count over all positive detection scans. If the count value was lower than the threshold, the label of that instance was 1.0. Otherwise the label was 0.0 and indicated a clean sample. The label appended with the vectorized array formed the return value in form of a labeled point. The vectorized array was therefore transformed into Spark's linear algebra vector structure. Besides all this, these arrangements did not cover all data sets with the labeling procedure because RDDs with unlabeled elements had to remain for unsupervised machine learning algorithms. In the end, a valid input data sample looked as shown in the figure 4.4.

```
def labeling(line:Array[Double]): LabeledPoint = {

    // if condition determines when data labeled to malicious
    if(line.take(dimension_of_features).count(_ == 1) < threshold)
        return LabeledPoint(1.0, Vectors.dense(line.take(dim_of_feats)))
    else
        return LabeledPoint(0.0, Vectors.dense(line.take(dim_of_feats)))
}
val data = data_array_cut.map(x => labeling(x))
```

Figure 4.3: Labeling Function & RDD Map Call

(0.0 [-1.0,-1.0,1.0,1.0,-1.0,1.0,-1.0,-1.0,1.0,-1.0,1.0,1.0,-1.0,-1.0,1.0,-1.0,1.0,-1.0,-1.0,-1.0,1.0,1.0,1.0,-1.0,-1.0,-1.0,1.0,1.0,1.0,1.0,-1.0,-1.0,1.0,-1.0,-1.0,1.0,1.0,1.0,-1.0,1.0,1.0,-1.0,-1.0,-1.0,-1.0,-1.0,-1.0,1.0,-1.0])

Figure 4.4: Labeled Point Element of RDD

4.3 Machine Learning Algorithms

Processing the valid input data with machine learning algorithms marked the second stage of the anomaly detection process. The goal in that stage was to train and create evaluation output for the visualization afterwards. First the valid input data had to be prepared though. Therefore each data set was split into smaller training and testing data sets. While 60% of the original valid data set turned into the training set, 40% became the test set. The samples were randomly partitioned. Due to the separation of each data set it was possible to take the training data set for training machine learning algorithms and the testing data set for evaluating the algorithms.

More precisely the training instances optimized the adjustable parameters of the different machine learning algorithms. Regarding the evaluation and the visualization, running the test data set through the trained machine learning algorithm provoked a detection result. With that detection result it was possible to measure the accuracy of the detection model. After the determination of the different machine learning algorithm parameters the predicted results of the test data instances came into the first column of a RDD. The second column of that RDD kept the original labels of the processed test data set. That RDD structure facilitated the evaluation of the machine learning algorithm. It allowed a comparison of the predicted versus the original labels of the test samples. For visualizing the accuracy of the algorithms, evaluation calculations determined the parameters for the the ROC and precision-recall curves of each detection result. Afterwards graphs depicted the plotted curves. This was done using the previously created RDD. The visualization section of this chapter explains the detailed calculation arrangements for the visualization values.

4.3.1 Naive Bayes

The data sets allowed the assumption of having conditional independent input instances. On that account it was possible to implement the Naive Bayes algorithm out of Spark's MLlib. The input to this algorithm had the labeled point structure with the feature values of 0.0 and 1.0. The parameter for additive smoothing λ kept it's default 1.0 value. This insured the calculation of the conditional probability in the Naive Bayes algorithm to work as it prevented the outcome of this calculation to become zero.

The fact of having binary feature values led to the choice of the bernoulli model type over the multinomial model type. Without having binary values the multinomial model type had to be chosen due to the ability of handling non binary feature values. Concerning large scale data processing as little computational effort as possible was beneficial. The bernoulli model type guaranteed less computational complexity and facilitated thereby the Naive Bayes algorithm.

The imported Naive Bayes algorithm was trained by calling its train method with the described input data set, λ, and model type. Afterwards calling the map method of the test set RDD marked the next action. The resulting RDD was the RDD for the evaluation of the machine learning algorithm. For receiving the predictions of the trained Naive Bayes algorithm the prediction method of the algorithm was called with the test data features.

4.3.2 Logistic Regression

The second machine learning algorithm was the Logistic Regression with LBFGS classification algorithm out of Spark's MLlib. The Limited Broyden Fletcher Goldfarb Shanno(LBFGS) optimization algorithm[22] in this algorithm optimized the adjustable parameters during the training procedure of the algorithm. This was done by minimizing the loss function in equation 4.1. The input instances were of the form of the regular labeled point structure described in figure 4.4. The default setting of the classification threshold value was 0.5. This threshold interpreted the output of the hypothesis function to become either 1.0 or 0.0. The default value of the intercept or constant of the regression equation was 0.0. The anomaly detection algorithms performed binary classification on data instances. That was why the number of classification classes was 2.

$$L(w; x, y) := \log(1 + e^{-yw^T x}) \tag{4.1}$$

$$f(z) = \frac{1}{1 + e^{-z}} \tag{4.2}$$

Having set all immutable algorithm modification values it was possible to train the imported algorithm with it's run method. The input to this function was the train RDD containing the described labeled point structures. The equation 4.2 set the prediction result to 1.0 if $f(w^T x) > 0.5$. Afterwards the predicted and original labels built the RDD for the evaluation of the machine learning algorithm. The RDD's map method accessed the labeled

point structure of the test instances and delivered the original label. The execution of the algorithm's prediction method with the test data features delivered the predictions.

4.3.3 DBSCAN

The third detection algorithm did not belong to Spark's MLlib. The unsupervised DBSCAN clustering algorithm did not require an optimization algorithm for adjustable parameter determination. Since a couple executions of the detection model already provided a sense of how to choose the ϵ distance value. The ϵ distance value was the threshold which determined if another instance belonged to the neighborhood or not. It therefore decided the density of instances through the comparison with the euclidean distance value between instances. The $minPts$ variable was 2 to mark instances having only itself as a neighborhood participant as a outliner. The density-based detection approach classified thereby samples and embodied a detection model. The return value of this detection algorithm consisted of a RDD keeping prediction results and original labels. Actually the unsupervised clustering approach did not require labeled instances. But the fact of having a reference to the predicted result kept us from discarding the original labels. Moreover this action conformed the DBSCAN algorithm to the other detection models which built a comparison RDD for the evaluation.

Since Spark's RDD was the input and output data structure the RDD form set the basic data collection structure. Moreover the RDD methods and the general RDD structure provided optimized large scale processing properties. First of all additional variables representing processing states extended the data instances in the RDD collection. Through passing external developed functions to RDD methods it was possible to modify states of instances. Besides state modification the n dimensional euclidean distance in equation 4.3 marked the only mathematical function solving the density decision problem.

$$d(p,q) = \sqrt[2]{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \ldots + (p_n - q_n)^2} \tag{4.3}$$

The euclidean distance calculated the neighborhood participants of instances. With the modification functions and the euclidean distance calculation it was possible to implement the DBSCAN algorithm having Spark structures as basic collections. Running the algorithm with a data set returned the prediction and original label RDD for the evaluation of the algorithm.

4.3.4 SVM

The next anomaly detection algorithm was the linear SVM with stochastic gradient descent(SGD) algorithm imported from Spark's MLlib. The SGD optimization algorithm optimized the adjustable parameters of the SVM algorithm during training. This was done by minimizing the loss function 4.4. The reason therefore was the performance of the SGD algorithm in regard to large scale data processing problems[8]. Again the first task was to determine all immutable modification parameters that would affect characteristics of the algorithm. Without automatic termination calculations the SGD algorithm required a iteration number for stopping the optimization iteration of the adjustable parameters of the

algorithm. This value was 100. The fast convergence of the SGD algorithm justified the choice of this parameter. Potential positive or negative hypothesis output was compared with a threshold of 0.0 and labeled the instance to one of two classes. The intercept value of the algorithm kept the default value of 0.0. The imported SVM algorithm used L2 regularization per default. This meant regulating the SVM's decision boundary concerning overfitting and bias.

$$L(w; x, y) := \max(0, 1 - yw^T x) \tag{4.4}$$

The input variables to the algorithm's train method were the number of iterations and the training set RDD containing labeled points of the form of figure 4.4. After the training the test set features fed the prediction method of the SVM algorithm. The equation $w^T x >= 0$ decided whether a prediction was 1.0 or 0.0. The prediction results and the original test instance labels formed the RDD for the evaluation of the machine learning algorithm.

4.4 Visualization

The prediction results of the different machine learning algorithms enabled the possibility of measuring the performance of these algorithms. With that it was feasible to display this outcome which implicitly meant to visualize the accuracy of the large scale anomaly detection process. The visualization process marked the last stage of the anomaly detection process.

The comparison of the predicted results and the original test label enabled the calculation of the parameters for the precision recall(PR) and the receiver operating characteristic(ROC) curves. This allowed to calculate the true positive, false positive, false negative, and true negative values for the determination of the precision, recall, and false positive rates. The true positive value represented the number of samples correctly predicted positive. The false positive value stood for samples wrongly predicted positive. False negative values were samples wrongly predicted negative. Lastly true negative values expressed samples correctly predicted negative [13]. The equations 4.5, 4.6, and 4.7 determined the precision, recall, and false positive rates.

$$Precision = \frac{true\ positive}{true\ positive + false\ positive} \tag{4.5}$$

$$Recall = \frac{true\ positive}{true\ positive + false\ negatives} \tag{4.6}$$

$$False\ Positive\ Rate = \frac{false\ positive}{false\ positive + true\ negative} \tag{4.7}$$

Every detection model had its own parameters for the PR and ROC curves. For displaying the curves there were 10 recall values plotted on the y-axis and depending on which curve 10 precision or false positive rate values plotted on the x-axis. The loop over a detection model with data sets having 10 different labeling thresholds made the plot possible. Even though Spark's MLlib provided class structures and methods for the calculation of evaluation metrics it was necessary to calculate these values explicitly. This meant to calculate the 10 resulting evaluation results and prepare a RDD containing all results. This RDD additionally hold border values for a correct visualization of the scale of axes. During each loop all algorithm evaluating parameters were calculated for the fact to have different algorithm comparisons. Additionally tables for the evaluation of speed were created. Therefore large scale anomaly detection process split itself into 3 different sections and measured each section separately. Preprocessing runtime, algorithm runtime, and visualization runtime set those 3 sections. The speed measurement of the visualization section became important due to two facts. It contained all calculations for the evaluation metrics and was a mixture of Spark and Scala structures.

Zeppelin SQL query commands display graphs. Providing the right data structures Zeppelin's query technique could display graphs in various ways with even switching the emphasis with regard to the displayed information. To fulfill the beneficial capabilities Zeppelin required transformations of the final RDDs into Spark's SQL data-frame structure. This allowed the illustration of evaluation results with SQL commands.

5 Evaluation

The implementation section proved the realization of anomaly detection processes using Spark. This set the prerequisite to evaluate Spark with regard to large scale anomaly detection problems. This chapter evaluates the algorithms and overall detection systems regarding speed, accuracy, and model interaction with Spark and points out violations.

5.1 Speed

Table 5.1: Speed Table Processing 200 Thousand Samples

Algorithm Name	Algorithm(train/predict) Runtime	Preprocessing Runtime	Visualization Runtime
Naive Bayes	32.77s	35.71s	34.98s
Logistic Regression	43.33s	36.38s	36.90s
DBSCAN	n/a	34.84s	n/a
SVM	50.86s	49.48s	34.78s

Table 5.2: Speed Table Processing 1 Million Samples

Algorithm Name	Algorithm(train/predict) Runtime	Preprocessing Runtime	Visualization Runtime
Naive Bayes	134.80s	143.28s	161.48s
Logistic Regression	162.30s	136.13s	150.47s
DBSCAN	n/a	139.31s	n/a
SVM	177.20s	149.81s	141.21s

The first Table 5.1 showed the overall speed measurements of the 200k sample data set while the second table 5.2 the speed performance of the 1 million sample set represented. Each row gathered times of a specific detection algorithm. The processing times were separated in algorithm, preprocessing, and visualization runtime.

The time gap between both tables was attributable to the different sizes of the data sets. Algorithm runtime was determined by the sum of training and prediction runtime. Each time referred to the corresponding training and test data sets. Varieties of runtimes

derived from varying characteristics and complexity of the machine learning algorithms. The Naive Bayes and DBSCAM algorithms took one loop over the training set to determine their free parameters. On the contrary Logistic Regression and the SVM algorithms used iterative optimization algorithms to determine the parameters. That meant multiple loops over the training data set. The speed of convergence to a certain error criteria stopped the parameter determination in the case of the other algorithms. Implementation methods contributed to speed performance as far as the use of Spark structures enhanced processing runtimes. Almost every exceptional utilization of regular Scala data structures slowed down processing time. Scala structures affected preprocessing runtimes as well as visualization runtimes drastically in the case of frequent use. Small deviations of preprocessing runtimes also depended on the varying number of MLlib imports. Variations of the treatment of evaluation metrics led to different visualization runtimes. Very accurate algorithm metrics with partly equal values led to unwanted summations during SQL queries and distorted graphs. Time consuming RDD extensions and modifications handled these problems. The extend of time loss depended on the complexity of the issue and amount of applications of non Spark structures.

In the case of the DBSCAN algorithm no runtime could be measured. This was due the implementation which increased the complexity of the algorithm drastically. The inflexible RDD structure caused the implementation of the DBSCAN algorithm. No direct change of data instances in RDDs was possible. Even indexing the data instances in the RDDs did not facilitate accessing possibilities. Capabilities of RDD methods were constrained to aggregation, reduction, looping, and other processing techniques. There was no way of accessing a single instance except of looping through the RDD and filter out the sample. This caused very long processing times. Resulting timeouts finally prevented the algorithm to produce runtime results.

Research of online anomaly detection over online streams[34] or other real time anomaly detection with streams[53] supported the trend of fast large scale anomaly detection using Spark. Spark's advanced DAG execution engine outperforms prefix tree, MapReduce, or other execution engines and delivers very fast anomaly detection results.

The data of the tables indicated Spark to be a suitable solutions for large scale anomaly detection algorithms concerning processing speed. In the case of external developed algorithms RDD methods had to be applicable to the algorithm problem characteristics. Otherwise increasing complexity prevented the detection system to produce results. The results of Spark's algorithms clarified the advantage of Spark structures concerning large scale data processing. Regarding processing runtimes it was beneficial to also utilize Spark's optimized data structures during preprocessing and evaluation calculations to provide a fast large scale anomaly detection process.

5.2 Accuracy

The figures 5.1, 5.2, and 5.3 illustrated some accuracy measurements of the anomaly detection algorithms. Each graph displayed either a receiver operating characteristics(ROC) or precision recall curve of one algorithm.

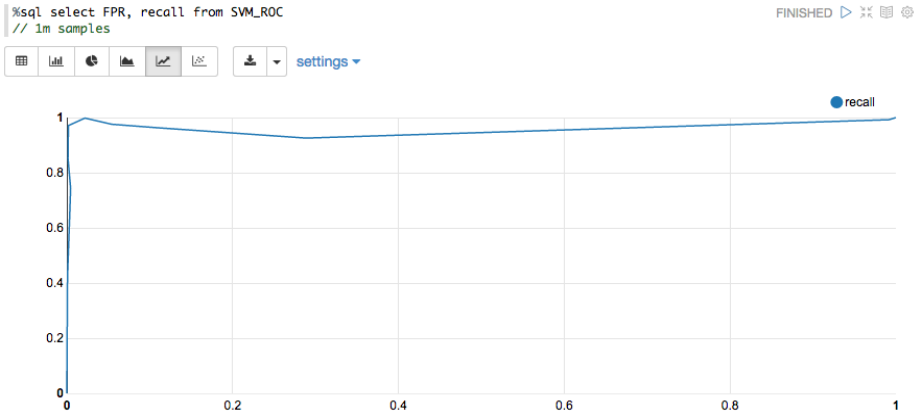


Figure 5.1: SVM ROC Curve 1m Samples

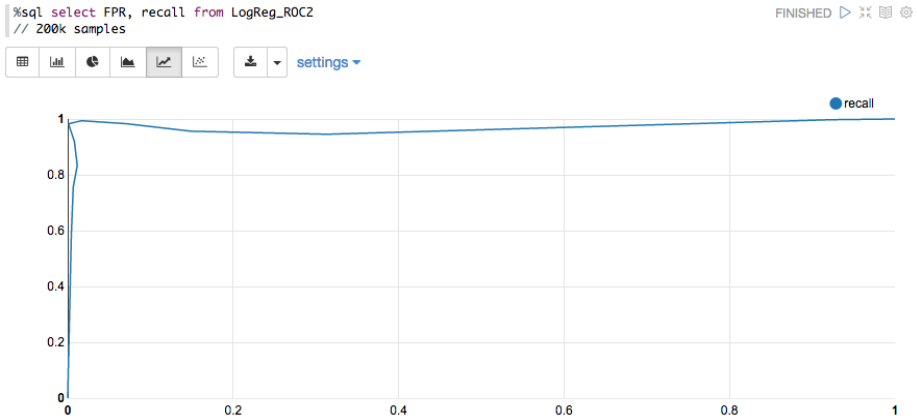


Figure 5.2: Logistic Regression ROC Curve 200k Samples

Generally the size of the area under the ROC curve marked the accuracy of a algorithm. The (0,1) point in classification space of a ROC curve represented the perfect classification. Therefore the closer the ROC curve passed this specific point or the larger the area under the curve the better the accuracy of the algorithm. Algorithms trained by the 1 million sample data sets justified the trend of more accurate ROC curves compared to algorithms trained by 200 thousand sample data sets. The reason for it was the fact that algorithms with more training samples could find more accurate decision boundaries through the parameter optimization process. ROC curves displayed thereby the performance of a binary classification system through plotting the true positive and false positive rates. A better fitting decision boundary affected the precision recall curves as well. Equal to ROC curve characteristics a large area under the precision recall curve marked exact accuracy. A 1.0 precision score meant that every labeled instance indeed belonged to the assigned class. While a 1.0 recall score signified that an item was labeled as belonging to a particular class. In other words precision recall curves delivered information about the amount of selected items that are relevant in regard to the amount of relevant items that are selected. The

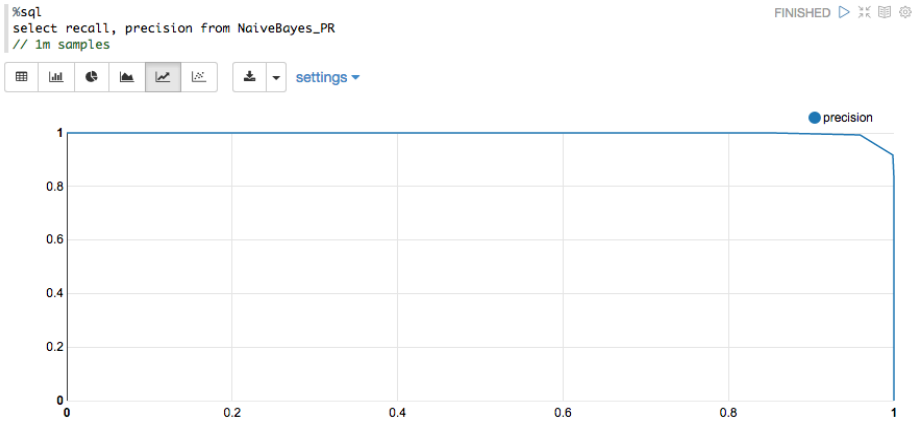


Figure 5.3: Naive Bayes Precision Recall Curve 1m Samples

effort of calculating 10 different ROC and precision recall values gave an accuracy overview over all labeling cases of the input data set. To provide various labeling characteristics over one data set the labeling threshold ranged from marking almost all instances as malicious to defining all items as clean.

Multiple factors were responsible for distinctions of curve shapes between different algorithms. Nevertheless mentioning all factors leading to varying characteristics of the algorithm evaluation curves would go beyond the scope of this thesis. The Naive Bayes and DBSCAN algorithm had one iteration over the training set to train their adjustable parameters. The regularization parameter for overfitting and bias regularization contributed accuracy improvements to the Naive Bayes, Logistic Regression, and SVM algorithms. Next the Logistic Regression and SVM algorithms utilized iterative optimization algorithms for the determination of their adjustable parameters. The SVM algorithm took additional kernel transformation calculations to find better decision boundaries. These advanced optimization algorithms were responsible for the convergence accuracy of the free parameters and set clear advantages to external implemented algorithms without these techniques. Varieties of detection models and their parameter setting allowed distinctive characterization metrics. As a result not all accuracy evaluation graphs reacted equal to the given large scale anomaly detection problem. Findings of other research about learning with kernels, optimization, and regularization supported the depicted trend. More complex and computationally expensive optimization solutions led to more accurate algorithm performance[37]. Unfortunately Zeppelins graph functionality sometimes messed up curve sections where accuracy measurements felt closer together.

The overall findings about algorithm accuracy suggest to utilize Spark's efficiently implemented optimization algorithms for improving accuracy of the anomaly detection model. In most cases more training data led to better detection results which recommends Spark due to the ability of fast large scale data processing. Even without suitable and included machine learning techniques Spark provides the ability to develop accurate solutions to various anomaly detection domains.

5.3 Model Interaction With Spark

Since the Apache Spark project has been in development, Spark's MLlib grew with the development but did not provide a rich variety of machine learning techniques as some established processing engines. The ability of reacting flexible on any kind of anomaly detection problem posed the question of how easy and well Spark admits the interaction of other data structures and algorithms with own structures. The evaluation of the interaction of external structures with Spark focused especially on Spark's most crucial property which is speed.

The anomaly detection procedure required the interaction of external data structures with Spark during preprocessing, model implementation, and evaluation calculations. The initial RDD contained data in form of Spark SQL structures. The conversion of this structure into the demanded Spark MLlib structure kept the outer RDD structure. The inner Spark SQL data structure was transformed into more flexible regular Scala structures. Backwards transformation of the Scala structures into Spark structures followed right after more complex calculations were done. This way it was possible to handle in Spark missing functionality. During implementation of external algorithms Spark structures were usually reduced with RDD methods until the amount of information was small enough to process it fast with common structures. Afterwards RDD structures were initialized and composed in any beneficial situation. Similar to implementation techniques of Spark structures the evaluation calculations mined RDD information until no further processing with Spark structures was possible. Only then structure transformation into visualization structures was executed. The trend of maintaining Spark structures was obvious. Spark structures allowed fast parallel and in memory computation of data. Other research about Spark and anomaly detection was consistent and supported the findings of the evaluation. During processing of big data streams RDD structures were reduced to structures where correlations of data could be detected[34]. Similar to the data mining techniques information was reduced through Spark's methods until further processing with other techniques was possible.

To sum up the interaction of external data structures with Spark's data structures simplified complex data processing and enabled flexible treatment options on limitations coming with Spark structures. Nevertheless the extensive use of Spark's transformation and data processing methods was important. Applying regular Scala functionality already on large scale data sets missed the purpose Spark provides. This would not take advantage of Spark's capability of optimized parallel processing. Knowledge about Spark's structures and methods enabled efficient interaction for a fast final processing result.

6 Discussion

Evaluation of the large scale anomaly detection process in regard to speed, accuracy and interaction possibilities of external data structures with Spark delivered the main findings of the study. Processing results of large scale anomaly detection recommend Spark as a general processing engine. Preprocessing, model implementation, and evaluation calculations were executed with Spark's optimized data structures. The opportunity of interaction with more flexible external data structures was responsible for fast processing in domains where Spark had no pre-build data structures. This enabled flexible solutions and broke limitations. Indeed paying attention to use external processing methods in only suitable cases was necessary to keep fast speed results. In cases where RDD methods do not fit the implementation procedures of algorithms Spark methods can drastically complicate problem solutions. The accuracy measurements of the detection models showed accurate results which further suggested the use of Spark for large scale anomaly detection processes.

Fast interaction of Spark structures with regular Scala structures was possible through the schema to mine information or data with Spark methods, process the smaller amount of data with flexible regular structures and methods, and recompose Spark's optimized structures with the gained results. This schema found application during all three processing stages of the large scale anomaly detection process. The factor of speed optimization justified the application of this technique. The core section of the large scale anomaly detection process was the detection model. The choice of parameters and settings for each algorithm explained the accuracy and speed outcomes. The bernoulli model type facilitated the calculations of the Naive Bayes algorithm. Moreover the Naive Bayes and the DBSCAN algorithms were suitable to large scale data problems because one loop through the training data set determined the free parameters. Regularization parameters and highly optimized optimization algorithms contributed to the large scale capability of the other chosen algorithms. The L-BFGS algorithm improved the large scale capability of the Logistic Regression algorithm[22]. While the SGD algorithm permitted the SVM algorithm to converge to an accurate solution of the adjustable parameters in only 100 iterations[8]. These modifications reinforced large scale suitability of the algorithms.

Nonetheless unavoidable flaws appeared in the current study of the large scale anomaly detection process. Positively the data sets came in different sizes. This enabled to experience and compare the behavior of the algorithm in regard to the varying sizes. However all detection models were classification based. Statistical, information theoretic, or nearest neighbor based anomaly detection techniques were not applied to the data sets. So this study constrained itself and its evaluation results on the actions taken during the large scale anomaly detection procedure. Despite that it was assumable that Spark would be a suitable solution to other large scale anomaly detection domains due to the capability of handling the large scale anomaly detection process of this thesis. In the case of this study a suitable

detection model was found because of diverse implementations of detection models. This was achieved by covering the behavior of Spark concerning model diversity. Although Spark's library packages did not provide a pre-defined solution all problem domains, Spark mastered unexpected upcoming problems through the interaction of external data structures. So even if the data set could not cover the arbitrary forms of raw input data, the study showed how flexible Spark can react. This flexibility allowed the implementation of the DBSCAN algorithm but on the other side RDD methods could not provide suitable functionality to solve the intermediate programming steps of the DBSCAN algorithm efficiently. The reaction of Spark indeed could not prove behavior on even more complex problems. Also the visualization graphs set only one example of evaluation calculation but indicated that Spark could respond to similar issues.

Another point was the non parallel execution of external regular Scala data structures. The bottleneck to processing performance was the interaction of Spark structures with non optimized data structures. Closer programming to Spark's optimization proceedings demanded data patterns could have influenced processing performance immensely.

Generally the implications of the findings of the study do not suggest not to use Spark in regard to anomaly detection processes. On the contrary the large scale processing engine of Spark and the gained results encouraged us to further use Spark in regard to anomaly detection. External developed algorithms highly depend on the functionality RDD methods provide. This leads to attention of method functionality when very specific problem characteristics arise.

7 Limitations

During working on large scale anomaly detection using Spark certain factors affect the performance and implementation of the detection procedure. Even though the large scale anomaly detection process runs these factors block or limit the intuitive implementation approach. This section points out the concept and associated limitations of the thesis.

7.1 Spark's MLlib

The techniques and algorithms of Spark's MLlib do not cover or solve the wide range of large scale anomaly detection problems. The concept of this study uses into Spark's MLlib included algorithms first. The implementation of these algorithms follows the standard description closely. In order to that Spark's MLlib included algorithms limit various treatment options of large scale anomaly detection problems to exactly those techniques Spark's MLlib provides. The techniques of Spark's MLlib do not always provide the optimal solution approach. Especially when it comes to more complex large data processing. With that these techniques slow down detection process drastically if they do not match the required problem treatment methods. There is even no solution to some specific problems.

The algorithms out of Spark's MLlib require particular input structures. Concerning feature extraction, vectorization, and labeling, preprocessing of raw data into the demanded structures slows down and limits the overall performance of the large scale anomaly detection process. The concept realizes the transformation of raw data instances into fitting input structures to MLlib machine learning algorithms. Also when utilizing external algorithms for detecting anomalies the preprocessing schema stays the same due to reuse and the advantages these structures provide with regard to speed.

With the extension of other machine learning algorithms the concept of this thesis tackles the problem of missing techniques in Spark's MLlib. This implies the interaction with Spark's data structures which leads to the next section.

7.2 Spark

Spark's data structures enable fast data processing. For instance the resilient distributed dataset(RDD) allows parallel in memory computation which is probably the most important aspect in large scale data processing. Associated to this a RDD can perform only specific operations and transformations. This means limitations in regard to programming flexibility. To execute a word count or filter a condition the standard methods suffice. But in other cases of more complex operations the regular solutions provided by Spark are not enough.

To handle more complex operations on RDDs the implementations of this study take detours around regular RDD methods. Regular transformation methods access RDDs for temporal use of native Scala data structures. The flexibility of Scala's data structures and methods simplifies processing of complex problems. Spark contains methods allowing backwards transformation of Scala structures for further use of Spark's fast RDD structures. Such ways bypass regular Spark operations but slow down data processing at the end.

Shortly before reaching the deadline of this thesis Spark released a 2.0 version update. Therefore it is interesting to compare and quickly review new features with regard to large scale anomaly detection.

Next to API usability, performance and operational improvements, and other additional upgrades Spark updated the MLlib. The biggest change is the switch to Spark's Data Frame based API as primary API. This means the RDD data structure becomes less important. It is now possible to perform machine learning algorithms on Data Frame data structures which provide more functionality in loading diverse data structures into Spark. The transformation of data variables into RDD structures becomes dispensable. Smaller updates still affect some RDD methods and improve user friendliness. The upgrade of the MLlib brings new clustering machine learning algorithms and extends thereby the flexibility of reacting on various problem characteristics. Concerning large scale data processing Data Frame structures use more efficient serialization and reduce overhead by calling MLlib algorithms more effectively. Thereby Data Frame data structures become more scalable and faster. The general performance improvements further increase the importance of Spark with regard to large scale data processing.

7.3 Virustotal

Virustotal malware detection results in JSON format represent the raw data sets. Processing all data samples into valid input for machine learning reduces the speed of the large scale anomaly detection process. Vectorization, labeling, and adaptation of sample features to a conform length are the factors leading to valid binary input instances for the algorithms.

Filtering RDD collections to receive instances with equal length of features reduces the amount of input data. This impairs each machine learning algorithm in regard to accuracy. Less training and test instances affect the overall performance of a detection model as it hampers an approximation algorithm to fit to the most accurate decision boundary.

8 Future Work

This chapter extends the following points that have been reviewed during the discussion and limitation chapters with suggestions for future research and further optimization:

- The classification based large scale anomaly detection process left space for experiencing other large scale anomaly detection techniques with Spark.
- The improvement of the speed bottleneck of interaction of Spark structures with external data structures could affect extensions to Spark's MLlib and Spark's general flexibility.
- The investigation of missing functionality of specific Spark methods could improve anomaly detection systems implemented with Spark.

8.1 Optimization Suggestion

The first point referred to the study's limitation of covering classification based large scale anomaly detection characteristics. Optimization in the anomaly detection domain could be achieved through testing Spark implemented large scale anomaly detection processes with other data sets. The nature of data in those data sets would have to differ to the data sets in this thesis for provoking a treatment with other anomaly detection techniques. A change of the information structure of the input data implied that Spark's MLlib would have to react and adapt to new tasks as well. With this action other missing techniques or utilities of Spark's MLlib could be revealed and tackled. Future research in the described section would further clear recommendations and the picture of Spark in regard to the wide range of application domains in large scale anomaly detection.

The next point focused on the importance of speed and with it large scale processing capability of large scale anomaly detection procedures with Spark. An improvement of the interaction of Spark with other data structures would affect all stages of analytic life cycles in data processing. Spark would thereby become more flexible in all areas. Add-ons to Spark's MLlib and services could be worth even more as new enhancements simplify tedious circumstances. To achieve new enhancements the schema of mining information and data through Spark methods, transforming or processing the reduced information with external services, and recomposing Spark's optimized structures with obtained information had to become modified. A deeper understanding of required data structure patterns for Spark's optimized processing engine would enable better integration of external services. Scala's

parallel programming functionality enabled preparation and transformation of regular data structures into the desired structure patterns for Spark's processing engine.

Spark inconvenient tasks that required treatment with detours could be taken over by external and more suitable structures. External services came thereby towards Spark for finding better solutions. On the other hand the approach of expanding or upgrading Spark's immutable data structures with more functionality would optimize the interplay between different structure architectures.

The last paragraph already led to the next point of the optimization suggestions section. Editing or extending methods of immutable Spark data structures could lead to improved processing behavior. Suggestions for further development of Spark in regard to anomaly detection systems were the enlargement and upgrading of preprocessing and evaluation methods. Specific methods could react on average input data problems during the acquisition of data. With that more transformations options concerning machine learning data processing was a missing functionality. The only consideration in regard to evaluation computation was to have less automated evaluation functions on machine learning algorithms. It was necessary to evaluate each algorithm on all possible characteristics of the input data set. Therefore varying labeling thresholds provoked diverse information distributions in the data set. Every time a trained algorithm produced an evaluation result through standard Spark methods it surrounded each result with additional values. On the contrary and to display accuracy of an algorithm it was mandatory to manually calculate every algorithm evaluation and surround the final collection of evaluation metrics with missing values for the visualization. This meant only one adjustment concerning visualization. So in the case of evaluation computation, to the core task reduced methods could provide better overall performance to the detection system.

9 Related Works

9.1 Anomaly Detection

The increase of new application domains for anomaly detection systems calls for research about anomaly detection solutions. A lot of research has been done about anomaly detection in general. The first popular research domain classifies application domains and consequently investigates detection techniques of specific problems. The paper of Varun Chandola[9] is an suitable example as it covers the wide range of different detection techniques of anomaly detection. This survey sets a structured and broad overview of anomaly detection components and connects detection techniques to application domains and research areas. Moreover it points out relative strengths and weaknesses of detection techniques.

Animesh Patcha[30] published a paper providing a survey of anomaly detection systems which represents the second popular research domain of anomaly detection. This work compares and investigates overall detection systems. The described detection systems implement all stages of an analytic life cycle. These anomaly detection processes react on certain problem characteristics. Therefore application domains determine the design of the overall procedures. Modular implementation of most systems components enable the evaluation and replacement of detection techniques within the systems.

In relation to the considered research this study is situated between these research domains. Similar to the first paper the detection system compares and evaluates various detection systems. Because of different evaluation goals this study searches for more detailed metrics due to the ability to provide the demanded information. As the detection process takes an experimental approach and has no defined application domain the detection procedure aims to be a flexible solution to any anomaly detection domain. Evaluation criteria and replaceable processing options let the detection model resemble to current research methods on anomaly detection systems.

9.2 Large Scale Data Processing

Effective and smart systems in Big Data processing are necessary for providing decent computation results. Not surprising is the amount of research on Big Data processing systems during recent years since every large company or institution deals with large scale data sets.

An example of research on large scale processing systems is the paper of Aaron N. Richter[35]. This paper analyses large scale processing frameworks and included machine

learning tool kits on speed, fault tolerance, usability, scalability, and extensibility. Spark and the MLlib is under the tested frameworks. Whereas this research paper illustrates general behavior of large scale processing frameworks the study provides a process applied to a specific domain. This means detection models and external extensions receive more detailed treatment.

The papers of Samuel Marchal[23] and Gema Bello-Orgaz[7] mark large scale processing frameworks performing analysis to a specific application domain. Among others Spark is implemented and tested as execution engine of the infrastructures. The first paper about large scale security monitoring relinquishes the use of machine learning algorithms for modeling detection results. By contrast the second paper about social Big Data only evaluates into Spark included structures without any extensions to the framework.

This study applies large scale data processing to a specific domain which in this case is anomaly detection. Additionally it investigates this field with diverse implementation possibilities. The work of the Amrita University[33] resembles this project more than the other large scale data processing investigations. The setup described in this journal consists of Apache Hadoop, Spark and Zeppelin. Large scale data sets are processed through Spark and visualized with the help of Zeppelin. However this work analyses the data sets with only one prediction model. The project of this thesis distinguishes itself due to the more extensive large scale data analysis.

10 Conclusion

The performance of Spark concerning large scale anomaly detection processing is mainly dominated by the bottleneck of interaction between Spark and external data structures. The consequences of this problem affect all processing stages and determine thereby performance metrics. Additionally the question about the quality of solving the bottleneck of interaction answers the investigation of integration of external algorithms into Spark's MLlib. Implicitly this solution quality determines Spark's applicability and flexibility to diverse anomaly detection problems. The behavior of Spark concerning application domains of anomaly detection does not clarify overall suitability due to classification based implementations only. As the contribution for the improvement of missing functionality of specific Spark data structures relies on Spark developers the interaction problem depends on solutions of external structures.

Nevertheless the explorations of Spark with regard to large scale anomaly detection deliver responses to the study's research questions. The fast and accurate implementations of detection models state Spark as an effective processing engine for large scale anomaly detection. The ability of Spark to execute the analytic life cycle from raw data to filtered information supports this statement. Solutions to preprocessing and evaluation calculations can be found easily. Only when it comes to advanced and optimized interaction solutions between Spark and external data structures more effort has to be invested. This means external extensions to the MLlib machine learning algorithms can work together with Spark's code design. Concerning optimization suggestions a deeper understanding of required structure patterns for Spark's optimized processing helps to fasten processing. Variation of data set sizes leads to more optimized and accurate detection models and speed increases respectively to the complexity of the overall anomaly detection system.

The results of the study indicate that Spark can theoretically compete with other large scale processing engines in any application domain. Anomaly detection sets therefore an example. Furthermore with Spark developed processing systems deliver very fast results. Especially when included library solutions match the given problem characteristic.

With included and highly optimized libraries Spark limits itself to solving only specific problem characteristics. External developed and integrated algorithms cannot keep up with the qualities Spark's MLlib algorithms provide. Spark's data structures and affiliated methods are constrained to particular demands only and cannot serve other claims. These points together with the fact of having a single nature of input data prevent the study from proving more statements.

To conclude the developed large scale anomaly detection concept proves that Spark has the ability to provide all necessary components for the implementation of such an analytic life

cycle. Despite a fast and accurate general performance, appearing issues could be primarily eliminated through future research about structure patterns Spark uses for advanced and highly optimized computing.

References

[1] Holmes-Totem/src/main/scala/org/holmesprocessing/totem/services/peinfo at master · HolmesProcessing/Holmes-Totem · GitHub. `https://github.com/HolmesProcessing/Holmes-Totem/tree/master/src/main/scala/org/holmesprocessing/totem/services/peinfo`, July 2016.

[2] Holmes-Totem/src/main/scala/org/holmesprocessing/totem/services/virustotal at master · HolmesProcessing/Holmes-Totem · GitHub. `https://github.com/HolmesProcessing/Holmes-Totem/tree/master/src/main/scala/org/holmesprocessing/totem/services/virustotal`, Feb 2016.

[3] Riak Cloud Storage. `http://docs.basho.com/riak/cs/2.1.1/`, July 2016.

[4] The Apache Cassandra Project. `http://cassandra.apache.org/`, July 2016.

[5] VirusTotal. `https://www.virustotal.com/`, Feb 2016.

[6] Henrik Bäcklund, Anders Hedblom, and Niklas Neijman. A density-based spatial clustering of application with noise,". *Data Mining TNM033*, pages 11–30, 2011.

[7] Gema Bello-Orgaz, Jason J Jung, and David Camacho. Social big data: Recent achievements and new challenges. *Information Fusion*, 28:45–59, 2016.

[8] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[9] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):15, 2009.

[10] Min Chen, Shiwen Mao, and Yunhao Liu. Big data: a survey. *Mobile Networks and Applications*, 19(2):171–209, 2014.

[11] A Cornuejols and M Moulet. Machine learning: A survey. *Knowledge-Based Systems: Advanced Concepts, Techniques and Applications*, pages 61–86, 1997.

[12] Douglas Crockford. Introducing json. *Available: json. org*, 2009.

[13] Jesse Davis and Mark Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning*, pages 233–240. ACM, 2006.

[14] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[15] The Apache Software Foundation. Welcome to Apache Hadoop. `http://hadoop.apache.org/`, July 2016.

[16] The Apache Software Foundation. Zeppelin. `http://zeppelin.apache.org/`, July 2016.

[17] Henri Gilbert and Helena Handschuh. Security analysis of sha-256 and sisters. In *International Workshop on Selected Areas in Cryptography*, pages 175–193. Springer, 2003.

[18] Prasanta Gogoi, B Borah, and DK Bhattacharyya. Anomaly detection analysis of intrusion data using supervised & unsupervised approach. *Journal of Convergence Information Technology*, 5(1):95–110, 2010.

[19] Apache Hadoop. Hadoop, 2009.

[20] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 59–72. ACM, 2007.

[21] Karl-Rudolf Koch. Bayes' theorem. In *Bayesian Inference with Geodetic Applications*, pages 4–8. Springer, 1990.

[22] Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989.

[23] Samuel Marchal, Xiuyan Jiang, Radu State, and Thomas Engel. A big data architecture for large scale security monitoring. In *2014 IEEE International Congress on Big Data*, pages 56–63. IEEE, 2014.

[24] Andrew McCallum, Kamal Nigam, et al. A comparison of event models for naive bayes text classification. In *AAAI-98 workshop on learning for text categorization*, volume 752, pages 41–48. Citeseer, 1998.

[25] Richard McMahon, Martin S Bressler, and Linda Bressler. New global cybercrime calls for high-tech cyber-cops. *Journal of Legal, Ethical and Regulatory Issues*, 19(1):26, 2016.

[26] Scott Menard. *Applied logistic regression analysis*. Number 106. Sage, 2002.

[27] Xiangrui Meng, Joseph Bradley, B Yuvaz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, D Tsai, Manish Amde, Sean Owen, et al. Mllib: Machine learning in apache. *JMLR*, 17(34):1–7, 2016.

[28] Naive Bayes Model. Naive bayes algorithms.

[29] Derek G Murray, Malte Schwarzkopf, Christopher Smowton, Steven Smith, Anil Madhavapeddy, and Steven Hand. Ciel: a universal execution engine for distributed data-flow computing. In *Proc. 8th ACM/USENIX Symposium on Networked Systems Design and Implementation*, pages 113–126, 2011.

[30] Animesh Patcha and Jung-Min Park. An overview of anomaly detection techniques: Existing solutions and latest technological trends. *Computer networks*, 51(12):3448–3470, 2007.

[31] Matt Pietrek. Inside windows-an in-depth look into the win32 portable executable file format. *MSDN magazine*, 17(2), 2002.

[32] Inc Pivotal Software. RabbitMQ: Persistence Configuration. `https://www.rabbitmq.com/persistence-conf.html`, Feb 2016.

[33] M Prathilothamai, AM Sree Lakshmi, and Dilna Viswanthan. Cost effective road traffic prediction model using apache spark. *Indian Journal of Science and Technology*, 9(17), 2016.

[34] Laura Rettig, Mourad Khayati, Philippe Cudré-Mauroux, and Michal Piórkowski. Online anomaly detection over big data streams. In *Big Data (Big Data), 2015 IEEE International Conference on*, pages 1113–1122. IEEE, 2015.

[35] Aaron N Richter, Taghi M Khoshgoftaar, Sara Landset, and Tawfiq Hasanin. A multi-dimensional comparison of toolkits for machine learning with big data. In *Information Reuse and Integration (IRI), 2015 IEEE International Conference on*, pages 1–8. IEEE, 2015.

[36] Robert E Schapire. The boosting approach to machine learning: An overview. In *Nonlinear estimation and classification*, pages 149–171. Springer, 2003.

[37] Bernhard Schölkopf and Alexander J Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2002.

[38] Kai Shen. Challenges of big data processing.

[39] Armin Shmilovici. Support vector machines. In *Data Mining and Knowledge Discovery Handbook*, pages 257–276. Springer, 2005.

[40] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)*, pages 1–10. IEEE, 2010.

[41] Apache Spark. Apache spark^{TM}-lightning-fast cluster computing, 2014.

[42] Apache Storm. Storm, distributed and fault-tolerant realtime computation. 2014.

[43] Ronald C Taylor. An overview of the hadoop/mapreduce/hbase framework and its current applications in bioinformatics. *BMC bioinformatics*, 11(Suppl 12):S1, 2010.

[44] Mark Veugelers, Jo Bury, and Stijn Viaene. Linking technology intelligence to open innovation. *Technological forecasting and social change*, 77(2):335–343, 2010.

[45] Željko Vrba, Pål Halvorsen, Carsten Griwodz, Paul Beskow, Håvard Espeland, and Dag Johansen. The nornir run-time system for parallel programs using kahn process networks on multi-core machines—a flexible alternative to mapreduce. *The Journal of Supercomputing*, 63(1):191–217, 2013.

[46] George Webster. Holmes Processing. `http://holmesprocessing.github.io/`, Feb 2016.

[47] George Webster, Zachary Hanif, Andre Ludwig, Tamas Lengyel, Apostolis Zarras, and Claudia Eckert. SKALD: A Scalable Architecture for Feature Extraction, Multi-User Analysis, and Real-Time Information Sharing. In *19th Information Security Conference (ISC)*, September 2016.

[48] Ian H Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.

[49] Reynold Xin, Parviz Deyhim, Ali Ghodsi, Xiangrui Meng, and Matei Zaharia. Graysort on apache spark by databricks. *GraySort Competition*, 2014.

[50] Rui Xu and Don Wunsch. *Clustering*, volume 10. John Wiley & Sons, 2008.

[51] Nong Ye et al. A markov chain model of temporal behavior for anomaly detection. In *Proceedings of the 2000 IEEE Systems, Man, and Cybernetics Information Assurance and Security Workshop*, volume 166, page 169. West Point, NY, 2000.

[52] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. Technical report, Technical Report UCB/EECS-2011-82, EECS Department, University of California, Berkeley, 2011.

[53] Wenke Zhang, Favyen Bastani, I-Ling Yen, Kevin Hulin, Farokh Bastani, and Latifur Khan. Real-time anomaly detection in streams of execution traces. In *High-Assurance Systems Engineering (HASE), 2012 IEEE 14th International Symposium on*, pages 32–39. IEEE, 2012.

[54] Xiaojin Zhu and Zoubin Ghahramani. Learning from labeled and unlabeled data with label propagation. Technical report, Citeseer, 2002.